SheLL: Shrinking eFPGA Fabrics for Logic Locking

Hadi M Kamali, Kimia Z Azar, Farimah Farahmandi, Mark Tehranipoor
Department of Electrical and Computer Engineering, University of Florida, Gainesville, FL, USA.
{h.mardanikamali, k.zamiriazar}@ufl.edu, {farimah, tehranipoor}@ece.ufl.edu

***Abstract*—The utilization of fully reconfigurable logic and routing modules may be considered as one potential and even provably resilient technique against intellectual property (IP) piracy and integrated circuits (IC) overproduction. The embedded FPGA (eFPGA) is one instance that could be used for IP redaction leading to hiding the functionality through the untrusted stages of the IC supply chain. The eFPGA architecture, albeit reliable, unnecessarily results in exploding the die size even while it is supposed to be at fine granularity targeting small modules/IPs. In this paper, we propose SheLL, which primarily embeds the interconnects (routing channels) of the design and secondarily twists the minimal logic parts of the design into the eFPGA architecture. In SheLL, the eFPGA architecture is customized for this specific logic locking methodology, allowing us to minimize the overhead of eFPGA fabric as possible. Our experimental results demonstrate that SheLL guarantees robustness against notable attacks while the overhead is significantly lower compared to the existing eFPGA-based competitors.**

***Index Terms*—Logic Locking, eFPGA, IP Redaction.**

I. Introduction

The ever-increasing rate of globalization in the semiconductor supply chain has raised big concerns over the trustworthiness of the IC supply chain, which results in emerging threats like IP piracy and IC overproduction [1]. Logic locking as a proactive IP protection solution can promise robustness if implemented appropriately. In logic locking, the designer will add programmability into the design using additional gates, known as key gates, and the input to these key gates, i.e., key input, which is only known to the trusted parties, will recover the correct functionality. Logic locking received significant attention over the last decade. However, the endless cat-and-mouse game between attacks and defenses in logic locking shows low reliability of many of these techniques [2].

One particular breed of logic locking solutions relies on the complexity of reconfigurable routing and logic modules using basic reconfigurable cells, i.e., multiplexers (MUXes) and look-up-tables (LUTs) [3]–[5]. These techniques can build strong robustness against the widely-used attacks on logic locking, e.g., Boolean satisfiability (SAT) attack and its variants [6], [7]. However, they are not structurally flawless, thus, allowing machine learning (ML) based attacks to break them [8], [9]. These reconfigurable-oriented logic locking solutions resemble the FPGA structures in which the bitstream serves as the configuration of MUXes/LUTs. Hence, few recent studies have extended this concept by utilizing the embedded FPGA (eFPGA) as the source of secrecy for post-manufacturing programmability. In such solutions, the designer will *redact* part(s) of the design by replacing them with eFPGA fabric(s) to add full reconfigurability for security purposes [10]–[12].

By mostly relying on the open-source eFPGA fabric generators and synthesizers, such as OpenFPGA [13], the existing eFPGA-based redaction solutions select part(s), module(s), or IP(s) of a system, and will replace them with the counterpart FPGA fabric generated, in which the corresponding bitstream recovers the functionality and serves as the secret, resembling the key of logic locking [10]–[12]. Both reconfigurable route&logic locking solutions [3]–[5] and eFPGA-based IP redaction techniques [10]–[12] follow almost the same attributes, such as (i) the use of symmetric structures for building the redacted logic (route&logic), (ii) inclusion of cyclical blocks in the wiring of the locked part(s), (iii) cascading logic and/or routing blocks, and (iv) large size of the secret configuration, and by possessing such attributes, these techniques resemble the attributes of universal circuits that meet preliminary formal requirements needed for the future of logic locking, specifically indistinguishability (IND-LL) [14].

Although the engagement of fully reconfigurability might be suited enough to meet the formal requirements, they incur striking and mostly prohibited area overhead, which makes them less applicable for resource-constrained devices [15], [16]. A recent design space exploration for attack resilience vs. area overhead on eFPGA-based solutions shows that even while identifying the part(s)/module(s)/IP(s) to be embedded into the FPGA has been done heuristically [12], the overhead would be at-least 2x [11], which brings down the usability of such approaches for the designers.

In this paper, by revisiting the utilization of eFGPA for IP redaction, we introduce SheLL, a framework for reconstructing and shrinking the building blocks/tiles of the selected eFPGA fabric in multiple ways, allowing us to have a customized fabric for IP protection purposes at much more acceptable overhead ratio. In SheLL, we also introduce a specific form of logic locking that is suited the most for mapping and placement on the eFGPA fabric. The SheLL framework enables the designer to get the benefit of routing clusters of eFPGA as a means of routing-based locking solution, and the framework is built in a way to maximize the utilization of the eFPGA resources. Our contributions are as follows:

(1) We revisit the eFPGA fabric generation/synthesis in open-source eFPGA automation tools, i.e., OpenFPGA [13] and FABulous [19]. We retouch the main steps to maximize resource utilization and minimize the tiles needed for redaction.

(2) We utilize FABulous [19] using the Skywater *130nm*, with better standard cell-based optimizations, which significantly improve the overhead ratio.

978-3-9819263-7-8/DATE23/© 2023 EDAA

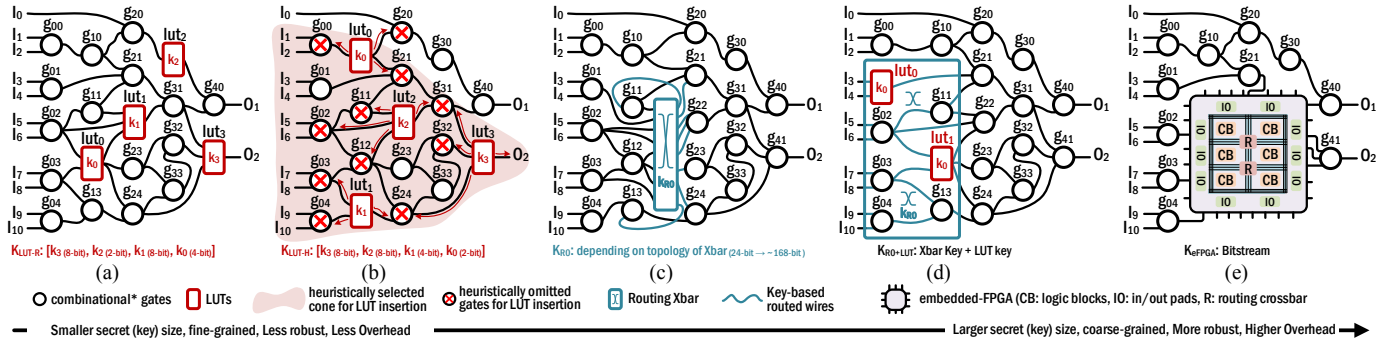

Fig. 1: Logic Locking Relying on Reconfigurability. (a) Traditional (Random) LUT insertion [17], (b) Heuristic LUT insertion [18], (c) MUX-based Routing Locking [3], (d) MUX+LUT Routing+Logic Locking [4], [5], (e) eFPGA-based IP Redaction [10]–[12]. (*: Circuit could consist of sequential elements (FFs).)

(3) We introduce a routing-oriented, yet logic-based, locking solution that fits with the lowest overhead ratio on the eFPGA fabric architecture, built using the FABulous framework. The routing-based mapping and placement are on the MUX chains of eFPGA, which saves the space in eFPGA-based redaction. (4) We comparatively evaluate the robustness as well as the overhead of SheLL vs. that of the existing (re)-configurable-based (mostly eFPGA-based) solutions.

II. Background and Prior Art

Reconfigurable logic was first used in [17] to prevent IP/IC piracy. Over the last decade, this breed has been extended through different avenues, from LUT-based logic locking to eFPGA-based IP redaction, as described below.

A. Reconfigurability for Locking

(i) Traditional LUT insertion [17]: As shown in Fig. 1(a), LUTs will be replaced with gates (random/test-based gate-to-LUT mapping), and LUTs' configuration serves as the secret (key). This method is already broken using the SAT attack [6].
(ii) Heuristic LUT insertion [18]: To be resilient against the SAT attack, this method uses logic-level (e.g., De Morgan's law with no back-to-back LUT as shown in Fig. 1(b)) and topological features (e.g., fanout counts and observability/controllability) for LUT-based locking. However, to achieve (SAT) robustness, large numbers of LUTs may be needed for designs, which degrade the efficiency in terms of overhead.
(iii) MUX-based Routing Locking: In [3], the relation between robustness and *clause-to-variable* (c2v) ratio[1] has been engaged for introducing SAT hard instances for logic locking purposes. This study proposes a cascaded MUX-based routing locking (for hiding the connectivity of the cells in the design), which has a close resemblance to the FPGA routing. However, since it is localized as shown in Fig. 1(c), they are vulnerable and eventually breakable by a specific ML-based attack [8].
(iv) MUX+LUT Routing+Logic Locking: As its name implies, in [4], [5], a cascaded set of MUXes has been twisted with LUTs for logic locking purposes. As shown in Fig. 1(d), this LUT+MUX twisting has a spirit similarity with the FPGA architecture, in which part of the configuration (key) is for logic (LGC) and the other part is for the routing and connectivity (ROUTE)[2]. All the attributes of this sub-group are in line with eFPGA redaction techniques.
(v) Embedded FPGA for Locking: This is the most complete version of reconfigurability that tries to hide the real functionality of the selected design portions (either LGC or ROUTE) behind a fully reconfigurable logic array, i.e. eFPGA fabric [10]–[12], shown in Fig. 1(e). The previous studies on eFPGA redaction are more related to the fabric sizes and architectures [10], [11], rather than investigating the module that must be redacted [12]. Compared to these techniques, **the main aim of the proposed SheLL is threefold:** (1) The redaction can be accomplished in a way that the eFPGA fabric is utilized the most (the highest utilization ratio), (2) the redacted part can be selected more ROUTE-based than LGC-based to minimize the number and the dependency of tiles, (3) non-cyclical MUX chains of the eFPGA fabric can be used as a lucrative resource for ROUTE-based locking. **The proposed SheLL not only considers the size, architecture, and module selection for the eFPGA redaction but also introduces a new mapping for eFPGA automation customized for locking.**

As shown in Fig. 1, from left to right, the robustness has an increasing rate. However, it has been achieved with a much larger key size, at much more coarse granularity, and excessive (almost prohibited) overhead. A design space exploration on LUT-based logic locking demonstrates that the overhead of a robust solution can go easily above 2x [20]. Similarly, another design space exploration on eFPGA-based IP redaction shows the worst trend for the overhead [11]. Although inefficient in terms of overhead, studies with more focus on the formalism of logic locking show that reconfigurability could be promising enough for a long life of logic locking. For instance, a recent study focuses on the definition of two main attributes (notions) needed for the formalism of the logic locking [14], which are (i) IND-LL and (ii) SIM-LL. Amongst these notions, IND-LL, which is recently revealed as one of the main requirements of logic locking [2], means that the adversary cannot get benefit from the structural differences between locked vs. original

[1]The SAT attack [6] input is in conjunctive normal form (CNF), which consists of clauses and variables, and the hardness of SAT solving can be related to the attributes of the CNF, e.g. clause-to-variable (c2v) ratio.

[2]LGC and ROUTE stand for the logic cells (gates) and routing wires and switches of the design, respectively. In Fig. 1, all g_{ij}s are LGC and all wires between g_{ij}s as well as all wires between I/Os and g_{ij}s are ROUTE.

circuit, and it is a more generic definition of being symmetric and distributed as concluded by [3], which is met by the use of reconfigurability [14] for locking purposes.

B. Threat Model

In this study, we follow the same threat model used in the previous eFPGA-based redaction solutions, i.e., the oracle-guided attack on logic locking. In this model, the attacker has access to the reverse-engineered (locked) netlist, the activated chip (oracle) with a fully-scanned architecture. The attacker is able of pinpointing the eFPGA fabric in the locked netlist and the location of bitstream storage cells (FFs). Since the SheLL framework is an eFPGA-based solution, an attack is successful when they can restore the correct bitstream of the eFPGA.

III. eFGPA Automation: Revisiting for Locking

Custom eFPGA modules can be created using the recently-published open-source FPGA fabric generation and synthesis automation frameworks, such as OpenFPGA [13] and FABulous [19]. Almost all existing eFPGA-based IP redaction techniques have used OpenFPGA for the automation of redaction. In OpenFPGA, the main steps are LUT-based synthesis and Fabric mapping that are done using Yosys and VPR/VTR, respectively. However, the OpenFPGA framework is not optimized efficiently either logically or physically, leading to an unnecessarily large fabric size with prohibited overhead: (i) the square-like eFPGA fabric may contain significant tiles that are not utilized. For instance, as demonstrated in Fig. 2, for an arbitrary design, named `desX.v` mapped on a 7x7 eFPGA fabric, 11 out of 49 tiles are not utilized ($\leq$77% utilization). (ii) The attributes of a significant portion of routing fabrics, switch multiplexers, and interconnection wires do not improve the robustness. For instance, a significant portion of wiring in eFPGA may add (combinational) cyclical blocks. Since the targeted module is usually acyclic, this can be tracked and ruled out by the attacker as a helpful pre-processing before running any form of attack. (iii) OpenFPGA does not use custom and the smallest possible bounding boxes for CLBs and switching multiplexers that are the most contributors to the eFPGA fabric. Also, OpenFPGA does not support MUX chains for modeling the logic, and in such cases, both LGC and ROUTE will be mapped to the tiles (CLBs), which significantly increases the fabric size needed for the redaction.

Considering that eFPGA in this application is for locking purposes (redaction), one can define the logic locking (redaction) in a way that such inefficiencies will be omitted through

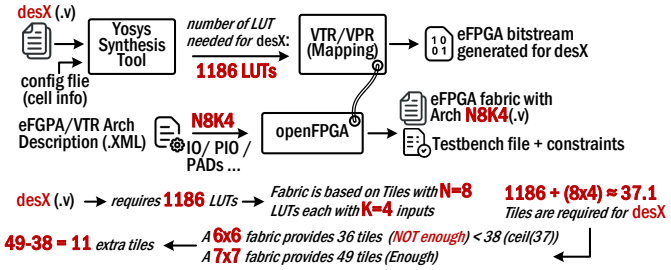

Fig. 2: Inefficient Mapping in OpenFPGA for eFPGA Fabrics.

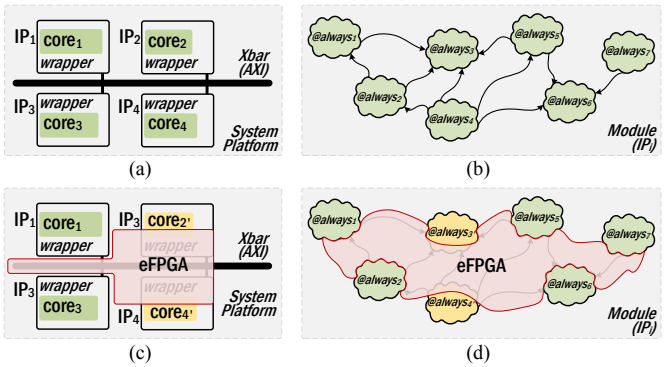

Fig. 3: How SheLL Locks. (a) Original System Platform, (b) *@always* Parts of an IP, (c) Locked System Platform, (d) Locked IP.

the eFPGA automation. With the least modification to the existing eFPGA automation flow, the main aim of the proposed SheLL is to mitigate such impactful inefficiencies using (1) a retouched automation flow for eFPGA-based redaction, (2) a new module selection methodology for the redaction part, and (3) using a different open-source eFPGA automation flow with higher physical (and also logical) optimization.

IV. Proposed Scheme: SheLL

The ultimate goal of the SheLL framework is to demonstrate that, for eFPGA-based redaction, it is more beneficial for the designers to focus more on ROUTE to be mapped into the eFPGA than LGC. Hence, we introduce a new module selection for redaction in the SheLL framework, whose big pictures have been demonstrated in Fig. 3. Unlike the other eFPGA-based solutions, SheLL, with a more efficient approach w.r.t. the overhead, enables the eFPGA-based redaction at both IP-level and SoC-level with more fine granularity. In SheLL (for SoC-level), most (but not all) part of the customized eFPGA fabric is dedicated to hiding the inter-IP interconnection (ROUTE), and a small part of the eFPGA resources is dedicated to the logic (LGC) redaction. Fig. 3(a) shows a typical system platform consisting of multiple IPs and the crossbar (*Xbar*). Running the SheLL framework on this example will produce the circuit depicted in Fig. 3(c). In this case, as a system-level solution, eFPGA is used for constructing the AXI-based *Xbar* between IPs. Additionally, a small (LGC) part of $core_2$ and $core_4$ and their wrappers will be mapped to the eFPGA fabric. The *Xbar* is a simple memory-addressed MUX-based arbitration between multiple AXI channels (ROUTE). This will be combined with a minimal logic (LGC) related to $core_2$ and $core_4$ to make the redacted part robust against removal-based attack (in which the adversary can replace the whole eFPGA with a AXI-based simple Xbar). Also, the minimal LGC selected from $core_2$ and $core_4$ is close (neighboring) to PI/PO of the AXI channel (ROUTE). In our experiment, we show that selecting the LGC close to ROUTE will minimize the dependency and overhead of tiles needed for redaction.

Fig. 3(b) and 3(d) shows how the SheLL framework works at IP-level description of the design. In this case, the SheLL framework is able to target *@always* blocks or *module* instan-

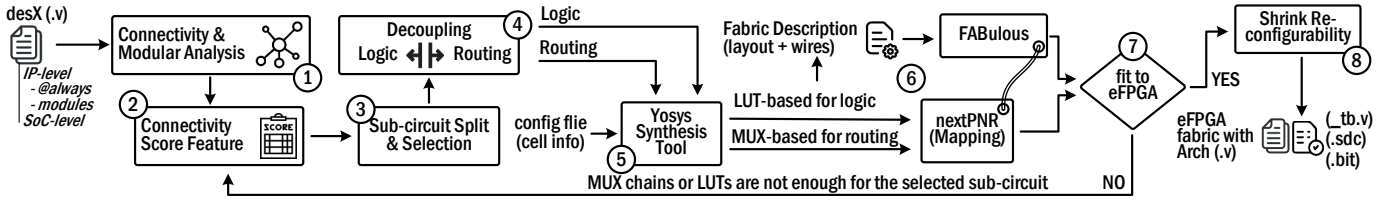

Fig. 4: SheLL Framework Proposed Flow

tiations. However, the main target is the connection between *module*s or *@always* blocks. Fig. 3(b,d) theoretically shows how it works on *@always* blocks. In this case, the signals transceiving (ROUTE) between different *@always* blocks and a set of direct logic to these transfers will be mapped to the eFPGA. Also, to select the LGC to be redacted, we applied a straightforward connectivity analysis to select the LGC leading to better propagation (corruptibility). For instance, in Fi. 3(d), a part of LGC dedicated to $@always_3$ and $@always_4$ (with more connections) will be selected for redaction.

In SheLL, via FABulous, we utilize MUX-based chains with custom cells for modeling and implementing the routing channels (ROUTE). It significantly reduces the number of LUTs and logic tiles required for mapping, leading to having a smaller fabric (lower overhead) for larger redaction. A simple comparison between the efficiency of OpenFPGA and FABulous (with and without MUX chain use) has been demonstrated in Table I ($\geq$ 50% improvement with custom MUX chain [21]). In FABulous, when the ROUTE is built using MUX chains, the required resources for generating the fabric are significantly reduced (compared to the non-MUX chain model in either FABulous or OpenFPGA). In line with this observation, SheLL focuses primarily on redacting ROUTE rather than LGC, resulting in better fabric utilization.

TABLE I: Resource Utilization for a ROUTE circuit (8 AXI channels Xbar).

Tool	Multiplexer	Flip Flop	Latch
OpenFPGA [13]	1650 M2s	650 DFFs	–
FABulous (std[3] cell)	560 M4s + 80 M2s	20 CFFs	650
FABulous (std cell w/ mux chain)	185 M4s + 63 M2s	12 CFFs	431

M2: Mux2 M4:Mux4 CFF: Custom FF

Fig. 4 shows the main (8) steps of the SheLL framework:
(1) Connectivity and Modular Analysis: Given the circuit to be locked (e.g., desX.v), the SheLL framework first builds the graph-based (connectivity) representation of the design. For the SoC level, it will be done more coarse-grained by analyzing inter-IP communications. For the IP level, it is more fine-grained by evaluating wiring between main structures like main sub-modules or the *@always* blocks. For either SoC-level or IP-level, after simply flattening and uniquifying the design, we utilize FIRRTL [22] for building an intermediate representation for easier to-graph conversion.
(2) Connectivity Score Feature and Sorting: After generating the graph-based representation from FIRRTL, we extract and use a set of graph-based (more focus on centrality measures [23]) as well as circuit-based attributes for the nodes as

[3]Custom cells are also defined in the FABulous framework through an iterative optimization (up to 30% die size shrinkage) for MUX-chain structure.

TABLE II: Graph/Circuit-based Measures in the proposed SheLL Framework.

Attribute	Detail	Coeff.	Objective
iDgC	Degree of the node (inlet/outlet)	α	High
oDgC	Degree of the node (outlet)	β	High
ClsC	Closeness to the observable/controllable nodes of the graph through the shortest path	γ	Low
BtwC	Node occurrence in the shorting paths between observable/controllable nodes	λ	Low
EigC	Neighboring node(s) type (gate type)	ξ	High
LuTR	LUT-based resource needed for the node	σ	Low

iDgC: inlet Degree Centrality **oDgC**: outlet Degree Centrality
ClsC: Closeness Centrality **BtwC**: Betweenness Centrality
EigC: Eigen Centrality **LuTR**: LUT-based Resource Utilization

demonstrated and elaborated upon in Table II. By defining a coefficient for each attribute, a score function will be defined as shown in Eq. 1. In the experimental results, we demonstrate how sweeping these coefficients affects the efficacy of the SheLL framework. Also, as one of the graph nodes attributes, we estimate the LUT needed per each node (based on the gate/module type)[4]. It leads to selecting the best logic around the routing that fits in the eFPGA fabric (best utilization).

$$score = \alpha.\mathbf{iDgC} + \beta.\mathbf{oDgC} + \gamma.\mathbf{ClsC} + \lambda.\mathbf{BtwC} + \xi.\mathbf{EigC} + \sigma.\mathbf{LUTR} \quad (1)$$

(3) Sub-circuit Selection: Based on the attributes, SheLL follows simple rules for selecting the best sub-circuit to be mapped on the selected eFPGA fabric (defined as an objective in Table II): (i) Nodes with the highest inlet/outlets are the best choice for routing-based locking (high **iDgC/oDgC**), (ii) Selected nodes should cover (indirect connection) a good portion of the design nodes ($\geq$ 50% node coverage) with lower observability/controllability (low **ClsC/BtwC**), (iii) The (estimated) sum of LUTs needed for the logic associated with the selected node must fit in the eFPGA fabric available resource (low **LuTR**), and (iv) Per each node selected with high inlet/outlet, a small (but generic) logic must be involved (high **EigC**). A big portion of the sub-circuit selected after this step is from ROUTE accompanied by a small part of LGC.
(4) Decoupling Logic and Routing: For the selected sub-circuit, the SheLL framework decouples the sub-circuit associated with the ROUTE from the sub-circuit associated with the LGC. Since sub-circuit selection (step 3) operates on connections and the logic around them (Fig. 3(a/c) and 3(b/d)), this decoupling will create two detached logic (LGC) and routing (ROUTE) sub-circuits.
(5) Yosys Synthesis: In SheLL, we call Yosys twice for the synthesis. One is for the synthesis of LGC which must be

[4]Instead of estimation, it can be done accurately using LUT-based Yosys synthesis per each neighboring node/module. However, for making the flow less time-consuming, we created an offline (estimated) database.

mapped to the LUTs of CLBs. In this case, synthesis is done based on LUT-based (tree of MUXes) mapping. The second synthesis is for ROUTE which must be mapped to MUX chains (not LUTs). The FABulous framework allows us to map these two sub-circuits separately using nextPNR [19].

(6) eFPGA Creation and Mapping: Once the synthesized (LUT-based LGC and MUX-based ROUTE) sub-circuits are ready, we use FABulous and nextPNR for generating the eFPGA fabric and mapping these synthesized sub-circuits into the CLBs and MUX chains, respectively. The fabric size will be determined based on the estimated LUTs, MUXes, and other resources required for the synthesized sub-circuits.

(7) eFPGA Fit Check: Although the eFPGA fabric size will be determined based on the estimation of all resources required for ROUTE and LGC subcircuits, it is possible that the eFPGA will not be large enough to map the subcircuits due to routing, mapping, and placement constraints. SheLL switches back to step 6 to select a larger fabric if it does not fit[5].

(8) Shrinking Reconfigurability and Size: Once the FABulous successfully maps the ROUTE and LGC sub-circuits to the fabric, the SheLL framework will shrink the reconfigurability based on the generated bitstream. In this step, part of the resources (MUX chains, LUTs, and FFs) not used for ROUTE and LGC sub-circuit will be removed (physically) based on the bitstream value. This is for reducing the possibility of applying any form of pre-processing for guessing the key values without any attack (e.g., removal of combinational stateful cycles [11]).

V. Experimental Results and Evaluation

To evaluate the efficiency and resiliency of ROUTE-oriented yet LGC-based redaction via the SheLL framework, we targeted a RISC-V-based SoC as well as a few individual IPs listed in Table III. We utilized both OpenFPGA [13] and FABulous [19] for generating eFPGA fabrics using open Skywater 130nm process [24]. The SheLL framework is built in Python, and the Verilog-related scripting around the eFPGA tool (e.g., decoupling ROUTE from LGC) has been done using PyVerilog framework [25]. The experiments include synthesis, verification, and physical design, which were carried out using Cadence Genus, Jaspergold, and Innovous, respectively (using the Skywater 130nm std library).

First, to show how the SheLL framework applies the redaction efficiently, we compared four different scenarios: (i) no-strategy redaction using OpenFPGA [10], [11], (ii) module/cluster filtering-based redaction using OpenFPGA [12], (iii) no-strategy redaction using FABulous, (iv) SheLL (ROUTE-oriented yet LGC-based) using FABulous. Table IV shows the targeted sub-circuit(s) for redaction (TfR) and the normalized overhead comparison in terms of area, power, and delay (A/P/D) for these scenarios. In SheLL, the sign "a→b" means the connection (ROUTE) between a/b is targeted for redaction. Please note that all these fabrics are tested using cyclic-reduction+SAT attack [26], and with a timeout of 48 hours, none of them were broken. According to these numbers, compared to other scenarios, SHeLL reduces the overhead by 53% (no-strategy in OpenFPGA), 55% (module/cluster filtering-based in OpenFPGA), and 67% (no-strategy in FABulous), respectively, in terms of area, power, and delay. The delay overhead improvement specifically shows how MUX chains of eFPGA fabric can be used for ROUTE with less configurable cells. Also, if the TfRs of cases 1-3 were similar to that of case 4, this improvement was better (see Table V).

In steps (2-3) of the proposed framework, we engage a set of attributes (graph-based and circuit-based) for the best sub-circuit selection. Table VI shows why the objectives for these attributes are defined as listed in Table II (particularly for minimizing the overhead). In this study, for 6 attributes (iDgC, oDgC, ClsC, BtwC, EigC, and LUTR), being high/low (h/l) for the coefficient means the *best/worst choice w.r.t. the attribute*. For instance, high LUTR means that for LGC the sub-circuit with the highest required LUT estimation will be selected. As shown, {h,h,l,l,h,l} can achieve the best results that are used as objectives in SHeLL (Table II). Future work will explore these attributes more quantitatively and more heuristically (e.g., use of (M)ILP, GA, or ML)

One crucial constraint that must be followed in the SheLL framework is that LGC must be close (direct connection) to ROUTE. This constraint also significantly contributes to the overhead reduction. Table VII shows different scenarios in terms of correlation between LGC and ROUTE. As shown, once the LGC and ROUTE are not directly correlated (non-neighboring parts), eFPGA routing and placement faces a huge overhead due to the back-and-forth inlet/outlet (+ extra pins) required for this form of mapping. But, once they are connected, it minimizes the routing fabric within the eFPGA for these two parts (i.e., between CLBs and ROUTEs).

[5]Based on the output log of the eFPGA automation tool (FABulous and NextPNR), the type of shortage (resources type) will be determined, and the SheLL framework scripts it to expand the fabric size as needed.

TABLE III: Specifications of the selected Benchmark Circuits.

Benchmark	Description	# of Modules	# of Input Pins	∼ # of Output Pins
PicoSoC	Size-Optimized RISC-V CPU	12	8-64	8-96
AES	AES Encryption/Decryption	11	16-128	16-128
FIR	Finite Impulse Response Filter	7	32-128	16-128
SPMV	Sparse Matrix Vector Multiplication	16	8-32	8-64
DLA	Lightweight DLA-like Accelerator	4	64-256	64-256

VI. Conclusion and Future Work

This paper presented SHeLL, an overhead-efficient automatic framework for eFPGA-based IP/SoC redaction. In SHeLL, routing-oriented yet logic-based parts of the design are analyzed and selected for redaction using FABulous eFPGA automation framework and MUX chains. This significantly reduces the overhead (compared to other eFPGA-based redaction techniques) while the resiliency is still guaranteed by benefiting from eFPGA fabric. Our flow shows how eFPGA optimization could be done specifically for security (redaction) purposes. The SHeLL framework defines a set of topological attributes of the circuit that directly affect the overhead of eFPGA-based redaction, whose further investigation can lead to a suited fine-grained redaction at acceptable overhead.

TABLE IV: Comparative (Normalized) Overhead in eFPGA-based IP Redaction (All cases are Resilient against SAT (Timeout set to 48 hours)).

Benchmark	Case 1 [10], [11]: No-Strategy via OpenFPGA				Case 2 [12]: Filtering via OpenFPGA				Case 3: No-Strategy via FABulous				Case 4: Proposed SheLL (ROUTE then LGC) via FABulous			
	TfR	A	P	D	TfR	A	P	D	TfR	A	P	D	TfR	A	P	D
PicoSoC	`/_mem_wr`	1.74	1.95	2.11	`/_mem_wr` + `/_regs_rdata`	1.87	1.97	2.28	`/_mem_wr` + `/_regs_rdata`	1.71	1.88	1.94	`/_mem_wr`→`picorv32.mem_wr` + `/_mem_wr_en`	1.39	1.45	1.47
AES	`/_addround_last`	2.11	2.34	3.15	`_addround_last` + `/_shrow_last`	2.07	2.33	3.25	`_addround_last` + `/_shrow_last`	1.98	1.94	2.22	`/_key_sch`→`top.addround` + `/_addround_xor`	1.38	1.51	1.55
FIR	`/_ternary_add`$_i$	2.97	3.11	4.02	`/_ternary_add`$_i$ + `/_ctrl_valid`	3.17	3.21	4.14	`/_ternary_add`$_i$ + `/_ctrl_valid`	2.89	2.99	3.23	`/_mult`$_j$ →`_add`$_i$ + `/_ctrl_valid`	1.66	1.77	1.82
SPMV	`/_ind_array_inc`	1.57	1.73	2.61	`/_ind_array_inc` + `/_len_check`	1.76	1.88	2.74	`/_ind_array_inc` + `/_len_check`	1.94	2.03	2.88	`/_mult`$_j$ →`_sum` + `/_len_check`	1.36	1.41	1.52
DLA	`/_active_check`	1.41	1.57	2.34	`/_active_check` + `/_drain_PE`	1.55	1.72	2.66	`/_active_check` + `/_drain_PE`	1.60	1.74	2.44	`/_DDR`$_j$ →`_PE`$_j$ + `/_max_pool_valid`	1.29	1.33	1.40

TfR: Targeted for Redaction A/P/D: Normalized Area/Power/Delay Overhead

TABLE V: Comparative (Normalized) Overhead in eFPGA-based IP Redaction with Same Target (ROUTE-based) for Redaction (All cases are SAT-Resilient).

Benchmark	Case 1 [10], [11]			Case 2 [12]			Case 3			Case 4: Proposed		
	A	P	D	A	P	D	A	P	D	A	P	D
PicoSoC	1.993	2.162	2.674	1.994	2.161	2.676	1.756	2.036	2.214	1.390	1.447	1.473
AES	2.505	2.814	3.450	2.505	2.814	3.450	2.274	2.470	2.715	1.384	1.509	1.548
FIR	3.251	3.50	4.68	3.421	3.559	4.697	3.31	3.57	3.82	1.663	1.768	1.816

Case 1 and 2 are equal as they use both OpenFPGA with no change.

TABLE VI: Attributes Used for Sub-Circuit Selection (TfR).

Benchmark	$\{\alpha, \beta, \gamma, \lambda, \xi, \sigma\}$														
	c_1:{l,l,l,l,h,l}			c_2:{h,h,h,h,h,l}			c_3:{h,h,l,l,l,l}			c_4:{h,h,l,l,h,h}			c_5:{h,h,l,l,h,l}		
	A	P	D	A	P	D	A	P	D	A	P	D	A	P	D
PicoSoC	1.58	1.59	1.97	~~1.41~~	~~1.58~~	~~1.45~~	1.42	1.46	1.46	1.81	1.93	1.99	1.39	1.45	1.47
AES	1.64	1.77	1.88	~~1.55~~	~~1.61~~	~~1.77~~	~~1.43~~	~~1.46~~	~~1.60~~	2.24	2.36	2.77	1.38	1.51	1.55
FIR	~~1.88~~	~~2.01~~	~~2.06~~	~~1.75~~	~~1.79~~	~~1.99~~	~~1.65~~	~~1.69~~	~~1.94~~	2.33	2.50	2.94	1.66	1.77	1.82
SPMV	1.66	1.70	1.83	1.36	1.41	1.64	1.35	1.42	1.58	1.77	1.78	2.08	1.36	1.41	1.52
DLA	1.36	1.45	1.59	1.31	1.32	1.55	1.38	1.53	1.95	1.58	1.64	2.09	1.29	1.33	1.40

h: high l:lowc c_1:low degree c_2:high closeness/betweenness
c_3:low eigen (masking gate type [and/or]) c_4:high LUT c_5: SHeLL choices
~~strikethrough~~ cells: Cases broken using the SAT attack.

TABLE VII: LGC/ROUTE correlation and its Impact on Overhead.

Benchmark	Indirect (depth 2)			Indirect (depth 1)			SHeLL: Direct (depth 0)		
	A	P	D	A	P	D	A	P	D
PicoSoC	2.717	2.957	4.621	2.640	2.928	4.311	1.390	1.447	1.473
AES	3.180	3.347	5.174	3.215	3.451	5.318	1.384	1.509	1.548
FIR	3.554	3.701	5.138	3.439	3.766	5.082	1.663	1.768	1.816

depth: Node-based Distance between LGC and ROUTE.

REFERENCES

[1] M. Rostami *et al.*, "A Primer on Hardware Security: Models, Methods, and Metrics," *Proc. of the IEEE*, vol. 102, no. 8, pp. 1283–1295, 2014.

[2] H. M. Kamali *et al.*, "Advances in Logic Locking: Past, Present, and Prospects," *Cryptology ePrint Archive, 2022/260*, 2022.

[3] H. M. Kamali *et al.*, "Full-lock: Hard Distributions of SAT Instances for Obfuscating Circuits using Fully Configurable Logic and Routing Blocks," in *Design Automation Conf. (DAC)*, 2019, pp. 1–6.

[4] J. Sweeney *et al.*, "Modeling Techniques for Logic Locking," in *Int'l Conf. On Comp. Aided Design (ICCAD)*, 2020, pp. 1–9.

[5] H. M. Kamali *et al.*, "InterLock: An Intercorrelated Logic and Routing Locking," in *Int'l Conf. On CAD (ICCAD)*, 2020, pp. 1–9.

[6] P. Subramanyan *et al.*, "Evaluating the Security of Logic Encryption Algorithms," in *Int'l Symp. on Hardware Oriented Security and Trust (HOST)*, 2015, pp. 137–143.

[7] K. Z. Azar *et al.*, "Smt attack: Next generation attack on obfuscated circuits with capabilities and performance beyond the sat attacks," *IACR Trans. on CHES*, pp. 97–122, 2019.

[8] L. Alrahis *et al.*, "UNTANGLE: Unlocking Routing and Logic Obfuscation using Graph Neural Networks-based Link Prediction," in *Int'l Conf. On Comp. Aided Design (ICCAD)*, 2021, pp. 1–9.

[9] K. Z. Azar *et al.*, "NNgSAT: Neural Network guided SAT Attack on Logic Locked Complex Structures," in *Int'l Conf. On Comp. Aided Design (ICCAD)*, 2020, pp. 1–9.

[10] P. Mohan *et al.*, "Hardware Redaction via Designer-Directed Fine-grained eFPGA Insertion," in *Design, Automation & Test in Europe Conf. & Exhibition (DATE)*, 2021, pp. 1186–1191.

[11] J. Bhandari *et al.*, "Exploring eFPGA-based Redaction for IP Protection," in *Int'l Conf. On Comp. Aided Design (ICCAD)*, 2021, pp. 1–9.

[12] C. Tomajoli *et al.*, "ALICE: An Automatic Design Flow for eFPGA Redaction," in *Design Automation Conf. (DAC)*, 2019, p. 781–786.

[13] X. Tang *et al.*, "OpenFPGA: An Opensource Framework enabling Rapid Prototyping of Customizable FPGAs," in *Int'l Conf. on Field Programmable Logic and Applications (FPL)*, 2019, pp. 367–374.

[14] P. Beerel *et al.*, "Towards a Formal Treatment of Logic Locking," *Cryptology ePrint Archive*, 2022.

[15] A. Sehgal *et al.*, "Management of Resource Constrained Devices in the Internet of Things," *IEEE Communications Magazine*, vol. 50, no. 12, pp. 144–149, 2012.

[16] A. Musaddiq *et al.*, "A Survey on Resource Management in IoT Operating Systems," *IEEE Access*, vol. 6, pp. 8459–8482, 2018.

[17] A. Baumgarten *et al.*, "Preventing IC Piracy using Reconfigurable Logic Barriers," *IEEE Design & Test*, vol. 27, no. 1, pp. 66–75, 2010.

[18] H. M. Kamali *et al.*, "Lut-lock: A novel LUT-based Logic Obfuscation for FPGA-bitstream and ASIC-hardware Protection," in *IEEE Computer Society Symp. on VLSI (ISVLSI)*, 2018, pp. 405–410.

[19] D. Koch *et al.*, "FABulous: An Embedded FPGA Framework," in *Int'l Symp. on Field-Programmable Gate Arrays (FPGA)*, 2021, pp. 45–56.

[20] G. Kolhe *et al.*, "Security and Complexity Analysis of LUT-based Obfuscation: From Blueprint to Reality," in *Int'l Conf. On Comp. Aided Design (ICCAD)*, 2019, pp. 1–8.

[21] K. L. Chung *et al.*, "Optimizing Tile Interfaces and the Configuration Logic in FABulous FPGA Fabrics," in *ACM/SIGDA Int'l Symp. on Field-Programmable Gate Arrays (FPGA)*, 2022, pp. 13–23.

[22] A. Izraelevitz *et al.*, "Reusability is FIRRTL ground: Hardware construction languages, compiler frameworks, and transformations," in *Int'l Conf. On Comp. Aided Design (ICCAD)*, 2017, pp. 209–216.

[23] S. Borgatti *et al.*, "A graph-theoretic perspective on centrality," *Social networks*, vol. 28, no. 4, pp. 466–484, 2006.

[24] Skywater Technology, "SkyWater Open Source PDK," https://github.com/google/skywater-pdk, 2020.

[25] S. Takamaeda-Yamazaki, "Pyverilog: A python-based hardware design processing toolkit for verilog hdl," in *Int'l Symp. on Applied Reconfigurable Computing*, 2015, pp. 451–460.

[26] J. Bhandari *et al.*, "Not All Fabrics Are Created Equal: Exploring eFPGA Parameters For IP Redaction," *arXiv preprint arXiv:2111.04222*, 2021.