# ARIANNA: An Automatic Design Flow for Fabric Customization and eFPGA Redaction

LUCA COLLINI and JITENDRA BHANDARI, New York University, USA CHIARA MUSCARI TOMAJOLI, Politecnico di Milano, Italy ABDUL KHADER THALAKKATTU MOOSA, New York University, USA BENJAMIN TAN, University of Calgary, Canada XIFAN TANG, Rapid Flex, USA PIERRE-EMMANUEL GAILLARDON, University of Utah, USA RAMESH KARRI, New York University, USA CHRISTIAN PILATO, Politecnico di Milano, Italy

In the modern global Integrated Circuit (IC) supply chain, protecting intellectual property (IP) is a complex challenge, and balancing IP loss risk and added cost for theft countermeasures is hard to achieve. Using embedded configurable logic allows designers to completely hide the functionality of selected design portions from parties that do not have access to the configuration string (bitstream). However, the design space of redacted solutions is huge, with trade-offs between the portions selected for redaction and the configuration of the configurable embedded logic. We propose ARIANNA, a complete flow that aids the designer in all the stages, from selecting the logic to be hidden to tailoring the bespoke fabrics for the configurable logic used to hide it. We present a security evaluation of the considered fabrics and introduce two heuristics for the novel bespoke fabric flow. We evaluate the heuristics against an exhaustive approach. We also evaluate the complete flow using a selection of benchmarks. Results show that using ARIANNA to customize the redaction fabrics yields up to  $3.3 \times$  lower overheads and  $4 \times$  higher eFPGA fabric utilization than a one-fits-all fabric as proposed in prior works.

CCS Concepts: • Hardware  $\rightarrow$  Methodologies for EDA; • Security and privacy  $\rightarrow$  Hardware security implementation; Hardware reverse engineering.

## **ACM Reference Format:**

Luca Collini, Jitendra Bhandari, Chiara Muscari Tomajoli, Abdul Khader Thalakkattu Moosa, Benjamin Tan, Xifan Tang, Pierre-Emmanuel Gaillardon, Ramesh Karri, and Christian Pilato. 2025. ARIANNA: An Automatic Design Flow for Fabric Customization and eFPGA Redaction. *ACM Trans. Des. Autom. Electron. Syst.* 1, 1, Article 1 (January 2025), 22 pages. https://doi.org/10.1145/3737287

Authors' Contact Information: Luca Collini, lc4976@nyu.edu; Jitendra Bhandari, jb7410@nyu.edu, New York University, New York, New York, USA; Chiara Muscari Tomajoli, chiara.muscari@mail.polimi.it, Politecnico di Milano, Milano, Italy; Abdul Khader Thalakkattu Moosa, at4856@nyu.edu, New York University, New York, New York, USA; Benjamin Tan, benjamin.tan1@ucalgary.ca, University of Calgary, Calgary, Alberta, Canada; Xifan Tang, xifan@rapid-flex.com, Rapid Flex, San Jose, California, USA; Pierre-Emmanuel Gaillardon, pegaillardon@cade.utah.edu, University of Utah, Salt Lake City, Utah, USA; Ramesh Karri, rkarri@nyu.edu, New York University, New York, New York, USA; Christian Pilato, christian.pilato@polimi.it, Politecnico di Milano, Milano, Italy.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org. © 2025 Copyright held by the owner/author(s). Publication rights licensed to ACM.



Fig. 1. FPGA redaction flow. Critical modules are replaced with custom eFPGA implementations.

## 1 Introduction

Securing hardware Intellectual Property (IP) is a crucial concern during the design and production of an Integrated Circuit (IC) [1]. With the multi-billion dollar investments required for cutting-edge manufacturing plants, many design houses are forced to outsource IC fabrication to external foundries. This situation has enormous security implications as an unscrupulous employee can steal the IC design to reverse engineer it and make illegal copies [2]. Design houses can use several techniques, like watermarking, split manufacturing, and logic locking, to safeguard their blueprints [1]. However, these strategies are not foolproof: watermarking is a *passive* method aimed to identify an IP theft after it has occurred [3]; split manufacturing requires high manufacturing expertise [4]; and logic locking is susceptible to a wide array of security breaches [2, 5], especially when the attacker can access a working chip (called *oracle*).

**FPGA redaction** is an innovative, promising method to counter reverse engineering attempts. This approach proposes substituting critical portions of the design with specially designed reconfigurable blocks, also known as *embedded FPGAs* (eFPGAs). The dual objectives of this technique are: (1) during fabrication, masking the ultimate functionality of the reconfigurable block behind the ambiguity of its reconfigurable nature; (2) during execution, implementing the original functionality by loading the correct bitstream.

Figure 1 shows an example where a module is replaced by a custom eFPGA fabric. Within this fabric, each unit represents a Configurable Logic Block (CLB). Cutting-edge tools for FPGA specialization (e.g., OpenFPGA [6] and FABulous [7]) allow engineers to map an HDL module into a soft eFPGA IP. This resultant IP can be seamlessly combined and synthesized with the chip's remaining components. The resilience of this FPGA-obfuscation method against SAT attacks is attributed to the need to decode a large set of "key bits" —essentially the complete configuration stream of the eFPGA— through the intricate interactions between input and output within the eFPGA structure [8, 9]. Additionally, these tailor-made eFPGAs present a lower overhead than standard, commercially available options [9, 10].

The process of FPGA redaction involves multiple stages for designers. Initially, they must determine which modules are optimal for redaction, considering both security implications and design perspectives. Following that, the task is to design and assimilate the tailored eFPGA fabric into the system. These challenges are intricately linked and frequently vary based on the specific application. Given the complexities, designers typically address these issues manually, which can result in suboptimal solutions [10, 11].

Previous work [12] focused on the EDA problem of *partitioning RTL modules* between eFPGA and ASIC and *creating the proper eFPGA fabrics* to implement the redacted modules. While the malicious foundry can retrieve modules implemented in ASIC, the flexibility of eFPGAs protects the redacted modules. In prior work [12], the eFPGA fabric configuration was Manuscript submitted to ACM

## ARIANNA: An Automatic Design Flow for Fabric Customization and eFPGA Redaction

kept fixed for all redacted designs, resulting in solutions that required more resources than necessary. The parameters for eFPGA fabrics are numerous, and the design space is vast, making its exploration a hard problem. In this work, we propose **ARIANNA** (Automatic eFPGA RedactIon with fAbric coNfiguratioN And module clustering), a **complete flow to identify the modules to be redacted, optimize the eFPGA fabric, and generate the corresponding chip design augmented with soft eFPGAs**. ARIANNA performs a progressive refinement of the solution by identifying candidate modules for redaction and clustering them to enable the creation of larger eFPGAs, whose fabric configurations are explored and characterized to select the best final implementation that minimizes the hardware overhead without sacrificing security. ARIANNA is built on top of the ALICE framework [12]. While ALICE focuses on identifying the modules for redaction and generating their corresponding soft eFPGAs with a pre-defined configuration, ARIANNA extends the ALICE approach by **defining the ultimate set of secure fabrics** and **fine-tuning the design-specific eFPGA fabric** to minimize the hardware cost. Our contributions can be summarized as follows:

- We build upon the ALICE framework to include heuristics for **eFPGA fabric parameter optimization** for hardware efficiency, obtaining a more holistic and fine-grained approach to hardware IP protection (Section 5.4).
- We present a **security evaluation** of eFPGA fabric as a pre-step to identify the configurations to explore with the novel fabric tailoring heuristics (Section 5.1, Section 6.1).
- We validate the proposed heuristics through extensive benchmarks against an exhaustive approach, showing their efficacy and efficiency (Section 6.2).
- We validate the complete framework on a set of benchmarks to show the efficacy of our **fine-tuned eFPGA fabrics** in **reducing the overheads** of hardware IP protection (Section 6.3).

This work effectively bridges the gap between module selection and eFPGA fabric optimization by focusing on **tailoring the eFPGA fabric**. Designers now have a more comprehensive toolset to consider functional characteristics, structural attributes, and eFPGA parameters, facilitating a more robust and efficient redaction process.

The rest of the paper is organized as follows. Section 2 introduces the background notions on custom eFPGA design flows and related architectures. In Section 3, we provide an overview of related works on IP protection with embedded FPGAs and attacks against them, while the threat model is defined in Section 4. In Section 5, we present the proposed framework for eFPGA redaction, while in Section 6 we evaluate the proposed heuristics for bespoke fabrics and the complete framework. In Section 7, we make our final comments and takeaways on our work.

## 2 Fundamentals on Embedded FPGAs

This section briefly overviews (embedded) FPGAs, emphasizing significant elements such as architectural alternatives and open-source toolchains that aid in versatile hardware development techniques. These facets are critical for creating tailored eFPGA solutions, especially for IP redaction. For a more in-depth exploration of FPGA architectures, we recommend consulting the comprehensive study by Boutros and Betz [13].

#### 2.1 FPGA Architectures

FPGAs are flexible architectures capable of being reprogrammed "on-site" (in the field) to manifest a particular digital design. Modern FPGAs utilize a tile-based architecture, consisting of recurrent tiles and a "sea" of routing resources, as depicted in Figure 2 ( $\mathbf{0}$ ). An *N*×*N* architecture means there are *N* tiles distributed in horizontal and vertical directions, respectively. configurable logic block (CLB) tiles are predominant in an FPGA and implement logic functions (both combinational and sequential). Modern FPGAs can also include specialized tiles, such as block RAM (BRAM) or digital Manuscript submitted to ACM



Fig. 2. A general FPGA architecture and its constituent parts.

signal processing (DSP) tiles, for storing data on-chip and performing efficient arithmetic operations. A heterogeneous tile-based FPGA allows designers to satisfy design needs while managing the aspects of power, performance, and area (PPA) within the architecture. Tile-based architectures present a more favorable balance between programmability and efficiency relative to other options [13]; this structure enables designers to concentrate individually on the challenges of routing and connecting signals inside a tile and the issue of "globally" interlinking tiles. Consequently, engineers can prioritize the optimization of a tile's layout, reducing the time spent on the placement and routing of tiles.

Figure 2 (**2**) details a CLB tile. It contains a CLB and different blocks for setting the connection between signals within and outside the tile. Refer to Figure 2 (**3**) for an in-depth view of the CLB structure. It comprises *N* basic logic elements (BLEs), interconnected via a local routing system. Each BLE is the basic unit for logic operation and encompasses a look-up table (LUT), a flip-flop (FF), and a 2-input multiplexer, as depicted in Figure 2 (**3**). A single-output Boolean function with *K* inputs can be mapped onto an LUT with the same number of inputs.

By configuring a 2-input multiplexer, a BLE can operate in combinational or sequential mode. To route interconnect CLB inputs and BLE inputs and outputs, the local routing architecture, typically implemented as a crossbar, includes a set of programmable multiplexers. The local routing guarantees that BLEs can be fully connected to each other and to every CLB input pin. To optimize the hardware cost, in this work, we focus on the structure of the CLB. The capability of a CLB is defined by these factors: (1) the input dimension of the LUTs, denoted as K; (2) the quantity of BLE within a CLB, represented as N; and (3) the total input count for the CLB, labeled I. The selection of these parameters is influenced by the balance between logic capability and its effects on size, timing, and power consumption. To have better resource utilization in a CLB, for any LUT size,  $I = \frac{K(N+1)}{2}$  has been shown to give good PPA [14]. For these reasons, we explore N and K as the two parameters of the eFPGA fabric configuration in this work.

The FPGA is programmed using a *bitstream*, in which each bit dictates certain fabric components, like routing setups and LUT data. The bitstream can be uploaded using frame-based methods [15] or scan-chain-based methods [16, 17]. For our research, we concentrate on the scan-chain based bitstream loading. In this approach, the complete bitstream is input sequentially, with each clock cycle loading one bit, utilizing a specific clock designated for this task.

# 2.2 Custom eFPGA Design Flow

Reconfigurable devices can implement any specific function by loading the proper configuration bitstream. In the case of hardware security, this post-manufacturing adaptability is crucial for safeguarding hardware intellectual property block (IP)s. Designers can embed the FPGAs into ASIC designs as ready-made blocks, with only the end-user handling Manuscript submitted to ACM

ARIANNA: An Automatic Design Flow for Fabric Customization and eFPGA Redaction



Fig. 3. ARIANNA leverages a state-of-the-art eFPGA design flow based on OpenFPGA [6]. The eFPGA netlist is integrated with the rest of the chip, while the configuration bitstream is kept secret.

their configuration. As a result, the function executed by the FPGA remains undisclosed to the manufacturing facility that cannot access the correct configuration bitstream.

Recently, open-source eFPGA prototyping tools are becoming increasingly popular [15, 18, 19]. These platforms facilitate the automated adaptation of FPGA structures, specially designed for distinct modules, encompassing the entire Verilog-to-bitstream process. For instance, Figure 3 illustrates the customization process based on OpenFPGA, suitable for eFPGA redaction [9]. OpenFPGA utilizes an XML-defined fabric parameter to generate the associated eFPGA IP ready for fabrication [6, 9, 20, 21]. The modules set for modification influence the eFPGA customization<sup>1</sup>. Leveraging open-source platforms provides designers with enhanced flexibility, allowing adjustments to many parameters, as highlighted in prior work [22]. This empowers users to devise architectures optimally tailored to the intended design. When integrated closely with processors on a unified chip, these structures can function as adaptable accelerators or co-processors [8, 10, 23, 24]. This integration can enhance a System-on-Chip (SoC) peak performance by up to 3.4× while reducing power consumption by approximately 2.9×.

Our study delves into FPGA structures characterized by varying parameters K and N within CLBs to obtain the most cost-efficient structure for redacting the given design. However, we can accommodate any fabric setup or even explore other parameters similarly, as our primary emphasis is on their application for redaction rather than their creation or security assessment.

#### 3 Motivation and Related Work

The protection of hardware IPs has become a major focus in recent years. Researchers proposed many methods, especially at low levels of abstraction (i.e., on gate-level netlists or physical designs, or directly during fabrication [25, 26]. For example, logic locking assumes the attacker cannot retrieve the correct functionality thanks to the protection of a "secret", the locking key [27]. Despite many advances [28–30], SAT attacks [31] and machine learning attacks [32–35] can be used to identify the I/O relationships and retrieve key bits when an activated chip is available, challenging the effectiveness of logic locking [2, 5].

FPGA redaction is a recent technique that aims to implement selected modules with soft or hard eFPGAs that are included in the design. The key idea is that (1) attackers in the foundry have no access to the bitstream configuration that can implement any possible functionality, while (2) end-user attackers that have access to an activated chip cannot

<sup>&</sup>lt;sup>1</sup>For a complete overview of the OpenFPGA flow, please see the tool documentation.

retrieve the correct bitstream. In this case, the "secret" corresponds to the configuration bitstream. While eFPGA redaction is considered more secure than logic locking, the design of FPGA-redacted ICs is complex, especially in the module partitioning between eFPGA and ASIC. Moreover, eFPGA redaction comes with higher overhead costs than logic locking techniques. This work proposes a complete flow that helps designers find a feasible module combination for eFPGA redaction, minimizing the overheads.

While recent studies focused on VLSI challenges of eFPGA integration [36], selecting the modules to be redacted is still a manual effort or requires at least a reference design. In the former case, designers have to identify the modules to be protected, for example, because they are part of the core business [8]. Designers may want to use FPGA redaction to protect the results of selected outputs with FPGA redaction without knowing the critical components. In the latter case, two or more designs are compared with each other to identify common parts (which are assumed to be common to many other designs) and different parts (which are the unique parts of the given design) [11]. However, designers may not have an alternative version of the same design to be compared with.

SheLL [37] is a framework proposed to reduce redaction overheads. In SheLL authors point out how the OpenFPGA framework can lead to suboptimal solutions with unutilized tiles and propose the use of the Fabulous[15] for redaction, together with a custom flow for mapping the logic to be redacted onto the LUTs. In this work, we explore the OpenFPGA fabric customization to tailor the eFPGA architecture to the redacted modules to save overheads. In reference [38], Sathe et al. highlight how the fabric choice for eFPGA redaction can drastically affect overheads without impacting security.

In reference [39], Rezaei et al. proposed two attacks to break eFPGA redaction. The DIP exclusion attack is based on the idea of excluding input patterns that lead sequential SAT attacks like CycSAT [40] to get stuck until the attack runs successfully. Brake & Unroll works by first breaking the simple combinational cycles and then unrolling hard cycles. The results show that the attacks are successful only for the simpler eFPGA fabric configurations In reference [41], Karmakar et al. propose variations of SAT attacks targeted to break eFPGA redaction. They validate their attack on the HeLLO CTF 2022 benchmarks, succeeding on small and medium benchmarks. These attacks all rely on the assumption that the attacker will gain full scan-chain access to break down the circuit into combinational logic cones.

FuncTeller [42] is an attack that aims to recover the redacted logic's functionality. The recovered functionality can then be synthesized for the eFPGA fabric under attack to obtain a bitstream. The attack aims at finding minterms for the ON-sets by performing a smart exploration of primary implicants. While results show that the success of FuncTeller does not depend on the eFPGA fabric configuration, the retrieved functionality is only approximate. FuncTeller also requires full scan chain access to have a black box view of all the combinational parts of the design. MANTIS [43] is a machine-learning-based attack on FPGA redaction that allows the retrieval of approximated bitstreams (10-20% error rates). It does not require full scan chain access, but it still requires scan chain access to the eFPGA I/Os.

Recent studies on the security of FPGA redaction show that the resilience to SAT attacks is correlated more with the eFPGA fabric configuration and its utilization rather than the implemented module(s) [8, 9, 22]. Moreover, from the before-mentioned attack studies, it emerges that it is possible to identify a subset of eFPGA fabrics that have shown to be resilient from reverse engineering attacks.

#### 4 Threat Model

The different attacks against eFPGA redaction discussed in Section 3 have in common the following aspects of the threat model: oracle access with full (limited for MANITS [43]) scan-chain access and isolation of the eFPGA fabric. Moreover, for all attacks but FuncTeller and MANTIS, a subset of secure fabrics (complex enough that SAT-based attacks are unfeasible) is identifiable. We assume that designers looking to protect their IPs will also protect the scan chain Manuscript submitted to ACM



Fig. 4. ARIANNA flow for automatic eFPGA-based redaction.

either by blowing fuses after testing or by using a secure scan chain protection technique such as [44]. Adopting such a countermeasure will make all the attacks mentioned in Section 3 unfeasible. Our work aims to explore the design space of secure solutions (those that stand against state-of-the-art attacks) to identify good solutions in terms of I/O and CLB utilization and area/power overheads. Secure fabric parameters can be identified by looking at the state of the art or by performing an analysis beforehand, as we present in Section 5.1.

## 5 ARIANNA Design Flow for eFPGA Redaction

Our proposed framework aims to aid the partitioning of RTL designs for eFPGA redaction by exploring module cluster configurations and tailoring the best-fitting fabric for each cluster. Figure 4 shows the proposed redaction flow. It starts from a behavioral Verilog<sup>2</sup> RTL. A YAML configuration file provides parameters such as the file containing available eFPGA fabric configurations, the number of eFPGAs to use for redaction, and the maximum number of I/O pins for each eFPGA. The maximum number of I/O can be used to bind the module I/O to the maximum I/O size of the largest admissible eFPGA fabric. For instance, a 4×4 fabric configuration has no more than 64 I/O pins [8, 9]. We support one or more eFPGAs with heterogeneous fabric configurations. Our ARIANNA flow is composed of three main phases:

- (0) Secure Fabric Identification: This pre-step needs to be performed offline once to identify a set of secure fabric parameters on which to perform design space exploration. Alternatively, one can look at the state-of-the-art to identify these parameters.
- Clustering Flow: This phase first identifies candidate redaction modules by filtering out modules that do not meet I/O constraints or do not affect signals of interest. It then creates clusters of candidate modules that can fit into the same eFPGA, which we refer to as *candidate module clusters*.
- (2) **Bespoke Fabric Flow**: This phase tailors a bespoke eFPGA fabric for each candidate module cluster. It identifies the fabric configuration that can host the candidate cluster yielding the lowest area overhead.

<sup>&</sup>lt;sup>2</sup>Limitations are only due to the HDL parser we use. Supporting another HDL language (e.g., VHDL) only requires the proper parser.

(3) **eFPGA Selection Flow**: This phase selects the eFPGAs (a number decided by the user) that best meet our objective (maximize redacted logic within our area constraint) with no overlapping modules.

Ultimately, we integrate the selected eFPGAs into the design, replacing the redacted instances with the corresponding eFPGA ones and rerouting the signals as needed. The final design and the eFPGA netlists can be given to physical design tools.

## 5.1 Secure Fabric Identification

This pre-step needs to be performed offline once to identify a set of secure fabric parameters on which to perform design space exploration. Alternatively, one can look at the state of the art to identify these parameters. A paradigmatic study on the security of fabric configurations was conducted in [22].

In this step, the designers need to identify the current state-of-the-art attacks that could be used to evaluate the security of the fabrics. Each fabric configuration needs to be evaluated against the identified attacks. This step is time-consuming but only needs to be performed once and eventually updated if new attacks are proposed. In general, the eFPGA fabrics that are compromised in this analysis will be excluded from the subsequent steps and the others can be explored and used in the rest of the flow.

The results of this step might show that some fabric configurations are secure only above certain eFPGA sizes. If that is the case, these configurations should be considered in the step and discarded at the end only if the yielded size is not secure. An ensemble of multiple and diverse attacks can be used in this phase to identify the most resilient ones. However, in some cases, the designers can decide to keep some unsafe solutions with additional security measures. For example, suppose a final solution result is susceptible to SAT attacks but has minimal overhead. The designers might keep it and integrate a secure scan chain protection technique like DisORC [44] to protect against SAT attacks. Obviously, the overall overhead should be considered when evaluating the solution.

#### 5.2 Module Filtering

This stage is dedicated to analyzing the initial design to determine which RTL modules are candidates for redaction. The pseudocode for this procedure is outlined in Algorithm 1. It starts with the initial RTL design D, a set of eFPGA parameters P (i.e., the upper limit on the number of I/O pins), and a list of target outputs O. The algorithm then employs two distinct sets of criteria—*functional* and *structural*—to determine the ultimate list R of modules to be redacted. *Functional* criteria aim to identify those modules that are more important for FPGA redaction from the functionality viewpoint, like the modules that directly affect the outputs in O. On the other hand, *structural* criteria aim to spot modules suitable for eFPGA implementation while eliminating those that would render the design unworkable, like modules that exceed the I/O limit in P. This balanced approach ensures that the selected modules are crucial for the desired functionality and compatible with the constraints and capabilities of eFPGA technology.

The algorithm begins by looking at the functional criteria. We enumerate the modules M in the initial design D (line 1), setting a starting score of zero for each (lines 3-5). We then construct the dataflow graph that captures the overall architecture of the RTL design. Subsequently, for every primary output in the target output list O, we update the scores of modules that exert a direct influence on that specific output (lines 6-9). Finally, modules with the highest scores are included in the list F, which comprises modules that are functionally significant for redaction (line 10).

In the subsequent phase, structural criteria are employed on each of the modules identified as functionally relevant for redaction (lines 12-15). Each module's compatibility with the specified eFPGA parameters is assessed (line 13). For Manuscript submitted to ACM

Algorithm 1: ARIANNA module filtering	g									
Input: Input RTL design D, eFPGA parameters P, list of selected outputs O										
<b>Output:</b> Set of candidate redaction modules <i>R</i>										
$1 M \leftarrow \text{ExtractInitialModules}(D)$	<pre>// Analyze input RTL design.</pre>									
$_2 S \leftarrow \emptyset$										
<sup>3</sup> foreach $m \in M$ do										
$4 \qquad S[m] \leftarrow 0$										
5 end										
6 foreach $o \in O$ do										
7 $T \leftarrow \text{IdentifyModules}(M, o)$	// Compute modules $T$ affecting $o$									
8 UPDATESCORE $(T, S)$	// Increment scores of modules $T$									
9 end										
10 $F \leftarrow \text{RankAndSelect}(M, S)$	<pre>// Select most relevant modules</pre>									
11 $R \leftarrow \emptyset$										
12 foreach $f \in F$ do										
13 <b>if</b> CHECKPARAMETERS(f, P) <b>then</b>										
14 $  R \leftarrow R \cup \{f\}$										
15 end										
16 return R										

instance, we calculate the module's number of I/O pins to evaluate its fit within the prospective eFPGA fabric. Modules that meet these structural conditions are then added to the list R (line 14).

The list *R* encompasses modules that not only influence a significant number of selected outputs but also meet the criteria for feasible eFPGA implementation. They can either be clustered together or stand-alone within an eFPGA fabric based on their dimensions. This step in the process is designed for adaptability, allowing for the inclusion of additional module-level filtering criteria as needed.

## 5.3 Cluster Identification

This stage focuses on identifying feasible combinations, called "clusters," that can be redacted onto an eFPGA. A cluster can include a single module (*single-module redaction*) or independent modules (*multi-module redaction*). A cluster is deemed valid if its eFPGA implementation complies with the specified constraints established by the designer, ensuring the solution remains within the predefined parameters.

Two modules are allowed in the same cluster only if independent and not part of the same hierarchy. Figure 5 shows a module hierarchy tree highlighting a valid and an invalid case. Although redacting a parent module without its children would be possible, that would require more I/O and data exchanges between the eFPGA and the non-redacted modules, increasing overheads.

Algorithm 2 shows the pseudo-code for cluster identification performed by ARIANNA. Taking as input a set of candidate redaction modules R and a set of eFPGA parameters P, it performs a *fixed-point analysis* to identify the set C of all candidate module clusters. Each of them is meant to fit into a single eFPGA; therefore, each cluster has to respect constraints dictated by the eFPGA parameters in P. We initialize the set C with the trivial clusters, the ones composed of a single module identified in the previous phase (lines 2-4). We then iteratively expand each cluster (lines 6-23). This part involves the recombination of each pair of admissible clusters to identify candidates for the current iteration (lines 8-17). Suppose the cluster was not already identified in the previous iterations and it respects the eFPGA constraints P and Manuscript submitted to ACM

Collini et al.



Fig. 5. Example of a valid and invalid cluster on a module hierarchy tree. Each node represents a module. An arrow from A to B means that module A instantiates module B. On the left, selecting B implicitly selects its sub-tree, including D and E. This is why, on the right, selecting both B and D explicitly yields an invalid cluster.

Algorithm 2: ARIANNA cluster identification
<b>Input:</b> Set of candidate redaction modules <i>R</i> , eFPGA parameters <i>P</i>
Output: Set of candidate module clusters C
$1 C \leftarrow \emptyset$
2 foreach $r \in R$ do
$3     C \leftarrow C \cup \{r\}$
4 end
$5 Flag \leftarrow False$
6 do
$7  D \leftarrow \emptyset$
8 foreach $c1 \in C$ do
9 foreach $c2 \in C$ do
10 if $c1 \neq c2$ then
11 $N \leftarrow c1 \cup c2$
12 <b>if</b> $N \not\subset D \land N \not\subset C \land CheckParameters(N, P)$ <b>then</b>
13 $D \leftarrow D \cup N$
14 end
15 end
16 end
17 <b>end</b>
18 $Flag \leftarrow False$
if $D \neq \emptyset$ then
$20 \qquad C \leftarrow C \cup D$
21 $Flag \leftarrow True$
22 end
23 while Flag
24 return C

hierarchy constraints (line 12). In that case, it is considered a new valid cluster and added to the list of current clusters (line 13). Each cluster's evaluation follows the same structural criteria applied to individual modules. For instance, in the context of multi-module redaction, the total number of I/O pins is calculated by aggregating the I/O pins of the individual modules within the cluster. The cluster earns the status of being admissible if it adheres to the designer's Manuscript submitted to ACM

predetermined constraints. At the end of each iteration, the new clusters (line 19) are added to the set C, and the procedure restarts. We terminate our algorithm when it is impossible to create new clusters by recombining the current ones. At the end of the procedure, each element of C is a candidate module cluster.

If a cluster contains more than one module, a module wrapper must be created, as OpenFPGA expects a single module. The module wrapper instantiates all the modules in the cluster and exposes their I/Os through its ports.

### 5.4 Bespoke Fabric Flow

This step aims to tailor a bespoke eFPGA fabric for each candidate module cluster identified in the previous phase. As shown in Section 2.1, OpenFPGA allows customization of CLBs, changing the number of LUTs (N) and the number of inputs for each LUT (K). We propose two design space exploration heuristics for finding N and K that will yield low area overhead. For both heuristics, we assume designers provide a range for N and K that they consider secure for their application, determined from the secure fabric identification step (Section 5.1).

Specifically, we propose two heuristics whose object is to find the smallest N and K that will yield the minimum tile number. Both heuristics start by identifying a lower bound for the tile number with the given cluster candidate. This is done by running OpenFPGA using the largest N and K provided by the user. This configuration provides the biggest CLBs. Therefore, the number of CLBs will be minimal. If the number of tiles exceeds the allowed one, we discard the current cluster candidate. Otherwise, we start our search.

The first heuristic minimizes N such that the tile number does not change and then minimizes K. We refer to this heuristic as NK. The second heuristic minimizes K such that the tile number does not change and then minimizes N, we refer to this heuristic as KN. A pseudo-code for the two heuristics is provided in Algorithm 3 and Algorithm 4, respectively. Both NK and KN heuristics begin by identifying the lower bound for the eFPGA size. To do this, we run OpenFPGA using the maximum N and K parameters provided by the user (line 1). If the lower bound size exceeds the maximum allowed, we end the procedure, marking the cluster as unfeasible due to resource constraints (line 3. If the identified lower bound size is admissible, we proceed by iteratively decreasing N (for NK heuristic) or K (for KN heuristic) until the size obtained by the reduced parameter increases (lines 5-10). If this happens, we select the N or K value from the previous iteration; otherwise, we stop at the minimum value for the parameter. Once we end this first iteration, we proceed by doing the same to minimize the second parameter (i.e., K for NK and N for KN – lines 12-17). This procedure explores only the minimum size configurations, which are the fastest to compute by OpenFPGA. Thus, it allows us to reduce the number of OpenFPGA runs required and eliminate the most demanding ones.

The framework can be run with either heuristic to obtain a bespoke fabric for each candidate module cluster. Given a module and a fabric configuration, OpenFPGA returns the corresponding eFPGA if the design is feasible and an error otherwise (e.g., when the cluster modules cannot be implemented for any reason). As a consequence, this phase also filters out unfeasible module clusters.

## 5.5 eFPGA Selection

At this stage, each candidate module cluster in *C* has been associated with its bespoke eFPGA fabric. The resulting candidate implementations must be characterized, ranked, and selected to determine the final solution. In this phase, we evaluate the utilization, and all candidate clusters select the best and final ones-more than one if the user decides to use more than one FPGA).

In Algorithm 5, we provide the pseudocode for this phase, detailing the step-by-step procedure. Initially, we parse the logs generated by OpenFPGA for the customized fabrics of each candidate module cluster, extracting essential data Manuscript submitted to ACM

```
Algorithm 3: NK Heuristic
  Input: Candidate module cluster C D, min N n, max N N, min K k, max K K, max
          size S
  Output: Optimal N No, optimal K Ko
 1 \ s \leftarrow \text{OpenFPGA}(C, N, K)
                                     // Get minimum size for current cluster
<sup>2</sup> if s > S then
      return None
 3
4 end
5 for i ∈ [1, N - n] do
      Si \leftarrow OPENFPGA(C, N - i, K)
      if Si > s then
          break
                                           // Decrease K until size increases
 8
      end
      No \leftarrow N-i
10
11 end
12 for i \in [1, K - k] do
      Si \leftarrow OPENFPGA(C, No, K - i)
13
      if Si > s then
14
         break
                                           // Decrease K until size increases
15
      end
16
      Ko \leftarrow K - i
17
18 end
19 return No, Ko
```

on CLB and I/O utilization (line 8). Subsequently, we compute a score for each fabric implementation, considering information regarding both I/O and CLB utilization as follows:

$$T_f = \frac{\text{MaxIOUtil} - \text{IOUtil}_f}{\text{MaxIOUtil}} + \frac{\text{MaxCLBUtil} - \text{CLBUtil}_f}{\text{MaxCLBUtil}}$$
(1)

where:

- IOUtil<sub>f</sub> and CLBUtil<sub>f</sub> represent the I/O and CLB utilization, respectively.
- MaxIOUtil and MaxCLBUtil represent the corresponding maximum I/O and CLB values for all analyzed eFPGAs, respectively.

This scoring approach balances the use of I/O and CLB resources and incorporates considerations related to security resilience. Fabrics with lower I/O utilization are generally more susceptible to certain attacks, as they can potentially reveal stuck-at-0 outputs more easily. Similarly, fabrics with lower CLB utilization provide less logic to be successfully recovered, further contributing to security resilience. These aspects are integral to our comprehensive evaluation framework, as described in our previous works [9, 22].

In our approach, we employ a *branch&bound algorithm* to systematically enumerate all possible combinations of eFPGAs that can be redacted together (lines 11-23), resulting in a comprehensive set of solutions. The algorithm begins with an empty working solution (line 9) and, at each iteration, strives to incorporate a new eFPGA implementation into each existing working solution (lines 12-22). Here, a solution denotes a collection of eFPGAs with non-overlapping module instances. If a solution reaches a terminal state (i.e., it either reaches the maximum allowable eFPGAs or redacts all eligible modules), it is appended to the final set of solutions (line 16). Otherwise, it remains in the working list for Manuscript submitted to ACM

Algorithm 4: KN Heuristic
<b>Input:</b> Candidate module cluster C D, min N n, max N N, min K k, max K K, max
size S
Output: Optimal N No, optimal K Ko
$s \leftarrow OPENFPGA(C, N, K)$ // Get minimum size for current cluster
2 if $s > S$ then
3 return None
4 end
5 <b>for</b> $i \in [1, K - k]$ <b>do</b>
$6  Si \leftarrow \text{OpenFPGA}(C, N, K - i)$
7 <b>if</b> $Si > s$ then
8 break // Decrease K until size increases
9 end
10 $Ko \leftarrow K - i$
11 end
12 <b>for</b> $i \in [1, N - n]$ <b>do</b>
13 $Si \leftarrow OpenFPGA(C, N - i, Ko)$
14 <b>if</b> $Si > s$ <b>then</b>
15 break // Decrease K until size increases
16 end
17 $No \leftarrow N-i$
18 end
19 return No, Ko

potential expansion (line 19). Upon the conclusion of this phase, the set *S* encompasses the complete assortment of viable solutions. Subsequently, we proceed to calculate a score for each solution. The score of a solution is derived as the summation of the scores of its constituent eFPGA implementations, with each score determined using Equation 1. The set *S* is then ranked based on these scores, with the highest-scoring solution being designated as the best and ultimate solution (line 25). This rigorous evaluation and ranking process enable us to identify the most optimal redaction solution from the pool of candidates.

In the final solution, we assemble a set of eFPGA implementations, each comprising a roster of module instances to be redacted. At this juncture, our next task is to regenerate the top module for ASIC implementation (referred to as the "Top ASIC module" in Figure 4). This involves substituting the redacted instances with their corresponding eFPGA instances. In scenarios involving multi-module redaction, where multiple modules may be distributed throughout the design, we conduct a "dominator tree" analysis on the module hierarchy. This analysis helps identify the optimal insertion points for eFPGA instances, intending to minimize wire lengths.

During this process, we re-route signals originating from the original instances to their respective eFPGA instances. Additionally, control signals are propagated to the top module as required. We also remap the module signals to correspond with the eFPGA GPIO (General Purpose Input/Output) signals to ensure correct connectivity.

Once these modifications are complete, the updated design and fabric netlists are ready for handoff to physical design tools, facilitating the translation of the design into a finalized, manufacturable ASIC implementation. This comprehensive approach ensures that the redaction process seamlessly integrates eFPGA solutions into the ASIC design, optimizing functionality and security.

```
Algorithm 5: ARIANNA eFPGA selection
   Input: Set of candidate module clusters C, eFPGA parameters P
   Output: Solution st
 1 F \leftarrow \emptyset
<sup>2</sup> foreach c \in C do
        f \leftarrow \text{CREATEEFPGA}(c, P)
 3
        if IsVALID(f) then
 4
           F \leftarrow F \cup f
 5
       end
 6
7 end
8 T \leftarrow \text{ComputeScore}(F)
9 W \leftarrow \{\}
                                                     // Initialize with empty solution
10 S \leftarrow \emptyset
11 foreach w \in W do
        foreach f \in F do
12
            c \leftarrow f \cup w
13
            if isValidSolution(c) then
14
                 if IsFINAL(c) then
15
                    S \leftarrow S \cup c
16
                 end
17
                 else
18
                  W \leftarrow W \cup c
19
                 end
20
            end
21
       end
22
23 end
24 S \leftarrow S \cup W \setminus \{\}
25 s_t \leftarrow \text{RankAndSelect}(S, T)
26 return st
```

## 6 Experimental Evaluation

We implemented a prototype of ARIANNA in Python, using the PyVerilog framework [45]. PyVerilog can parse the Verilog designs, analyze and manipulate the resulting Abstract Syntax Tree (AST), and regenerate the output files, including those fed into the OpenFPGA toolchain for eFPGA creation.

To identify the set of secure fabrics, we run a security evaluation considering IcySAT [46] as the reference attack. We selected IcySAT as it is the most powerful attack with an open-source implementation that we were able to find.

Then, we conduct an exhaustive analysis of the design space of fabric parameters for each benchmark. This preliminary analysis helped us formulate the proposed heuristics and allowed us to show the complexity of the problem. We show the results obtained with the proposed heuristics for the bespoke fabrics, comparing them with the exhaustive approach. Eventually, we show results from the complete flow using Cadence Genus 18.14 for logic synthesis and Cadence Innovus 18.10 for physical design, targeting the NanGate 45nm Open Cell Library.

Table 1 shows the benchmarks we used to validate ARIANNA. The table reports the number of modules and instances that can be redacted. We report the range of the I/O pin count for such modules. We identified the main data output(s) of each design as the outputs of interest for the module filtering phase. These benchmarks are commonly used to evaluate

ARIANNA: An Automatic Design Flow for Fabric Customization and eFPGA Redaction

Suite	Design	Modules (#)	Instances (#)	I/O pins [min, max]
	DES3	11	11	[12, 301]
CED	FIR	5	5	[64, 384]
CEP	SHA256	3	3	[38, 774]
	SASC	2	3	[23, 28]
IWLS05	USB_PHY	3	3	[17, 33]
OpenROAD	GCD	10	11	[6, 68]
Opencores	Nautilus	8	8	[19, 93]

Table 1. Characteristics of the selected benchmarks

RTL locking [47]. In the context of eFPGA redaction, we can consider these designs as IPs part of a bigger SoC. The designer wants to protect this IP, but redacting the whole IP is not feasible. Using ARIANNA, the designer can find redaction solutions that satisfy the constraints.

#### 6.1 Secure Fabric Identification

For the secure fabric identification pre-step, we considered IcySAT [46] as the reference attack. IcySAT is the most powerful attack with an open-source implementation that we could find. We ran IcySAT on each fabric configuration with a timeout of 48 hours. Designers might choose a different attack or multiple attacks and limit or increase the eFPGA size they are interested in and the attack(s) timeout.

To run the attack, we must convert the gate-level netlist into a format that can be understood by an attack tool, treating the configuration bitstream as a collection of "key inputs." In an eFPGA, the bitstream is stored in configuration flip-flops. These configuration flip-flops are linked together in a scan chain that is controlled by a programming clock (prog\_clk). To locate the configuration scan chain, we perform a depth-first search of the netlist, beginning from the scan\_in\_head port and continuing until we arrive at the scan\_in\_tail. Every flip-flop (FF) in the traversal path influenced by the programming clock (prog\_clk) retains the configuration bitstream. The sequence in which the configuration FFs are identified aligns with the order of the bitstream. The recognized configuration FFs serve as primary key inputs to transform the eFPGA netlist into a version compatible with IcySAT. To create an Oracle, we utilize the same locked netlist but assign the key bits the configuration values derived from the bitstream produced in the OpenFPGA process.

Figure 6 shows the results for each fabric configuration for a 4x4, 5x5, and 6x6 eFGPA size. The results show that for 4x4 eFPGA size, all attacks run to completion before timeout. For a 5x5 eFGPA size, only the smaller tiles get to completion before the timeout. For 6x6 eFPGA size, all fabrics stop at timeout. From this, all fabrics should be considered as all would be safe from IcySAT if the final eFPGA size is 6x6 or bigger.

## 6.2 Heuristics Evaluation

In our preliminary analysis, we ran an exhaustive search for each candidate module cluster and saved all the results with the different fabric configurations. Overall candidate module clusters, overall benchmarks (291 in total), and the number of times an FPGA with a nonminimum number of tiles has the minimum area equal to 22. Figure 7 shows how many times each fabric configuration (identified by the N-K pair) yields the best result, their average relative cost (with respect to the best result), and the standard deviation of the relative cost for GCD, Nautilus, and DES3. We show these three benchmarks as they are more meaningful because they have a larger number of candidate module clusters Manuscript submitted to ACM



Fig. 6. SAT attack results [s]. Timeout is 48 hours.

Table 2. Heuristic vs Exhaustive approach. The cost is relative to the best solution found with the exhaustive search (the best solution has cost 1).

Ponohmori	# best		Avg.	Rel. Cost	Std. Re	el. Cost	Time [s]					
Denchinark	NK	KN	NK	KN	NK	KN	NK	KN	Exh.			
DES3	247	222	1.0	1.14	0.0	0.45	5962	5453	31170			
FIR	0	1	1.56	1.03	0.46	0.03	58	53	968			
SHA	1	1	1.0	1.0	0.0	0.0	13	10	162			
SASC	0	1	1.53	1.0	0.0	0.0	15	14	86			
USB	0	0	1.04	1.33	0.0	0.0	23	21	373			
GCD	2	11	4.78	4.36	13.61	13.70	289	287	1986			
Nautilus	6	7	1.21	1.07	0.27	0.14	277	274	5653			

compared to the rest of the benchmarks. From the heatmaps, we can see two different cases. For GCD (top row), no predominant fabric configuration is always good, especially as highlighted by the average relative distance from the best. For DES3 (bottom row) we can see that the most predominant fabric configuration (K6 and N4), is not very good on average (250% average relative cost), this is reflected by the high standard deviation, indicating that this fabric is either very good or very bad. Conversely, the K6 N2 configuration has a very low average relative cost (104%) and very low standard deviation, indicating that this configuration is always pretty good, although not very frequently the best. Though, we must notice that is always pretty good for the candidate module clusters in this design, for GCD that configuration has an average relative cost of 252%. This motivates the tailoring of a bespoke fabric for each candidate module cluster. As the number of best solutions that do not have minimum tile numbers is marginal, we search for the solutions with minimum tile numbers. See Section 5.4 for a detailed explanation of the proposed heuristics.

We ran the redaction flow to evaluate the proposed heuristics and focused on the bespoke fabric results for each candidate cluster. We compared the results of the two heuristics against the exhaustive analysis results shown in Section 6.2. Table 2 summarizes the results, reporting the average and standard deviation of the relative cost and the computation time. The two heuristics perform very well in all benchmarks, with GCD being an outlier. The two heuristics present very close computation times, often over an order of magnitude lower than the exhaustive approach. In fact, as the heuristics start exploring the bigger tiles and stop at the first iteration in which the grid size increases, they do not only cut the number of explored solutions, but they also explore the least expensive ones in terms of computations. This is because the computation time of OpenFPGA increases as the number of tiles increases. Manuscript submitted to ACM

				# best						ļ	Avg. relati	ve cost (1	L is optim	al)					Std.	of relative	e cost		
7.0 -	0	0	0	0	0	0	0	7.0 -	3.90	3.46	3.76	6.35	6.35	7.19	8.48	7.0 -	3.05	1.16	1.09	11.09	11.09	10.97	14.40
6.0 -	0	0	0	0	0	0	0	6.0 -	3.15	2.66	2.89	3.53	5.24		5.40	6.0 -	3.20	1.13	1.05	1.20	11.28	11.26	11.24
¥ 5.0 -	0	0	0	0	2	1	0	⊻ 5.0 -	2.67	2.21	2.17	2.36	2.09	4.42	4.36	⊻ 5.0 -	3.31	1.14	1.13	1.11	0.65	11.41	11.42
4.0 -	0	3	2	1	0	1	0	4.0 -	5.84	2.50	1.82	2.04	1.96	2.18	2.32	4.0 -	7.61	3.35	1.18	1.16	1.14	1.12	1.10
3.0 -	2	2	6	2	1	2	4	3.0 -	9.19	4.80	2.17	1.79	1.80	1.93	1.92	3.0 -	10.28	6.44	2.91	1.18	1.17	1.16	1.18
	2.0	3.0	4.0	5.0 N	6.0	7.0	8.0		2.0	3.0	4.0	5.0 N	6.0	7.0	8.0		2.0	3.0	4.0	5.0 N	6.0	7.0	8.0
ċ		i	2	3	4	5	,	6	ź	3	4	ś	6 :	7 8	ģ		2	4	6	8	10	12	14
				# best						,	Avg. relati	ve cost (1	L is optima	al)					Std.	of relative	e cost		
7.0 -	0	0	0	0	0	0	0	7.0 -	5.15	2.62	2.72	2.53	2.68	2.87	6.09	7.0 -	2.82	1.26	1.23	0.79	0.80	0.98	12.48
6.0 -	1	1	0	0	0	1	3	6.0 -	5.04	3.52	2.58	2.78	2.31	2.19	5.58	6.0 -	2.95	2.28	1.30	1.17	0.82	0.88	12.62
¥ 5.0 -	0	0	0	2	2	0	0	⊻ 5.0 -	5.08	5.17	3.05	2.50	2.43	2.13	2.16	⊻ 5.0 -	2.91	2.79	1.93	1.39	1.42	0.90	0.95
4.0 -	0	0	0	0	0	0	1	4.0 -	7.88	5.06	5.15	5.27	3.62	3.72	2.12	4.0 -	7.81	2.93	2.83	2.69	2.19	2.11	0.88
3.0 -	0	0	0	0	0	0	0	3.0 -	11.78	5.16	5.31	5.16	5.19	5.28	5.35	3.0 -	13.10	2.81	2.65	2.81	2.78	2.68	2.61
	2.0	3.0	4.0	5.0 N	6.0	7.0	8.0		2.0	3.0	4.0	5.0 N	6.0	7.0	8.0		2.0	3.0	4.0	5.0 N	6.0	7.0	8.0
0.	D	0.5	1.0	1.5	2.0	2.5	3.0			4	6		8	10			2	4	,	5	8	10 10	12
				4 h +																			
7.0 -	0	0	0	# best	0	0	0	7.0 -	1.49	2.10	2.89	e cost (1 3.28	3.63	) 4.19	4.66	7.0 -	0.05	0.11	5td.	of relative 7.61	e cost 7.56	7.48	7.41
6.0 -	36	0	219	0	0	0	0	6.0 -	1.04	1.38	2.52	2.83	2.98	3.24	3.64	6.0 -	0.01	0.03	7.73	7.68	7.66	7.62	7.56
¥ 5.0 -	0	2	0	0	0	0	0	⊻ 5.0 -	15.75	10.92	6.73	3.55	3.40	2.77	2.76	× 5.0 -	8.37	7.34	5.13	2.96	2.31	1.15	0.72
4.0 -	0	0	1	0	0	0	0	4.0 -	19.08	15.55	14.02	10.54	7.31	5.71	4.18	40-	10.74	8 25	6 79	7.06	5.73	3.43	2.86
3.0 -	0	0	0	0	0	0	0	3.0 -	20.43	20.26	19.44	19.12	18.67	16.43	15.10	2.0	11.25	11.20	10.86	10.68	10.76	8.50	6.68
	2.0	3.0	4.0	5.0	6.0	7.0	8.0		2.0	3.0	4.0	5.0	6.0	7.0	8.0	5.0 -	2.0	3.0	4.0	5.0	6.0	7.0	8.0
				N								N								N			
ċ	2	5 50	75	100 1	125 150	175	200		2.5	5.0	7.5	10.0	12.5 1	5.0 17	.5 20.0	. '		2	4	6	8	3	10

Fig. 7. Number of best results, average and standard deviation of distance from the best result for GCD (top row), Nautilus (middle row), and DES3 (bottom row) for each fabric configuration (given by the N-K pair). Darker is better. There is not a predominant configuration. The most frequent best solution for DES3 has a high average relative cost, highlighted by its std. Motivating the need for heuristics to search the space.

#### 6.3 Framework Evaluation

To evaluate the complete flow, we set ARIANNA to run in two configurations for each heuristic and compared our results with the previous work [12] where N and K were set to 4. We run two configurations to see the effects of redacting with a single but bigger eFPGA versus redacting with two smaller eFPGAs. In cfg1, we set the maximum I/O pin count of the modules that can redacted to 64, and the limit is two eFPGAs. In cfg2, the maximum I/O pin count is 96, and the limit is one eFPGA. The results of the Clustering Flow (Section 5.3) are going to give the same inputs to the two heuristics, which will potentially find different solutions for the bespoke fabrics (Section 5.4) of the candidate clusters, leading to different scores in the final selection phase (Section 5.5). We validated the designs with Cadence Genus 18.14 for logic synthesis, targeting the NanGate 45nm Open Cell Library.

In Table 3, we present the results at each flow step. In Table 4, we present the final redacted solutions with their respective CLB and I/O utilization. In Figure 8, we present the overhead results.

Config.	Design	# Instances	Module Fi	ltering	Cluster I	dent.		]	Fabric Ex	ploration	& eFPC	GA Sele	ction		
			Time [s]	$\ R\ $	Time [s]	$\ C\ $	Time [s]			# Oper	nFPGA	runs	S		
							ALICE	NK	KN	ALICE	NK	KN	ALICE	NK	KN
	DES3	11	338.7	8	1.2	218	748.4	4383.1	4210.3	218	1716	1632	3151	2219	2219
, si	FIR	5	0.4	1	0.0	1	2.1	19.8	16.1	1	8	7	1	1	1
: GA	SHA256	3	15.3	1	0.0	1	5.3	11.3	11.0	1	4	4	1	1	1
E F D O E F	SASC	3	0.3	1	0.0	1	2.8	15.1	13.9	1	6	6	1	1	1
4 I/ 2 €	USB_PHY	3	1.2	3	0.0	3	9.7	21.3	21.1	3	7	7	1	1	1
° %	GCD	11	0.6	8	0.0	17	32.5	193.9	183.2	17	96	98	78	69	69
	NAUTILUS	8	104.1	3	0.0	6	36.0	95.4	99.7	6	24		6	6	6
	DES3	11	336.7	8	1.3	255	861.3	5356.0	5454.1	255	2012	1928	232	247	247
	FIR	5	0.2	3	0.0	3	29.3	57.1	59.5	3	16	15	3	2	2
G^g	SHA256	3	15.1	1	0.0	1	5.3	10.5	10.4	1	4	4	1	1	1
FF O	SASC	3	0.3	1	0.0	1	2.7	13.9	14.0	1	6	6	1	1	1
16 C	USB_PHY	3	1.2	3	0.0	3	9.5	20.8	23.7	3	7	7	1	1	1
6 X	GCD	11	0.6	9	0.1	43	94.3	527.2	287.2	43	238	141	33	32	19
	NAUTILUS	8	134.6	5	0.0	9	73.4	181.8	99.7	9	35	23	6	5	5

Table 4. Redaction results after running ARIANNA with two different configurations with the previous work [12] (prev.) and the proposed heuristics. Redacted module lists are separated by ";" to indicate different eFPGAs. In all cases, the I/O utilization is improved using the novel heuristics for bespoke fabrics.

Config.	Design		Redacted Modules		eI	PGA size		eF parar	PGA ns (N-	K)	CLB Util. [%]			I/O I	Util. [%	[]
		ALICE	NK	KN	ALICE	NK	KN	ALICE	NK	KN	ALICE	NK	KN	ALICE	NK	KN
	DECa	sbox4, sbox3, sbox2;	sbox8, sbox3, sbox2, sbox1;	sbox8, sbox3, sbox2, sbox1;	6x6	4x4	4x4	4-4	4-6	4-6	100	100	100	23	100	100
	DES3	sbox8, sbox7, sbox6, sbox5	sbox7, sbox6, sbox5, sbox4	sbox7, sbox6, sbox5, sbox4	7x7	4x4	4x4	4-4	4-6	4-6	100	100	100	25	62	62
1: bits PGAs	FIR	right mul. block	right mul. block	right mul. block	6x6	5x5	5x5	4-4	6-4	7-3	69	78	89	50	67	67
cfg 64 I/C & 2 eF	SHA256	k constants	k constants	11x11	4x4	4x4	4-4	8-6	8-6	85	100	100	13	59	59	
	SASC	sasc fifo	sasc fifo	sasc fifo	7x7	5x5	5x5	4-4	6-6	8-4	76	89	100	14	24	24
	USB_PHY	usb tx phy	usb tx phy	usb tx phy	7x7	5x5	5x5	4-4	7-6	8-5	80	100	89	11	18	18
	CCD	UnitCtr, Mux;	ZeroComp, RegEn;	ZeroComp, RegEn;	5x5	4x4	4x4	4-4	6-6	8-3	100	100	100	65	97	97
	GCD	RegEn	UnitCtr, Mux	UnitCtr, Mux	4x4	4x4	4x4	4-4	8-5	8-5	100	100	100	53	80	80
	NAUTILUS	ALU, LIFO;	RegFile, ShiftByte;	RegFile, ShiftByte;	13x13	6x6	6x6	4-4	8-6	8-6	90	94	94	16	42	42
		RegFile	ALU	ALU	9x9	4x4	4x4	4-4	8-7	8-7	86	100	100	16	45	45
ts iA	DES3	sbox8, sbox7, sbox6, sbox5, sbox4, sbox1	sbox8, sbox7, sbox6, sbox5, sbox4, sbox3, sbox2, sbox1	sbox8, sbox7, sbox6, sbox5, sbox4, sbox3, sbox2, sbox1	8x8	5x5	5x5	4-4	4-6	4-6	100	89	89	31	83	83
2: 2 bi	FIR	block right	block right	block right	6x6	5x5	5x5	4-4	5-6	7-3	88	100	100	52	69	69
fel ∑efe	SHA256	k constants	k constants	k constants	11x11	4x4	4x4	4-4	8-6	8-6	85	100	100	13	59	59
8 %	SASC	sasc fifo	sasc fifo	sasc fifo	7x7	5x5	5x5	4-4	6-6	8-4	76	89	100	14	24	24
	USB_PHY	usb tx phy	usb tx phy	usb tx phy	7x7	5x5	5x5	4-4	7-6	8-5	80	100	89	11	18	18
	GCD	ZeroComp, Mux	UnitCtr, Mux	UnitCtr, Mux	5x5	4x4	4x4	4-4	8-5	8-5	100	100	100	69	97	97
	NAUTILUS	Control Unit	Control Unit	Control Unit	8x8	6x6	6x6	4-4	5-7	8-4	83	100	100	48	72	72

Diving into Table 3, we can see how the module instances are filtered at the first phase, which identifies the candidate modules set R. This first phase also includes the dataflow analysis needed for the module filtering. The big fluctuations of the time needed in this phase across the different benchmarks reflect the dataflow complexity of the designs. After module filtering, combinations of these modules are identified in the cluster identification phase, which yields the candidate module cluster C. Here, we can see that the size of C can drastically increase from the size of R when the candidate modules are all independent (like the Sbox modules in DES3). The module filtering and cluster identification phases are performed in the same way for all flows. For this reason, we did not report separate data as we did for the Manuscript submitted to ACM



Fig. 8. Area, Power, and Timing results. The proposed heuristics outperform the state of the art, reducing overheads.



(a) cfg1: two 4×4 N=4; K=6 eFPGA (103,007 $\mu m^2$ )



(b) cfg2: one 5×5 N=4; K=6 eFPGA (113,723µm<sup>2</sup>)

Fig. 9. Physical layouts of DES3 solutions using the NK heuristic. In both configurations, the redacted modules are the same, on the left distributed between two eFPGAs and on the right all into the same eFPGA. The proportions are to scale.

fabric exploration phase. From the fabric exploration and eFPGA selection phase, we can see how the computational load is higher with the heuristics as they need to perform more OpenFPGA runs for the parameter exploration. The runtime is still reasonable for an EDA flow. From Table 4 and Figure 8, we can see that this computational overhead is well spent. We can identify different scenarios. For DES3, we can see how, in both configurations, the heuristics allow us to redact more modules while getting lower overheads (almost  $2\times$  lower). The redacted modules for FIR, SHA256, and SASC do not change, though tailoring the fabrics gets up to  $3.3\times$  lower overheads. For GCD and Nautilus, the redacted modules change for bigger ones, which increases the overheads by  $1.2\times$ . USB\_PHY shows increased area overheads with improvements only in the timing for the heuristics while redacting the same module. This configuration finds a case where more but smaller tiles end up in a lower area than fewer bigger tiles. The latter configuration (identified by KN heuristics) reduces the timing overhead by  $\sim 50\%$  while increasing area by  $\sim 15\%$ . In DES3, using the heuristics, the final redacted modules are the same in cfg1 and cfg2, with cfg1 splitting the redacted modules in  $2.4\times4$  eFPGAs and cfg2 using a single  $5\times5$  eFPGA. From the synthesis results, we can see that the former presents lower overheads, but from our security analysis in Section  $5.1.4\times4$  eFGPAs are orders of magnitude weaker to SAT attacks.

In all cases, the I/O utilization is improved. DES3 is the only benchmark for which the CLB utilization does not improve, although the I/O utilization for DES3 has the biggest improvements across our benchmarks. Higher fabric utilization means we are wasting fewer resources and that the solutions are more resilient from attacks [8, 9, 22].

Figure 9 shows the two physical designs for DES3 using the NK heuristic (which, by chance, coincides with the KN ones). This benchmark is small, so most of the chip is occupied by the eFPGA(s). However, the overhead will become less relevant when the component is inserted into a larger system-on-chip (like PicoSoc in reference [9]). We can notice how, despite redacting the same modules, using two smaller eFPGAs yields a slightly smaller area ( $\approx 10\%$ ). A designer might prefer this solution as it would require an attacker to retrieve 2 bitstreams instead of one.

Looking at the two proposed heuristics, both yield bespoke fabrics that improve redaction solutions. The KN heuristics often yield better synthesis results with lower area and power overheads. A designer might want to use the KN heuristic first and then use NK only if the results are unsatisfactory. Using both heuristics is still much faster than the exhaustive approach (see Table 2).

Comparing our results with SheLL [37] is not straightforward as the set of benchmarks differs considerably, and the only common benchmark is FIR. They claim up to  $2\times$  reduction in overheads compared to the ALICE [12] approach, whereas our results show up to a 3.3× reduction (for SHA).

In cases where the final eFPGA size and fabric configuration are not considered secure from the secure fabric identification step, the designer has two options: discard the solution or integrate additional security measures to mitigate the attacks. For example, in our case, 4×4 fabrics are not secure against the IcySAT attack that we considered for our secure fabric identification. The designer could still choose 4×4 fabrics if they also integrate a secure scan chain protection technique like DisORC [44].

## 7 Conclusions and Future Work

This paper proposes ARIANNA, an expanded version of the ALICE framework, focusing on the eFPGA fabric parameter selection problem. We proposed two heuristics for the design space exploration of fabric parameters for eFPGA redaction. We first analyzed the heuristics isolated in the parameter selection phase against an exhaustive approach and then in the complete framework against the previous work.

Our results showed significant improvements (up to  $3.3 \times$  lower overheads and  $4 \times$  higher fabric utilization) can be obtained by tailoring bespoke fabrics for eFPGA redaction. Compared to state-of-the-art, we find that SheLL [37] reduced overheads by 55% compared to ALICE, whereas our work reached improvements of up to 330%.

Our heuristics allow for a more scalable framework with respect to an exhaustive approach. Moreover, the proposed heuristics yield higher utilization of the eFPGA fabric, meaning less wasted resources and better security [8, 9, 22]. With this contribution, ARIANNA is now a complete framework that can tackle both the module selection and the fabric configuration problems faced when applying eFPGA redaction. Future work includes expanding the framework to explore more eFPGA configuration parameters for the fabrics and the overall architecture. This step can include ML-driven exploration techniques. Also, selection methods can be extended to include more criteria or perform a fine-grain decomposition and redaction of larger modules.

## References

- W. Hu, C.-H. Chang, A. Sengupta, S. Bhunia, R. Kastner, and H. Li, "An overview of hardware security and trust: Threats, countermeasures, and design tools," *IEEE Transactions on CAD of Integrated Circuits and Systems*, vol. 40, no. 6, 2021.
- [2] K. Shamsi, M. Li, K. Plaks, S. Fazzari, D. Z. Pan, and Y. Jin, "IP protection and supply chain security through logic obfuscation: A systematic overview," ACM Trans. on Design Automation of Electronic Systems (TODAES), vol. 24, no. 6, pp. 1–36, 2019.
- [3] A. Abdel-Hamid, S. Tahar, and E. Aboulhamid, "A survey on ip watermarking techniques," Design Automation for Embedded Systems, vol. 9, p. 211–227, 2004.
- [4] T. D. Perez and S. Pagliarini, "A survey on split manufacturing: Attacks, defenses, and challenges," IEEE Access, vol. 8, pp. 184013–184035, 2020.

#### ARIANNA: An Automatic Design Flow for Fabric Customization and eFPGA Redaction

- [5] B. Tan, R. Karri, N. Limaye, A. Sengupta, O. Sinanoglu, M. M. Rahman, S. Bhunia, D. Duvalsaint, R. D. Blanton et al., "Benchmarking at the frontier of hardware security: Lessons from logic locking," 2020.
- [6] X. Tang, E. Giacomin, B. Chauviere, A. Alacchi, and P.-E. Gaillardon, "Openfpga: An open-source framework for agile prototyping customizable fpgas," *IEEE Micro*, vol. 40, no. 4, pp. 41–48, 2020.
- [7] D. Koch, N. Dao, B. Healy, J. Yu, and A. Attwood, "Fabulous: An embedded FPGA framework," in ACM/SIGDA FPGA, 2021, pp. 45–56.
- [8] P. Mohan, O. Atli, J. Sweeney, O. Kibar, L. Pileggi, and K. Mai, "Hardware Redaction via Designer-Directed Fine-Grained eFPGA Insertion," in DATE, 2021.
- [9] J. Bhandari, A. K. Thalakkattu Moosa, B. Tan, C. Pilato, G. Gore, X. Tang, S. Temple, P.-E. Gaillardon, and R. Karri, "Exploring efpga-based redaction for ip protection," in *IEEE/ACM ICCAD*, 2021.
- [10] P. D. Schiavone, D. Rossi, A. Di Mauro, F. K. Gürkaynak, T. Saxe, M. Wang, K. C. Yap, and L. Benini, "Arnold: An efpga-augmented risc-v soc for flexible and low-power iot end nodes," *IEEE Trans. on VLSI Systems*, vol. 29, no. 4, pp. 677–690, 2021.
- [11] J. Chen, M. Zaman, Y. Makris, R. D. S. Blanton, S. Mitra, and B. C. Schafer, "Decoy: Deflection-driven hls-based computation partitioning for obfuscating intellectual property," in ACM/IEEE Design Automation Conference (DAC), 2020, pp. 1–6.
- [12] C. M. Tomajoli, L. Collini, J. Bhandari, A. K. T. Moosa, B. Tan, X. Tang, P.-E. Gaillardon, R. Karri, and C. Pilato, "Alice: An automatic design flow for efpga redaction," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, ser. DAC '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 781–786. [Online]. Available: https://doi.org/10.1145/3489517.3530543
- [13] A. Boutros and V. Betz, "Fpga architecture: Principles and progression," IEEE Circuits and Systems Magazine, vol. 21, no. 2, pp. 4–29, 2021.
- [14] E. Ahmed and J. Rose, "The effect of lut and cluster size on deep-submicron fpga performance and density," IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 12, no. 3, pp. 288–298, 2004.
- [15] D. Koch, N. Dao, B. Healy, J. Yu, and A. Attwood, "FABulous: An Embedded FPGA Framework," in *The 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. Virtual Event USA: ACM, Feb. 2021, pp. 45–56. [Online]. Available: https://dl.acm.org/doi/10.1145/3431920.3439302
- [16] P. Mohan, O. Atli, O. Kibar, M. Zackriya, L. Pileggi, and K. Mai, "Top-down Physical Design of Soft Embedded FPGA Fabrics," in *The 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. Virtual Event USA: ACM, Feb. 2021, pp. 1–10. [Online]. Available: https://dl.acm.org/doi/10.1145/3431920.3439297
- [17] X. Tang, E. Giacomin, A. Alacchi, B. Chauviere, and P.-E. Gaillardon, "OpenFPGA: An Opensource Framework Enabling Rapid Prototyping of Customizable FPGAs," in 2019 29th International Conference on Field Programmable Logic and Applications (FPL), Sep. 2019, pp. 367–374, iSSN: 1946-1488.
- [18] X. Tang, E. Giacomin, B. Chauviere, A. Alacchi, and P.-E. Gaillardon, "OpenFPGA: An Open-Source Framework for Agile Prototyping Customizable FPGAs," *IEEE Micro*, vol. 40, no. 4, pp. 41–48, Jul. 2020, conference Name: IEEE Micro.
- [19] A. Li and D. Wentzlaff, "PRGA: An Open-Source FPGA Research and Prototyping Framework," in *The 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. Virtual Event USA: ACM, Feb. 2021, pp. 127–137. [Online]. Available: https://dl.acm.org/doi/10.1145/3431920.3439294
- [20] J. Luu, J. H. Anderson, and J. S. Rose, "Architecture Description and Packing for Logic Blocks with Hierarchy, Modes and Complex Interconnect," in Proceedings of the 19th ACM/SIGDA International Symposium on Field Programmable Gate Arrays, ser. FPGA '11. New York, NY, USA: Association for Computing Machinery, 2011, p. 227–236. [Online]. Available: https://doi.org/10.1145/1950413.1950457
- [21] X. Tang, E. Giacomin, G. D. Micheli, and P.-E. Gaillardon, "FPGA-SPICE: A Simulation-Based Architecture Evaluation Framework for FPGAs," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 3, pp. 637–650, Mar. 2019. [Online]. Available: https://ieeexplore.ieee.org/document/8576622/
- [22] J. Bhandari, A. K. T. Moosa, B. Tan, C. Pilato, G. Gore, X. Tang, S. Temple, P.-E. Gaillardon, and R. Karri, "Not all fabrics are created equal: Exploring efpga parameters for ip redaction," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 31, no. 10, pp. 1459–1471, 2023.
- [23] "Intel Xeon+FPGA Platform for the Data Center," https://reconfigurablecomputing4themasses.net/files/2.2%20PK.pdf. [Online]. Available: https://reconfigurablecomputing4themasses.net/files/2.2%20PK.pdf
- [24] P. D. Schiavone, D. Rossi, A. Di Mauro, F. K. Gurkaynak, T. Saxe, M. Wang, K. C. Yap, and L. Benini, "Arnold: An eFPGA-Augmented RISC-V SoC for Flexible and Low-Power IoT End Nodes," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 29, no. 4, pp. 677–690, Apr. 2021. [Online]. Available: https://ieeexplore.ieee.org/document/9369856/
- [25] J. J. V. Rajendran, "An overview of hardware intellectual property protection," in IEEE International Symposium on Circuits and Systems (ISCAS), 2017, pp. 1–4.
- [26] Z. U. Abideen and S. P. T. D. Perez, "From fpgas to obfuscated easies: Design and security trade-offs," in IEEE AsianHOST, 2021, pp. 1-4.
- [27] A. Chakraborty, N. G. Jayasankaran, Y. Liu, J. Rajendran, O. Sinanoglu, A. Srivastava, Y. Xie, M. Yasin, and M. Zuzak, "Keynote: A disquisition on logic locking," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 39, no. 10, 2020.
- [28] M. Yasin and O. Sinanoglu, "Evolution of logic locking," in IFIP/IEEE VLSI-SoC, 2017.
- [29] D. Sisejkovic, L. M. Reimann, E. Moussavi, F. Merchant, and R. Leupers, "Logic locking at the frontiers of machine learning: A survey on developments and opportunities," in 2021 IFIP/IEEE 29th International Conference on Very Large Scale Integration (VLSI-SoC), 2021, pp. 1–6.
- [30] J. Gandhi, D. Shekhawat, M. Santosh, and J. G. Pandey, "Logic locking for ip security: A comprehensive analysis on challenges, techniques, and trends," *Computers & Security*, vol. 129, p. 103196, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167404823001062
- [31] P. Subramanyan, S. Ray, and S. Malik, "Evaluating the security of logic encryption algorithms," in IEEE HOST, 2015, pp. 137–143.

- [32] L. Alrahis, S. Patnaik, M. Shafique, and O. Sinanoglu, "Omla: An oracle-less machine learning-based attack on logic locking," IEEE Transactions on Circuits and Systems II: Express Briefs, vol. 69, no. 3, pp. 1602–1606, 2022.
- [33] L. Alrahis, S. Patnaik, J. Knechtel, H. Saleh, B. Mohammad, M. Al-Qutayri, and O. Sinanoglu, "Unsail: Thwarting oracle-less machine learning attacks on logic locking," *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 2508–2523, 2021.
- [34] D. Sisejkovic, F. Merchant, L. M. Reimann, H. Srivastava, A. Hallawa, and R. Leupers, "Challenging the security of logic locking schemes in the era of deep learning: A neuroevolutionary approach," *J. Emerg. Technol. Comput. Syst.*, vol. 17, no. 3, May 2021. [Online]. Available: https://doi.org/10.1145/3431389
- [35] D. Sisejkovic, L. Collini, B. Tan, C. Pilato, R. Karri, and R. Leupers, "Designing ml-resilient locking at register-transfer level," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, ser. DAC '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 769–774. [Online]. Available: https://doi.org/10.1145/3489517.3530541
- [36] P. Mohan, O. Atli, O. Kibar, M. Zackriya, L. Pileggi, and K. Mai, "Top-down physical design of soft embedded fpga fabrics," in ACM/SIGDA FPGA, 2021, p. 1–10.
- [37] H. M. Kamali, K. Z. Azar, F. Farahmandi, and M. Tehranipoor, "Shell: Shrinking efpga fabrics for logic locking," in 2023 Design, Automation & Test in Europe Conference & Exhibition (DATE), 2023, pp. 1–6.
- [38] C. Sathe, Y. Makris, and B. C. Schafer, "Investigating the effect of different efpgas fabrics on logic locking through hw redaction," in 2022 IEEE 15th Dallas Circuit And System Conference (DCAS), 2022, pp. 1–6.
- [39] A. Rezaei, R. Afsharmazayejani, and J. Maynard, "Evaluating the security of efpga-based redaction algorithms," in *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*, ser. ICCAD '22. New York, NY, USA: Association for Computing Machinery, 2022. [Online]. Available: https://doi.org/10.1145/3508352.3549425
- [40] H. Zhou, R. Jiang, and S. Kong, "CycSAT: SAT-based attack on cyclic logic encryptions," in 2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), Nov. 2017, pp. 49–56, iSSN: 1558-2434.
- [41] P. Karmakar, M. Bharani, and C. Karfa, "Evaluating the robustness of large scale efpga-based hardware redaction," in 2024 37th International Conference on VLSI Design and 2024 23rd International Conference on Embedded Systems (VLSID), 2024, pp. 517–522.
- [42] Z. Han, M. Shayan, A. Dixit, M. Shihab, Y. Makris, and J. J. Rajendran, "FuncTeller: How well does eFPGA hide functionality?" in 32nd USENIX Security Symposium (USENIX Security 23). Anaheim, CA: USENIX Association, Aug. 2023, pp. 5809–5826. [Online]. Available: https://www.usenix.org/conference/usenixsecurity23/presentation/han-zhaokun
- [43] C. G. Sathe, Y. Makris, and B. C. Schafer, "Mantis: Machine learning-based approximate modeling of redacted integrated circuits," in 2023 Design, Automation & Test in Europe Conference & Exhibition (DATE), 2023, pp. 1–6.
- [44] N. Limaye, E. Kalligeros, N. Karousos, I. G. Karybali, and O. Sinanoglu, "Thwarting all logic locking attacks: Dishonest oracle with truly random logic locking," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 40, no. 9, pp. 1740–1753, 2021.
- [45] S. Takamaeda-Yamazaki, "Pyverilog: A python-based hardware design processing toolkit for verilog hdl," in Applied Reconfigurable Computing (ARC), Apr 2015.
- [46] K. Shamsi, D. Z. Pan, and Y. Jin, "IcySAT: Improved SAT-based Attacks on Cyclic Locked Circuits," in 2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), Nov. 2019, pp. 1–7, iSSN: 1558-2434.
- [47] C. Pilato, A. B. Chowdhury, D. Sciuto, S. Garg, and R. Karri, "ASSURE: RTL locking against an untrusted foundry," IEEE Trans. on VLSI Systems, vol. 29, no. 7, 2021.