ALICE: An Automatic Design Flow for eFPGA Redaction

Chiara Muscari Tomajoli[1*], Luca Collini[1*], Jitendra Bhandari[2], Abdul Khader Thalakkattu Moosa[2], Benjamin Tan[3], Xifan Tang[4], Pierre-Emmanuel Gaillardon[4], Ramesh Karri[2], Christian Pilato[1]
[1]Politecnico di Milano, Italy, [2]New York University, USA, [3]University of Calgary, Canada, [4]University of Utah, USA

ABSTRACT

Fabricating an integrated circuit is becoming unaffordable for many semiconductor design houses. Outsourcing the fabrication to a third-party foundry requires methods to protect the intellectual property of the hardware designs. Designers can rely on embedded reconfigurable devices to completely hide the real functionality of selected design portions unless the configuration string (bitstream) is provided. However, selecting such portions and creating the corresponding reconfigurable fabrics are still open problems. We propose ALICE, a design flow that addresses the EDA challenges of this problem. ALICE partitions the RTL modules between one or more reconfigurable fabrics and the rest of the circuit, automating the generation of the corresponding redacted design.

1 INTRODUCTION

Hardware Intellectual Property (IP) protection is becoming one of the most important concerns during Integrated Circuit (IC) design and manufacturing [7]. Due to the globalization of the supply chain, more semiconductor design houses are forced to outsource IC fabrication to third-party foundries to keep the costs sustainable. However, rogue employees can steal the IC design and make illegal copies [15]. Design houses are using protections like *watermarking*, *split manufacturing*, and *logic locking* to protect the critical parts of their designs [7]. All these methods have their limitations: watermarking is only a *passive* method [1]; split manufacturing requires advanced manufacturing skills [11], and logic locking is challenged by a broad range of attacks [15, 18], especially when the attacker can access a working chip (called *oracle*).

FPGA redaction is a novel, promising technique that aims to thwart reverse engineering attacks by exploiting the flexibility of reconfigurable devices. Critical parts are mapped on and replaced by specific reconfigurable blocks (called embedded FPGAs - eFPGAs) with a two-fold goal: (1) during fabrication, reconfigurable devices can implement any arbitrary functions, without revealing their intended functionality; (2) during execution, they can be configured to implement the correct functionality by classic FPGA programming methods. Figure 1 shows an example, where a module is replaced by a custom eFPGA fabric. Inside, each block represents a Configurable Logic Block (CLB). Modern FPGA specialization tools, like OpenFPGA [19] and FABulous [8], allow designers to start from a HDL module and generate the corresponding soft eFPGA IP that can be integrated and synthesized with the rest of the chip. The FPGA-redaction resilience to SAT attacks comes from a large number of "key bits" to be recovered (i.e., the entire eFPGA configuration bitstream) and a more complex I/O relationship in the eFPGA fabric [4, 10]. Custom eFPGAs have smaller overhead than commercial, off-the-shelf ones [4, 14].

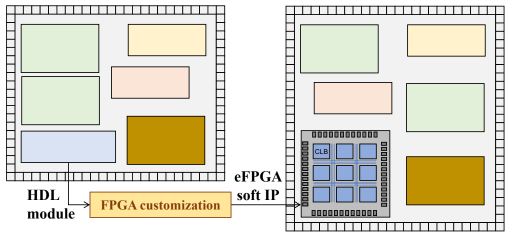


Figure 1: FPGA redaction flow. Critical modules are replaced with custom eFPGA implementations.

FPGA redaction requires the designers to perform several steps. First, it requires them to select the best modules to be redacted from both security and design viewpoints. Then, it requires the creation and integration of the corresponding custom eFPGA fabric. These two problems are strictly interdependent and often application-dependent. For these reasons, designers currently solve these by hand, potentially leading to sub-optimal solutions [6, 14].

This paper focuses on the EDA problem of *partitioning RTL modules* between eFPGA and ASIC and *creating the proper eFPGA fabrics* to implement the redacted modules. While modules implemented in ASIC can be retrieved by the malicious foundry, the flexibility of eFPGAs protects the redacted modules. We propose **ALICE** (Automatic module seLectIon for seCurity-aware EFPGA redaction), a **complete flow to identify the modules to be redacted and generate the corresponding soft eFPGAs**. Starting from the set of candidate redaction modules, ALICE performs a progressive refinement of the solution by filtering out inadmissible modules, clustering the remaining ones to enable the creation of larger eFPGAs, and characterizing them in terms of hardware cost and security resilience to select the best final implementation. After presenting the background of our work (threat model, eFPGA design flow, and related work), we present our main contributions:

- we refine the list of modules to be redacted (Section 4);
- we group independent modules into clusters (Section 5) and we characterize the corresponding eFPGA fabric (Section 6);
- we evaluate our automatic creation of FPGA-redacted designs on common benchmarks for hardware IP protection (Section 7).

Designers can combine functional characteristics (e.g., modules that affect selected outputs), structural characteristics (e.g., maximum number of I/O pins), and eFPGA parameters (e.g., maximum number of eFPGA instances) to guide the redaction process.

[*]C. Muscari Tomajoli and L. Collini contributed equally to this work.

DAC '22, July 10–14, 2022, San Francisco, CA, USA
© 2022 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-9142-9/22/07…$15.00
https://doi.org/10.1145/3489517.3530543

2 BACKGROUND

2.1 Threat Model

We assume the attackers have access to the chip design, can isolate the eFPGA fabric, and have access to an oracle, i.e., a fully-scanned and unlocked design. In this way, they can observe input/output behaviors of this part to apply SAT-based attacks [16]. The attackers have to retrieve the correct bitstream to restore the real functionality. This is the typical threat model for recent eFPGA redaction works [4, 10]. In this scenario, **the eFPGA security comes more from the fabric parameters and way the designer uses the fabrics rather than the specific redacted modules themselves** [3, 4]. We also assume the designers will use state-of-the-art eFPGA parameters from the security viewpoint [3].

2.2 Custom eFPGA Design Flow

Reconfigurable devices can implement any arbitrary function after fabrication by simply changing the configuration bitstream. This is a key feature for hardware IP protection. Designers can integrate the FPGAs as pre-existing blocks in ASIC designs, while their configuration is done only by the final user. The function implemented on the FPGA is thus unknown to the foundry.

Custom eFPGAs can be created with open-source tools, like OpenFPGA [19] or FABulous [8]. Such frameworks allow the automatic customization of FPGA architectures, which are tailored to specific modules with a complete Verilog-to-bitstream flow. For example, Figure 2 shows the OpenFPGA-based customization flow that can be used for eFPGA redaction [4]. OpenFPGA starts from an XML specification of the fabric parameters and produces the corresponding fabrication-ready eFPGA IP [4, 19]. The modules to be redacted will drive the customization of the eFPGA. Using open source frameworks offers additional degrees of freedom to the designer, where one can tune many parameters, as shown in [3]. This will allow the user to come up with architectures that are most suitable for the given design. Thanks to a more tight integration of the soft eFPGA modules, the resulting System-on-Chip architectures can significantly reduce area and performance overheads [10, 14].

In this work, we explore an FPGA architecture composed of Configurable Logic Blocks (CLBs) that are built with four 4-input LUTs, as proposed and evaluated in several recent works [4, 8, 19]. However, we can support any fabric configuration since our work is more focused on how to use them rather than in their generation and security evaluation. Indeed, we can support even off-the-shelf fabrics to be later integrated into the final chip.

2.3 Related Work

Hardware IP protection is a hot topic 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 [2, 13]. For example, logic locking assumes the attacker is not able to retrieve the correct functionality thanks to the protection of a "secret", the locking key [5]. Despite many advances [20], SAT attacks [16] 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 [15, 18].

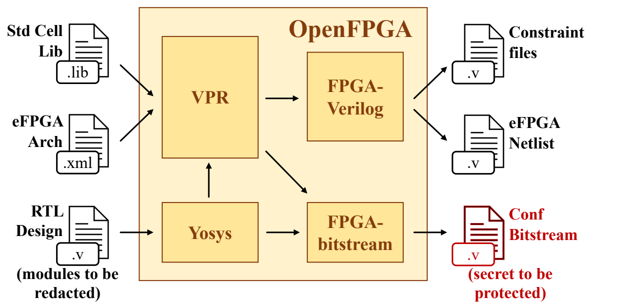


Figure 2: ALICE uses an eFPGA design flow based on OpenFPGA [19]. The eFPGA netlist is integrated with the rest of the chip, while the configuration bitstream is kept secret.

FPGA redaction is a recent technique that aims at implementing 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 configuration of the bitstream that can implement any possible functionality, while (2) end-user attackers that have access to an activated chip cannot retrieve the correct bitstream. In this case, the "secret" corresponds to the configuration bitstream. However, the design of FPGA-redacted ICs is complex, especially in the module partitioning between eFPGA and ASIC.

While recent studies focused on VLSI challenges of eFPGA integration [9], the selection of the modules to be redacted is still 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 [10]. 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) [6]. However, designers may not have an alternative version of the same design to be compare with.

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) [3, 4, 10]. For this reason, we focus more on the selection of the functionality to be redacted (together with its EDA implications), **assuming the fabric configuration as given and "secure"**, and aiming at maximizing the fabric (both I/O and CLB) utilization.

ALICE performs the **automated FPGA redaction of a given design**, identifying the modules that have impact on selected outputs and enabling the possibility of grouping them into the same eFPGA to **maximize its utilization**. ALICE also supports **multiple eFPGA instances** to give more flexibility to the designer.

3 ALICE DESIGN FLOW FOR EFPGA REDACTION

Our redaction flow is shown in Figure 3. It starts from the RTL description of the design to be redacted in Verilog[1] and a set of parameters for the flow (in a custom YAML configuration file). Such parameters include eFPGA fabric configurations (e.g., as specified in

[1]Limitations are only due to the HDL parser that we use. Supporting another HDL language (e.g., VHDL) only requires the proper parser.

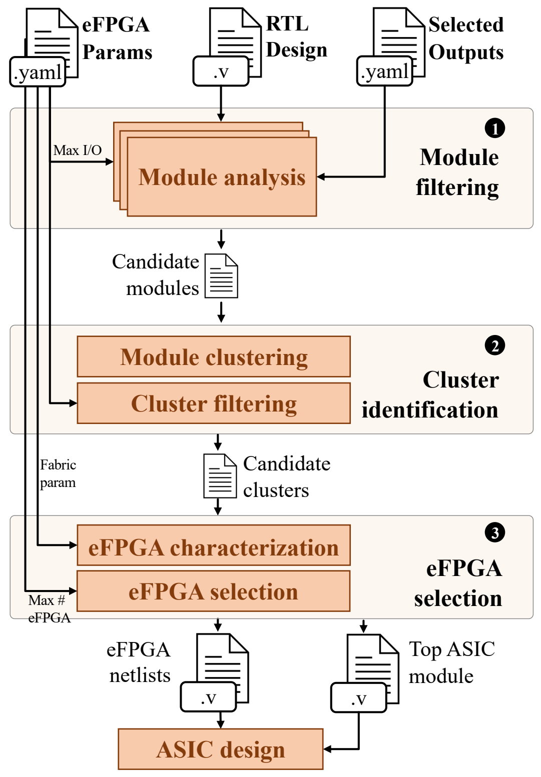


Figure 3: ALICE flow for automatic eFPGA-based redaction.

the OpenFPGA configuration file), the maximum number of eFPGAs to be instantiated, and the maximum number of I/O for each of them. The number of I/O pins is also a rough indication of the type of eFPGA that the designer aims at using. For example, a 4×4 fabric configuration has no more than 64 I/O pins [4, 10]. Currently we support (one or more) eFPGAs with the identical fabric architecture and maximum number of used I/O pins. While we contend that this setup will create a more regular physical design, adding support in the future for eFPGAs with different configurations is possible.

ALICE focuses on how to partition an RTL design and generate the corresponding eFPGA-enhanced IC with three main phases: **module filtering**, **cluster identification**, and **eFPGA selection**. During module filtering, we analyze the design to identify *candidate redaction modules*, while discarding the ones that do not satisfy specific constraints. In the second phase, the candidate modules are clustered into *candidate module clusters*. Again, solutions that do not satisfy specific constraints are discarded. The result of this phase is a set of candidate clusters that are then characterized by running the flow for the creation of the corresponding eFPGA fabrics. We finally apply an algorithm to select the eFPGAs that maximize our objectives (i.e., minimum hardware overhead and maximum security) with no overlapping sets of redacted modules. The resulting redacted RTL description is reproduced along with the fabrics of the selected eFPGAs. The final output is the description of the final system ready for ASIC design. ALICE is a modular flow that can be extended with additional criteria for selection. It can also interface with other eFPGA tools for characterization and include further metrics for security assessment, if needed.

Algorithm 1: ALICE module filtering

```
Input: Input RTL design $D$, eFPGA parameters $P$, list of selected outputs $O$
Output: Set of candidate redaction modules $R$
1  $M \leftarrow$ ExtractInitialModules($D$)      // Analyze input RTL design.
2  $S \leftarrow \emptyset$
3  foreach $m \in M$ do
4      $S[m] \leftarrow 0$
5  end
6  foreach $o \in O$ do
7      $T \leftarrow$ IdentifyModules($M, o$) // Compute modules $T$ affecting $o$
8      UpdateScore($T, S$)          // Increment scores of modules $T$
9  end
10 $F \leftarrow$ RankAndSelect($M, S$)        // Select most relevant modules
11 $R \leftarrow \emptyset$
12 foreach $f \in F$ do
13     if CheckParameters($f, P$) then
14         $R \leftarrow R \cup \{f\}$
15 end
16 return $R$
```

4 MODULE FILTERING

This phase analyzes the input design to determine the list of RTL modules that must be considered for redaction. Algorithm 1 shows the pseudocode of our procedure. Our algorithm starts from the input design D, the list of eFPGA parameters P (e.g., maximum number of I/O pins), and the list of selected output O. The designers can provide a list of outputs that they want to "protect". The algorithm then applies *functional* and *structural* criteria to obtain the final set R of candidate redaction modules. Functional criteria aim at identifying modules that are more important for FPGA redaction from the functionality viewpoint. Structural criteria aim at identifying modules that can be effectively implemented with eFPGA, excluding the ones that would lead to an unfeasible solution.

We list the modules M of the input design D (line 1), assigning an initial zero score for each of them (lines 3-5). We create the dataflow graph of the entire RTL design and, for each selected output, we increase the scores of the modules that have a direct impact on it (lines 6-9). We select the top-score modules and add them to the list F of functionally-relevant modules for redaction (line 10).

In the next stage, we apply structural criteria to each functionally-relevant module for redaction (lines 12-15). We check whether each module is compatible with the given eFPGA parameters (line 13). For example, we compute the number of I/O pins of the module to check if it fits into the potential eFPGA fabric. If the module satisfies the constraints, it can be added to the final list R (line 14).

The list R represents feasible modules that affect a relevant number of (selected) outputs and can be clustered or implemented alone in an eFPGA (depending on their size). This phase can be easily extended with more module-level filtering criteria.

5 CLUSTER IDENTIFICATION

Given the set of candidate modules R, we find all valid combinations (*clusters*) that can be redacted onto an eFPGA. A cluster can be composed of a single module (*single-module redaction*) or a set of independent modules (*multi-module redaction*). In both cases, the cluster is valid if the corresponding eFPGA implementation is admissible (i.e., it respects the given designer's constraints).

Algorithm 2 shows the pseudo-code of the procedure used in ALICE. It performs a *fixed-point analysis* to identify the set C of

Algorithm 2: ALICE cluster identification

```
Input: Set of candidate redaction modules $R$, eFPGA parameters $P$
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                 if $N \not\subset D \wedge N \not\subset C \wedge$ CheckParameters$(N, P)$ then
13                     $D \leftarrow D \cup N$
14                 end
15             end
16         end
17     end
18     $Flag \leftarrow False$
19     if $D \neq \emptyset$ then
20         $C \leftarrow C \cup D$
21         $Flag \leftarrow True$
22     end
23 while $Flag$
24 return $C$
```

all candidate module clusters. Each of them is meant to fit into a single eFPGA. We initialize the set C with clusters composed of the single modules identified in the previous phase (lines 2-4). We then proceed iteratively to create new and larger clusters (lines 6-23). This part recombines each pair of admissible clusters (lines 8-17). If the cluster was not already identified in the previous iterations and it respects the input constraints (line 12), it is added to the list of current clusters (line 13). Indeed, each cluster is analyzed with the same structural criteria used for the modules. For example, in the case of multi-module redaction, the number of I/O pins is the aggregated number of the I/O pins of the single modules. The cluster is admissible if it respects the limit given by the designer. At the end of each iteration, new clusters (line 19) are added to the set C and the procedure restarts. We terminate our algorithm when, after recombining the current clusters, it is not possible to create new ones. Each element of C is a candidate module cluster.

6 EFPGA SELECTION

Each candidate module cluster in C can be implemented by an eFPGA. The set of resulting candidate implementations must be now characterized, ranked, and selected to determine the final solution. In this phase, we evaluate all candidate clusters to determine whether the corresponding eFPGA fabrics are admissible, determine all feasible solutions, and select the best and final one.

Algorithm 3 shows the pseudocode used in ALICE. First, we generate the top module corresponding to each candidate cluster and run the selected eFPGA customization flow (i.e., OpenFPGA in our case) on it (lines 2-7). In the case of multi-module redaction, we create a top Verilog module that instantiates all independent modules. OpenFPGA returns the corresponding fabric if the design is feasible and an error otherwise (e.g., when the cluster modules cannot be implemented for any reason). Since the designer can specify the range of permitted fabric sizes, we also check that the resulting fabric is admissible (line 4). If so, the fabric is added to the list F of valid implementations (line 5). At this point, we give a score to each fabric implementation (line 8). The score combines information about I/O and CLB utilization as follows:

Algorithm 3: ALICE eFPGA selection

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

$$T_f = \alpha \cdot \frac{MaxIOUtil - IOUtil_f}{MaxIOUtil} + \beta \cdot \frac{MaxCLBUtil - CLBUtil_f}{MaxCLBUtil} \quad (1)$$

where $IOUtil_f$ and $CLBUtil_f$ represent the I/O and CLB utilization, respectively, while $MaxIOUtil$ and $MaxCLBUtil$ represent the corresponding maximum values for all analyzed eFPGAs. In this way, both contributions range between 0 and 1. α and β are two user-defined parameters to balance the contributions. The score T embeds information related to security resilience. Indeed, eFPGA implementations with poor I/O utilization are more prone to attacks because it is easier to identify stuck-at-0 outputs. Similarly, fabrics with low CLB utilization have less logic to be (successfully) recovered [3, 4]. We then use a *branch&bound algorithm* to enumerate all possible eFPGA combinations that can be redacted together (lines 11-23) and obtain the full set of solutions. In particular, we start from an empty working solution (line 9) and, at each step, we aim to add a new eFPGA implementation to each current working solution (lines 12-22). A solution represents a set of eFPGAs with no overlapping module instances. If the solution is final (i.e., it reaches the maximum number of allowed eFPGAs or it redacts all the admissible modules), it is added to the final set of solutions (line 16). Otherwise, we keep it in the working list for further expansion (line 19). At the end of this phase, the set S contains the total set of admissible solutions. We now assign a score to each of them. The score of a solution is the sum of the scores of its eFPGA implementations, each of them obtained with Eq. 1. We rank the set S according to the score and the one with the highest score is the best and final solution (line 25).

Table 1: Characteristics of the selected benchmarks

Suite	Design	Modules (#)	Instances (#)	I/O pins [min, max]
CEP	DES3	11	11	[12, 301]
	FIR	5	5	[64, 384]
	IIR	5	5	[66, 384]
	SHA256	3	3	[38, 774]
IWLS05	SASC	2	3	[23, 28]
	USB_PHY	3	3	[17, 33]
OpenROAD	GCD	10	11	[6, 68]

The final solution includes a set of eFPGA implementations, each of them containing a list of module instances to be redacted. At this point, we need to regenerate the top module for ASIC implementation (*Top ASIC module* in Figure 3) where we replace the redacted instances with the corresponding eFPGA instances. In case of multi-module redaction, the different modules can be spread around the design. In this case, we apply a *dominator tree* analysis on the module hierarchy to identify the best point where to insert the eFPGA instance and minimize wire length. Signals from the original instances are re-routed to the corresponding eFPGA instance, while its control signals are propagated to the top module. We also remap the module signals to the eFPGA GPIO signals for correct connection. The updated design, along with the fabric netlists, can be given to physical design tools.

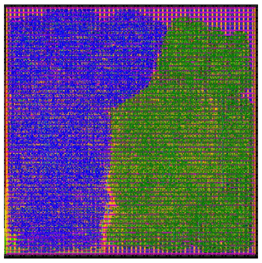
7 EXPERIMENTAL EVALUATION

We implemented a prototype of ALICE in Python, using the PyVerilog framework [17]. PyVerilog can parse the Verilog designs, analyze, and manipulate the resulting Abstract Syntax Tree (AST), and regenerate the output files, including the ones fed into the OpenFPGA tool chain for eFPGA creation. Table 1 shows the benchmarks that we used to validate ALICE. They are commonly used to evaluate RTL locking [12]. The table reports the number of modules and the instances that can be redacted. We report the range of the I/O pin count for such modules. For each design, we identified the main output(s) to be given to the module filtering phase.

We configure ALICE to run with two configurations. 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. These experiments will show how to use ALICE to implement more but smaller eFPGAs or fewer but larger eFPGAs. We set $\alpha = \beta = 1$ (Eq. 1) in both cases.

We run the OpenFPGA flow to implement the eFPGA fabrics composed of 4-input fracturable LUTs, 4 logic elements for each CLB, and 8 GPIOs for each I/O tile. Future work will explore these eFPGA parameters. Each OpenFPGA run aims at identifying the most suitable fabric (i.e., the one with minimum size) to implement the given module(s). We finally validated the designs with Cadence Genus 18.14 for logic synthesis and Cadence Innovus 18.10 for physical design, targeting the NanGate 45nm Open Cell Library.

Table 2 shows the results that we obtained when running ALICE on the benchmarks with the two configurations. In particular, we report: the number of candidate redaction modules ($|R|$), as obtained after *module filtering*, the number of candidate module clusters that are created ($|C|$), as obtained after *cluster identification*, the total number of valid eFPGA implementations, and the number of total solutions ($|S|$), as obtained during *eFPGA selection*. We also report the characteristics of the eFPGAs in the final solution, along with the total number of redacted modules.


(a) `cfg1`: two 4×4 (52,629μm^2)

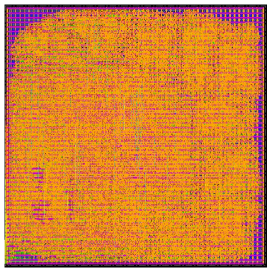

(b) `cfg2`: one 5×5 (54,512μm^2)

Figure 4: Physical layouts of two GCD solutions with different number of eFPGAs. The figures are in scale.

In all benchmarks except IIR in the first configuration, we are able to find at least a feasible solution for the given eFPGA parameters. In the case of IIR in `cfg1`, the smallest modules have already more I/O pins (66 pins – see Table 1) than the maximum allowed I/O pin count of the eFPGA (64 pins). Indeed, the filtering phase does not produce any valid candidate redaction module and the flow cannot continue. Increasing the number of I/O pins to 96 (`cfg2`) allowed us to find a solution, showing how **ALICE can guide the designer in the identification of modules to be redacted**.

DES3 and GCD present other interesting results. They have more instances than the other benchmarks and ALICE is able to find several candidate clusters in both configurations (more than 200 for DES3 and at least 19 for GCD). Both GCD and DES3 have modules with a high variance in the number of I/O pins. So we can create multi-module redaction clusters when combining modules with low and high number of I/O pins, but clusters having modules with many I/O pins become invalid. Also, when using more eFPGAs (`cfg1`), the number of possible cluster combinations and, in turn, solutions grows significantly. The tool is then able to find solutions with two smaller eFPGAs (two 8×8 for DES3 and two 4×4 for GCD) or one larger eFPGA (14×14 for DES3 and 5×5 for GCD). The designers can use these results in different ways. For DES3, they can use the second implementation with a 14×14 eFPGA because it redacts many more modules than the first case. For GCD, the two solutions are equivalent from the area viewpoint (see data in Figure 4) and have almost the same number of redacted modules. The designer could be motivated to use the first solution because it requires the attacker to recover more bit-streams. Figure 4 shows screenshots of the two physical designs for GCD. This testcase is small and so most of the chip is occupied by the eFPGAs. However, the same modules will become less relevant when the component is inserted into a larger system-on-chip (like PicoSoc in [4]). In all cases, **area/time/power overheads are in line with previous studies on FPGA redaction [4, 10] as they are more related to the fabric architectures and sizes rather than the specific modules that are redacted**.

Table 2 reports also the execution time of each phase. Note that *module filtering* includes the time spent for dataflow analysis to

Table 2: Results after running ALICE with two different configurations.

Configuration	Design	# Instances	Module filtering		Cluster identif.		eFPGA selection				
			Time	$\|R\|$	Time	$\|C\|$	Time	# valid eFPGAs	$\|S\|$	eFPGA size	# redacted modules
`cfg1`: 64 I/O pins and 2 eFPGAs	DES3	11	289.21s	8	0.96s	218	905.12s	216	2,105	8×8, 8×8	4
	FIR	5	0.17s	1	<0.01s	1	1.43s	1	1	6×6	1
	IIR	5	1.36s	0	-	-	-	-	(n.a.)^1	-	-
	SHA256	3	12.87s	1	<0.01s	1	4.80s	1	1	12×12	1
	SASC	3	0.20s	1	<0.01s	1	1.36s	1	1	7×7	1
	USB_PHY	3	2.03s	2	<0.01s	3	5.99s	1	1	7×7	1
	GCD	11	0.39s	9	0.01s	28	32.10s	19	76	4×4, 4×4	2
`cfg2`: 96 I/O pins and 1 eFPGA	DES3	11	295.65s	8	1.17s	255	1,093.03s	255	245	14×14	8
	FIR	5	0.15s	3	<0.01s	3	16.76s	3	3	6×6	1
	IIR	5	0.29s	2	<0.01s	2	24.40s	2	2	15×15	1
	SHA256	3	12.68s	1	<0.01s	1	4.83s	1	1	12×12	1
	SASC	3	0.20s	1	<0.01s	1	1.34s	1	1	7×7	1
	USB_PHY	3	1.92s	2	<0.01s	3	5.77s	1	1	7×7	1
	GCD	11	0.45s	10	0.05s	70	91.28s	37	33	5×5	3

^{1}The module with the minimum I/O count already exceeds the maximum I/O size of the eFPGA (see Table 1).

determine which modules affect the selected outputs, while *eFPGA selection* includes the time to run OpenFPGA on all valid solutions. Dataflow becomes more complex as the complexity (and not necessarily the number) of the RTL modules increases and is mostly independent of the configuration parameters. For example, SHA256 takes more time than GCD even if it has only three modules. Running OpenFPGA is generally very fast, except for large eFPGA instances. However, it is always in the order of tens of seconds in the worst case (see IIR with two large solutions in `cfg2`). In general, the time for the *eFPGA selection* phase grows linearly with the number of solutions to be tested. In FIR, even if the solutions for the two configurations are the same, the `cfg2` analysis takes much longer because there are more and larger eFPGA candidates (up to 19×19) that are later excluded because of low score.

8 CONCLUSIONS AND FUTURE WORK

This paper presented ALICE, a methodology for automatic eFPGA redaction. ALICE analyzes the given RTL design, identifies the modules that have an effect on selected outputs, and cluster them. Such clusters are then characterized with an open-source FPGA customization flow and we select the ones that can maximize security. We show that ALICE can identify different solutions based on the given parameters (e.g., maximum number of I/O pins and eFPGA instances). Our flow can be part of a larger exploration that co-optimizes eFPGA parameters and module selection. It can also include a pre-processing step to perform fine-grained redaction: It can decompose large modules into smaller instances so that only part of them are effectively redacted.

ACKNOWLEDGEMENTS

R. Karri was supported in part by ONR Award # N00014-18-1-2058, NSF Grant # 1526405, NYU Center for Cybersecurity, and NYUAD Center for Cybersecurity. X. Tang and P.-E. Gaillardon were supported in part by DARPA, under the grants # FA8650-18-2-7855 and # FA8650-18-2-7849.

REFERENCES

[1] A. Abdel-Hamid, S. Tahar, and E. Aboulhamid. 2004. A Survey on IP Watermarking Techniques. *Design Automation for Embedded Systems* 9 (2004), 211–227.
[2] Z. U. Abideen and S. P. T. D. Perez. 2021. From FPGAs to Obfuscated eASICs: Design and Security Trade-offs. In *IEEE AsianHOST*. 1–4.
[3] J. Bhandari, A. K. T. Moosa, B. Tan, C. Pilato, G. Gore, X. Tang, S. Temple, P.-E. Gaillardon, and R. Karri. 2021. Not All Fabrics Are Created Equal: Exploring eFPGA Parameters For IP Redaction. arXiv:2111.04222
[4] J. Bhandari, A. K. Thalakkattu Moosa, B. Tan, C. Pilato, G. Gore, X. Tang, S. Temple, P.-E. Gaillardon, and R. Karri. 2021. Exploring eFPGA-based Redaction for IP Protection. In *IEEE/ACM ICCAD*.
[5] A. Chakraborty, N. G. Jayasankaran, Y. Liu, J. Rajendran, O. Sinanoglu, A. Srivastava, Y. Xie, M. Yasin, and M. Zuzak. 2020. Keynote: A Disquisition on Logic Locking. *IEEE Trans. on CAD of Integrated Circuits and Systems* 39, 10 (2020).
[6] J. Chen, M. Zaman, Y. Makris, R. D. S. Blanton, S. Mitra, and B. C. Schafer. 2020. DECOY: DEflection-Driven HLS-Based Computation Partitioning for Obfuscating Intellectual PropertY. In *ACM/IEEE Design Automation Conference (DAC)*. 1–6.
[7] W. Hu, C.-H. Chang, A. Sengupta, S. Bhunia, R. Kastner, and H. Li. 2021. An Overview of Hardware Security and Trust: Threats, Countermeasures, and Design Tools. *IEEE Transactions on CAD of Integrated Circuits and Systems* 40, 6 (2021).
[8] D. Koch, N. Dao, B. Healy, J. Yu, and A. Attwood. 2021. FABulous: An Embedded FPGA Framework. In *ACM/SIGDA FPGA*. 45–56.
[9] P. Mohan, O. Atli, O. Kibar, M. Zackriya, L. Pileggi, and K. Mai. 2021. Top-down Physical Design of Soft Embedded FPGA Fabrics. In *ACM/SIGDA FPGA*. 1–10.
[10] P. Mohan, O. Atli, J. Sweeney, O. Kibar, L. Pileggi, and K. Mai. 2021. Hardware Redaction via Designer-Directed Fine-Grained eFPGA Insertion. In *DATE*.
[11] T. D. Perez and S. Pagliarini. 2020. A Survey on Split Manufacturing: Attacks, Defenses, and Challenges. *IEEE Access* 8 (2020), 184013–184035.
[12] C. Pilato, A. B. Chowdhury, D. Sciuto, S. Garg, and R. Karri. 2021. ASSURE: RTL Locking Against an Untrusted Foundry. *IEEE Trans. on VLSI Systems* 29, 7 (2021).
[13] J. J. V. Rajendran. 2017. An overview of hardware intellectual property protection. In *IEEE International Symposium on Circuits and Systems (ISCAS)*. 1–4.
[14] P. D. Schiavone, D. Rossi, A. Di Mauro, F. K. Gürkaynak, T. Saxe, M. Wang, K. C. Yap, and L. Benini. 2021. Arnold: An eFPGA-Augmented RISC-V SoC for Flexible and Low-Power IoT End Nodes. *IEEE Trans. on VLSI Systems* 29, 4 (2021), 677–690.
[15] K. Shamsi, M. Li, K. Plaks, S. Fazzari, D. Z. Pan, and Y. Jin. 2019. IP Protection and Supply Chain Security through Logic Obfuscation: A Systematic Overview. *ACM Trans. on Design Automation of Electronic Systems (TODAES)* 24, 6 (2019), 1–36.
[16] P. Subramanyan, S. Ray, and S. Malik. 2015. Evaluating the security of logic encryption algorithms. In *IEEE HOST*. 137–143.
[17] S. Takamaeda-Yamazaki. 2015. Pyverilog: A Python-Based Hardware Design Processing Toolkit for Verilog HDL. In *Applied Reconfigurable Computing (ARC)*.
[18] B. Tan, R. Karri, N. Limaye, A. Sengupta, O. Sinanoglu, M. M. Rahman, S. Bhunia, D. Duvalsaint, R. D. Blanton, et al. 2020. Benchmarking at the Frontier of Hardware Security: Lessons from Logic Locking. arXiv:2006.06806
[19] X. Tang, E. Giacomin, B. Chauviere, A. Alacchi, and P.-E. Gaillardon. 2020. OpenFPGA: An Open-Source Framework for Agile Prototyping Customizable FPGAs. *IEEE Micro* 40, 4 (2020), 41–48.
[20] M. Yasin and O. Sinanoglu. 2017. Evolution of logic locking. In *IFIP/IEEE VLSI-SoC*.