Exploring eFPGA-based Redaction for IP Protection

Jitendra Bhandari*, Abdul Khader Thalakkattu Moosa*, Benjamin Tan*, Christian Pilato†,
Ganesh Gore‡, Xifan Tang‡, Scott Temple‡, Pierre-Emmanuel Gaillardon‡ and Ramesh Karri*
*New York University, New York, USA
†Politecnico di Milano, Milan, Italy
‡University of Utah, Utah, USA

***Abstract*—Recently, eFPGA-based redaction has been proposed as a promising solution for hiding parts of a digital design from untrusted entities, where legitimate end-users can restore functionality by loading the withheld bitstream after fabrication. However, when deciding which parts of a design to redact, there are a number of practical issues that designers need to consider, including area and timing overheads, as well as security factors. Adapting an open-source FPGA fabric generation flow, we perform a case study to explore the trade-offs when redacting different modules of open-source intellectual property blocks (IPs) and explore how different parts of an eFPGA contribute to the security. We provide new insights into the feasibility and challenges of using eFPGA-based redaction as a security solution.**

***Index Terms*—Embedded FPGA, Hardware Security, Redaction**

I. Introduction

In response to concerns about the integrity of the integrated circuit (IC) supply-chain, researchers have proposed numerous design obfuscation and locking techniques to protect hardware intellectual property blocks (IPs) [1]–[10]. Such techniques involve supplementing designs so as to induce errors in the presence of incorrect key inputs (e.g., adding XOR/XNOR gates randomly [10]) or introducing structures that legitimate users later populate with elements of the design that are withheld during fabrication (e.g., restoring withheld constants [1]). Programmable fabrics have been added to the repertoire of defenses against reverse engineering and IP piracy, especially as a counter to Boolean satisfiability-based (SAT) attacks [11] and variants thereof [2]. In *reconfigurable fabric-based redaction*, designers select parts of a design and implement it by programming a fabric separate from the remaining design, as shown in Fig. 1. A potentially un-trusted foundry manufactures the design without the programming information for the fabric (e.g., the configuration bit-stream) which the designer withholds. Fabrics include embedded field programmable gate arrays (eFPGAs) [4], [6], coarse-grained reconfigurable architectures [12], and transistor fabrics [13].

Benjamin Tan and Ramesh Karri are supported in part by the Office of Naval Research under Award Number #N00014-18-1-2058. This work was supported in part by NYU CCS. Ganesh Gore, Xifan Tang, Pierre-Emmanuel Gaillardon are supported by AFRL and DARPA under agreement number FA8650-18-2-7855, and Scott Temple, Pierre-Emmanuel Gaillardon are supported by AFRL and DARPA under agreement number FA8650-18-2-7849. Jitendra Bhandari and Abdul Khader Thalakkattu Moosa contributed equally to this work.

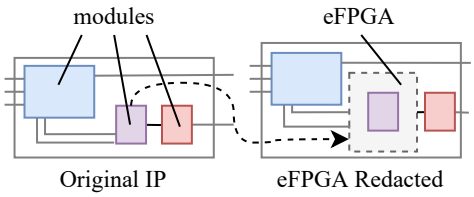


Fig. 1. eFPGA-based redaction takes a module of an IP and replaces it with a reprogrammable fabric that can replace the redacted functionality.

When adopting eFPGA-based redaction for IP protection, designers are faced with a number of decisions and design challenges. These include customizing/selecting the fabric configuration, deciding which of the module(s) in the IP to move to the eFPGA, and dealing with the overheads that result from the process of integration and implementation in the ASIC design flow. Prior work has begun to explore these challenges by selecting functionality to redact from high-level (C-based) designs to maximize a security metric until hitting an area overhead threshold [6] or by choosing a single part of a design—at the register transfer level (RTL)—which guides the generation of an eFPGA fabric [4]. As with many security solutions, there is a trade-off between security and other design factors, such as area and timing [4]–[6]. Thus far, prior work has suggested eFPGAs for feasibly redacting an individual IP [4], [6]; however, given the nascent state of eFPGA-based redaction, we need more insights into the practical implications of using this IP protection approach so as to help designers make better redaction decisions.

In this work, we provide key insights into the use of eFPGAs for redacting RTL designs through a case study of redacting different hand-crafted hierarchical RTL IPs, where different modules are candidate units for redaction. We investigate adapting an open-source FPGA design flow [14] to produce different eFPGAs configurations, depending on the module to be redacted, and assess the impact on a range of open-source IPs. We explore factors that contribute to the security offered by eFPGAs-based redaction, and explore the factors that contribute to the security of eFPGA fabrics. The three main contributions are:

- insights from a security evaluation of eFPGA-based redaction based on different redaction decisions, under an oracle-based, scan-chain accessible attack model

- results from a case study using open-source IPs to explore challenges and area/timing trade-offs of different redaction decisions, using an open-source FPGA tool flow [14]
- insights and guidance for future work in eFPGA-based redaction of RTL designs

In Section II, we provide the context of our work and describe the potential of open-source eFPGAs for hardware security. In Section III, we provide an overview of an eFPGA-based redaction flow, with a particular focus on the decision and challenges faced by designers. Section IV is our deep dive into the characteristics of eFPGA fabrics that provide security benefits of redaction. We present the impact of different module choices using a set of open-source IPs from locking/obfuscation work, in Section V, and discuss insights from our study in Section VI.

II. Motivation and Background

A. Related Work: Intellectual Property Protection

Traditional techniques for hardware IP protection include locking-based methods, where designers insert additional gates (controlled by an input key) to thwart reverse engineering [2], [3], [15]–[17]. When applied at high levels of abstraction, such as RTL, these methods are effective because they hide the essential semantics of the design, but entail significant increases in overhead [1]. However, there is an ongoing back-and-forth battle between attacks and locking-based defenses, where Oracle-guided attacks [11], [18]–[21] and existence of structural artifacts [22], [23] pose considerable challenges to defenders.

eFPGAs, comprising configurable logic blocks (CLBs) containing look-up tables (LUTs), flip-flops, and routing logic, can be programmed to implement arbitrary functionality. This allows a designer to implement "sensitive" parts of the design in an eFPGA, post-fabrication, by means of setting the configuration bit-stream— unseen by potentially untrusted parties during manufacture. Hence, eFPGA-based redaction is a potential panacea for reverse-engineering attacks; the regular structure of an eFPGA, avoids apparent structural biases while appearing to pose a challenge to attackers by introducing a large key-space (i.e., configuration bitstream) [4], [6]. Using an eFPGA for redaction offers expressiveness and complexity compared to focusing on replacing parts of a design with LUTs [8] or by obfuscating routing structures [7].

However, how to identify portions of a design to redact is an open issue; the designer must not only identify the "sensitive" parts but also decide how much of them can be moved onto the eFPGA. This requires careful evaluation of the security benefits of eFPGA implementation while limiting overhead. Prior work studied this problem from a high-level synthesis (HLS) perspective, where security is explored in terms of operations redacted and the number of cells on which those operations are mapped [6], or by mapping the logic that differs between variants of the same functionality [5]. While HLS studies provide insight into targets for redaction, they do not fully characterize practical implications of an eFPGA fabric to support redaction; for instance, the eFPGA fabric dimensions needed and the fabric interfaces are not apparent.

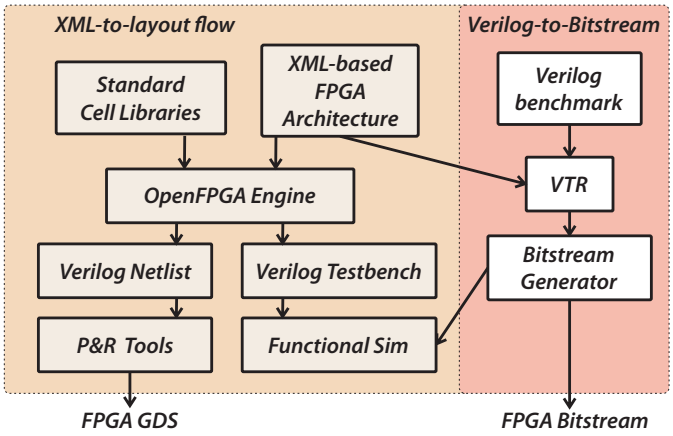


Fig. 2. Open-source eFPGA design flows: (a) XML-to-layout generation for chip designers; and (b) Verilog-to-Bitstream generation for end-users.

"Designer-directed" redaction at the register transfer level using a custom fabric generation flow [4] produces a fabric for parts of the design the designer intuits as "security critical". The security analyses show that the fabric offers a promising level of SAT-attack resilience, but the evaluation is limited to a small number of designs. Our study sheds light on practical issues with eFPGA redaction at the register transfer level by extending the analysis of prior work. We adapt an open-source FPGA design flow and redact a wider range of IPs.

B. Open Source (e)FPGA Design Flows

The trends in heterogeneous computing have increased interest in embedded field programmable gate arrays (eFPGAs) fabrics due to their flexibility and adaptability. In commercial products, FPGAs are tightly integrated to processors in a single-chip, serving as a co-processor or a programmable accelerator [24], [25]. Thanks to eFPGAs, the peak performance of a System-on-Chip (SoC) can be improved by 3.4× along with a 2.9× power reduction. Different SoCs require different eFPGA fabrics from architecture to physical layouts, depending on the application context. For instance, eFPGAs designed for machine learning applications require a high density of *Digital Signal Processing* (DSP) blocks, embedded memories, and architectural enhancements which can implement *Multiply-accumulate* (MAC) operations efficiently.

Driven by demand, there are a few open-source tools to prototype customizable (e)FPGAs [14], [26], [27]. Fig. 2 illustrates principles of OpenFPGA framework to prototype customizable eFPGAs [14]. In the *XML-to-layout flow*, chip designers can generate fabrication-ready eFPGA layouts using well-known XML-based architecture description languages [28], [29]. The architecture description languages allow designers to customize FPGA architecture down to circuit elements, supporting standard cells and highly flexible hardware IPs. The core engine converts the architecture description to synthesizable or technology-mapped Verilog netlists that

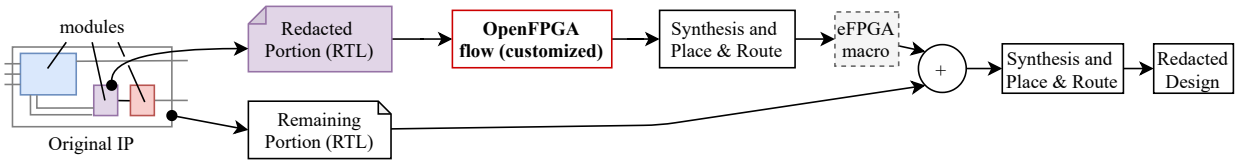


Fig. 3. An eFPGA-based redaction flow for RTL IP, where the redacted portion (module) is manually picked by a designer. We adapt the OpenFPGA flow to produce the required eFPGA fabric, which we then treat as a macro to be connected to the remaining portion of the design.

model a complete eFPGA fabric. Then, the auto-generated Verilog netlists can be fed into established ASIC design tool suites, especially *Place&Route* (P&R) tools, for generating GDSII layouts and performing sign-offs. In addition, self-testing Verilog testbenches can be automatically generated to ease pre- and post-layout verification. The Verilog testbenches can validate the correctness of an eFPGA by simulating a complete process in practice, including bitstream downloading and eFPGA operation. As argued in prior work, the ability to create custom, small, fabrics can provide a better fit for redaction compared with off-the-shelf eFPGA IP [4].

In the *Verilog-to-Bitstream flow*, end-users can implement HDL designs on the eFPGAs. HDL designs are first synthesized by Yosys [30] and physically mapped (packed, placed and routed) on the eFPGA programmable resources by VPR [31]. The implemented design is translated to a bitstream which is compatible with configuration protocols of eFPGAs.

Open-source efforts aim to overcome two major technical barriers of contemporary eFPGA development: (1) the time-consuming physical design process—by leveraging the sophisticated ASIC design tools rather than manual layouts, and (2) the ever increasing design complexity of associated electronic design automation (EDA) tool-chain—by using well-known open-source FPGA architecture exploration tools, e.g., VPR [31], rather than developing ad hoc, in-house tools. Previous work has shown that using the design flows in Fig. 2, the development cycle of a 160k-LUT FPGA layout is ∼24 hours and its performance is competitive against commercial products [14], [32]. In this paper, we thus investigate the OpenFPGA framework to implement eFPGA fabrics [14] for redaction and give insight into our experience.

III. Redaction and Practical Challenges

A. Overview of the Redaction Process

Fig. 3 illustrates a general eFPGA-based redaction flow for redacting individual, hierarchically designed IPs at the register transfer level. Typically, designers redact a module after designing them [1], [4], which points towards hand-crafted modules in the IP as the starting point for potential redaction targets. To prepare the redaction fabric, we run the selected module through the OpenFPGA framework [14] which selects and generates the smallest eFPGA fabric configuration given an architecture definition. We simulate the generated fabric to verify that the intended functionality is correct, and if so, take the synthesizable Verilog netlist through a physical design flow that comprises synthesis, followed by floorplanning, placement and routing. In contrast to related work [4], we treat the eFPGA fabric as a *macro*. After integrating this macro with the rest of the design, the IP, as a whole, is put through the design flow, resulting in a final GDSII file.

B. Practical Considerations

There are several practical considerations that designers need to keep in mind when using eFPGAs for redaction, including the fabric utilization, impact on timing, and the area overhead introduced by integrating a fabric.

Resource Utilization: When one redacts an IP, the selected module(s) (the "redaction modules") needs to fit into the eFPGA fabric; designers need to be aware of the resources available in a particular fabric size, especially if one were to adopt an HLS-based "top-down" approach to increase cell usage [6]. The alternative approach is to find a fabric size that matches the requirements of the "designer-directed" redaction choice [4]. However, the minimum fabric size is driven by different factors of the redaction module. The interface of the module (number of inputs and outputs) will affect the number of I/O tiles required, while the number of state elements (registers/flip-flops) will affect the number of CLBs. Either factor can dominate the final eFPGA size, causing a sub-par use of the fabric used for redaction.

Timing: The chosen redaction unit can possibly lie in the critical path. FPGA structures tend to have longer delays compared to full ASIC designs due to the general nature of the large pool of available gates for logic and routing. Thus, the redacted portion in the eFPGA will likely be slower compared to the same design implemented directly in the ASIC. The designer should be aware of the impact on the overall IP's timing characteristics, including the effect of the redacted portion, otherwise the targeted performance is compromised.

Area: In addition to timing issues, the introduction of an eFPGA fabric will have considerable implications on area, particularly as the number of CLB and I/O tiles increases non-linearly with each increase in the square eFPGA fabric's dimensions. This places another constraint on the design portion selected for the redaction—a redaction choice that requires a fabric that encompasses too much area, in the context of the IP as a whole, could be too impractical. In a related vein, the module selected for redaction could have numerous instances in the IP; the designer could possibly create a larger fabric to redact several instances, instantiate multiple eFPGAs, or possibly choose to redact only one.

To gain insights into these practical considerations, we explore the fabrics needed to redact different parts of typi-

cal IPs (presented in Section V). On top of these practical considerations, we need to consider the security implications of eFPGA fabrics—we explore this in the next section.

IV. Security Analysis of eFPGA Fabrics

For more insight into the security offered by using eFPGA-based redaction, we explore and discuss their characteristics, in particular, SAT attack resilience, given the miter-based SAT attack's strength against other locking/obfuscation approaches [2]. Previous work has suggested that large FPGA bitstream lengths make SAT-based attacks impractical [6] and the results in Mohan *et al.*'s evaluation appear to support this claim [4]. In this work, we begin to investigate how the different structural elements of the eFPGA contribute to SAT resilience. We also perform a security evaluation of different fabric sizes in a high performance computing (HPC) environment, with jobs running on a compute node with an Intel Xeon Platinum 2.9 GHz with 64–512 GiB of RAM.

A. Threat Model and Assumptions

We consider an attack model favoring an attacker with access to a fully-scanned and unlocked design (i.e., an oracle) in addition to the netlist of the IP; this is typical from prior work [4]. An adversary has to overcome three challenges before they can launch a reverse-engineering attack. First, they have to isolate the eFPGA fabric from the rest of the IP; this is possible since the regular structure of the fabric stands out from the rest of the design (as seen in Fig. 5). Second, for oracle access, they should be able to control and observe the inputs and outputs of the fabric and all the flip-flops. As a worst-case assessment, we endow the adversary with these capabilities although there are orthogonal efforts to mitigate this attack model [10]. Third, the attacker cannot trivially extract the FPGA bitstream [6]. Physical attacks (e.g., optical probing [33]) are out of scope.

B. Why do FPGAs appear to be SAT-attack resilient?

Using this threat model and assumptions, the first hurdle for an adversary is to formulate the design and miter circuit inputs to a SAT solver [11], treating the configuration bitstream as the "key". SAT solvers fail in the presence of combinational loops (cyclic designs) [20], producing unstable results or repeatedly returning distinguishing input patterns. These loops emerge from the re-configurable routing network of the eFPGA. The sequence of re-configurable logic represented by the chain of LUTs/CLBs interconnected by this re-configurable network adds a polynomial complexity to the SAT formulation.

To launch a SAT attack on designs with (potential) loops, like eFPGAs, we need to pre-process the netlist to add constraints to break the loops. Multiple approaches have been proposed to modify the SAT attack for cyclic designs. CycSAT [20] and BeSAT [19] are two approaches. However, eFPGAs have hard combinational loops what CycSAT cannot resolve. These hard loops are intertwined; even when CycSAT breaks a loop to make the circuit acyclic, atleast one loop remains. The acyclic constraints generated by CycSAT overlook such loops and live-locks the solver into repeating the same distinguishing input patterns (DIPs). Be-SAT can break such loops by pruning the keys leading to live-lock DIPs. However, it has exponential complexity in key-size. IcySAT [18] is a practical loop-breaking alternative that finds a subset of feedback nets that when "removed" make the netlist acyclic. The circuit is then unrolled with respect to these feedback nets, with an unroll factor equalling the size of the feedback set. The unrolled circuit can feed into any SAT attack tool.

TABLE I
The amount of unrolling and the number of clauses when preparing the eFPGA fabric for the SAT-based attack

Fabric	Unroll Factor	# Clauses (millions)
3×3	190	6
4×4	628	67
5×5	1441	324

C. Security Evaluation Setup

To formulate a SAT attack to recover the eFPGA bitstream, the eFPGA netlist has to be redefined as a key-controlled netlist, with the configuration bitstream being the key. In an eFPGA, a bitstream is loaded into the configuration flip-flops (FFs) as a sequence of configuration bits. The configuration FFs are interconnected as a scan-chain driven by a programming clock ($prog\_clk$). In our attack setup, we write a script to expose the configuration FFs as primary key inputs by traversing along the scan chain.

To identify the configuration scan chain, we do a depth-first search (DFS) starting from the $scan\_in\_head$ port, until we reach the $scan\_in\_tail$. All FFs in the traversal path driven by the programming clock ($prog\_clk$) store the configuration bitstream. The order in which the configuration FFs are detected along the path corresponds to the bitstream order. The detected configuration FFs are exposed as primary key inputs to convert the eFPGA netlist into a typical SAT attack netlist. This key-exposed netlist is fed to our implementation of IcySAT [18] to unroll the hard loops. To model an oracle, we use the same netlist, but add constraints to set the key-bits to the configuration values from the bitstream generated in the OpenFPGA flow. The unrolled netlist and the oracle netlist are used with KC2 attack tool [21] in our experiments.

D. Impact of Fabric Size

To better understand the impact of fabric size, we synthesized square eFPGAs fabrics of various configurations, based on CLBs with eight 4-LUTs (more details in §V-C) surrounded by I/O tiles, and converted them into unrolled netlists (as described in the previous section). As attack difficulty correlates with how the circuit is modeled for the SAT attack, we categorize eFPGA fabrics in terms of number of feedback nodes to be broken (Unroll Factor) and the clause size of eFPGA netlist, shown in Table I. The size of IcySAT unrolled netlist equals the product of unroll-factor and the number of clauses required to represent the original circuit—both factors

TABLE II
RESULTS FROM ATTACKING DIFFERENT PARTS OF THE BISTREAM

Bitstream	Clauses	Variables	Time	Key-size
I/O bits	6197406	2436085	68.7	12
Routing Bits	5951166	2313613	105.96	336
Logic Bits	6043126	2359351	67.4	215

contribute to the clauses added per iteration of the SAT attack and contributes to attack complexity.

Table III shows the result of attack on different fabric sizes. FPGA fabrics mapped with multiple designs shown in Table V was subjected to SAT attack. It was observed that the complexity of the attack increases exponentially as we increase eFPGA fabric size. Our attack of the 3×3 fabric was successful, and was completed on average in 534 s. We tried to launch similar attacks on the 4×4 and 5×5 fabrics, but these were not able to complete within 48 hours, which suggests that at least a 4×4 fabric should be selected, as a minimum, for redaction. In attempting to launch these attacks, we needed to increase the amount of RAM available to KC2 (we doubled the allocation in the HPC system each time); the attack on 4×4 fabric only stopped crashing with 128 GiB of RAM, while the attack on the 5×5 fabric required 512 GiB. To see if the attack time of a fabric is affected by the design it implements, we tried to attack three different designs in the 3×3 fabric. No significant differences were observed in attack time. This suggests that, for a fixed fabric, the attack complexity might be independent of the bitstream. In future, we will extend this work to validate the observation with results on larger fabrics.

E. Analysis of Different Parts of the Bitstream

Bits in the eFPGA bitstream can be categorized depending on the part of the eFPGA they configure. We identified three main parts *routing configurations bits* comprising Switch Block and Connection Block configuration bits (i.e., the logic within tiles for routing signals), *logic configuration bits* comprising the configuration bits for the CLBs, and the *I/O configuration bits*, which configures the I/O ports to be input or output in the eFPGA. We attempted a "piece-wise" analysis of the challenge in recovering the different configuration bits, assuming all others are known, for insights into which elements of an eFPGA might be harder/easier to recover. This can inform redaction-centric eFPGA fabric design in future.

We launched a partial SAT attack on the 3×3 fabric to recover a particular class of configuration bits while assuming the others are unknown. Table II shows the recovery time for different components of the bitstream while assuming other bits are known. We observed that routing and logic bits have similar complexity in terms of attack run-time required per bit of bitstream. This is anticipated as CLBs and routing units constitute MUX-trees with configuration bits either controlling select inputs of MUXes or acting as a data input of a MUX and hence represent a similar logic at gate-level of abstraction.

However attacking I/O bits resulted in a larger attack run-time per bit. We intuit that this arises from the low output corruptibility resulting from the I/O bits. For partial SAT attack, we assume routing and CLB bits are known; this configures the inputs being used by the eFPGA. For example, assume an eFPGA configuration that uses 2 out of 10 possible inputs. Since the eFPGA logic is configured to use the 2 inputs, changing the don't-care inputs does not lead to a DIP, as required by the SAT attack. The SAT solver has to search for the correct I/O bit configuration such that the two care inputs can be used to find a DIP. This limits DIP equivalence class, increasing solver time.

F. Exploring the Impact of Partial Bitstream Recovery

In this section, we explore the security compromise when an attacker can recover a part of bitstream through a side-channel, replicating and extending the study in prior related work [4]. For instance, since the ASIC/eFPGA interface could be identified from the netlist, the attacker might be able to guess the I/O configuration bits, thus reducing the key search space. To explore the security under such a scenario, we explore how the attack time varies with number of unknown key-bits. Fig. 4 shows how the attack-time varies with number of unknown key-bits.

Counter to intuition that attack run-time is proportional to the number of unknown configuration bits, we found that when a small subset of configuration bits are known, the attack-time is greater when compared to the attack-time when all key-bits are unknown. To confirm that this is not a consequence of the random nature of configuration bits chosen to be key-bits, we run two trials choosing different random set of configuration bits. The information on known key-bits is added to the circuit formulation as constraints. Consequently, the effective number of variables remains the same. When there is a substantial information on the key variables, it reduces the search space of the Davis-Putnam-Logemann-Loveland (DPLL) algorithm [34] within the solver, reducing the solver effort. However, when a small subset of keys are known, this information may burden the solver, without significantly pruning the search space. This may unpredictably increase or decrease the run time as suggested by this experiment. We conclude that the attacker needs to recover a substantial number of key-bits to invalidate the SAT resiliency claim. To verify this argument from a practical perspective, we launched an attack assuming that the attacker was successful in recovering all I/O configuration bits (12/563 bits) in a fabric by snooping at the ASIC-eFPGA interface and launched an attack with 551 unknown keys and 12 known I/O configuration bits. The partial SAT attack took about 1045 seconds when compared to 545 seconds for the full SAT attack.

V. EXPERIMENTAL EXPLORATION

A. Experimental Overview

To explore practical issues and feasibility of eFPGA-based redaction, we consider, as a case study, a set of open-source IPs that comprise hand-crafted modules. These IPs reflect various application domains and feature in logic locking/obfuscation literature. For each IP we examine the modules and their

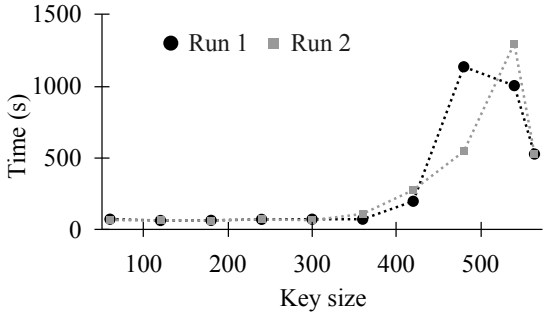


Fig. 4. Time dependency on key-size for partial SAT Attack for two random configuration bits chosen to be key-bits, for a 3×3 eFPGA fabric.

TABLE III
RESULTS OF SAT ATTACK ON DIFFERENT FABRIC NETLISTS

Fabric size	Circuit	Unroll factor	Bit-stream	# Clauses	Time (s)	# Variables
3×3	2-bit adder	190	563	5775606	543.5	226668
3×3	1-bit adder	190	563	5775606	529.3	226668
3×3	2-bit Multiplier	190	563	5775606	527.6	226668
4×4	memory write	628	1904	_	TO	_
5×5	zero comparator	1441	4184	_	TO	_

characteristics. We select several modules to put them through the OpenFPGA flow to identify the fabric size required to redact the module, synthesize the fabric to produce an eFPGA macro, Fig. 5a, and then put the combined IP through the physical design flow Fig. 5b. For this study, we target the FreePDK 45 nm technology library [35]. We do synthesis using Cadence Genus 18.1 and use Cadence Innovus 18.10 for physical design. Synthesis was performed on a server with AMD EPYC 7551 (32 Core, 512 GB RAM).

B. Case Study IPs

For this study, we redact a variety of RTL IPs to gain a broader sense of the implications of eFPGA-based redaction. Table IV shows the IPs, the number of unique RTL modules, and the ranges of input/output bits in the modules. The IPs include small designs, like GCD from the OpenRoad project [36], to larger designs, like GPS from the MIT Lincoln Labs Common Evaluation Platform [37]. These IPs perform arithmetic and cryptographic operations and appear as targets for obfuscation in prior work [1]. Given the number of modules in each IP at the register transfer level, the designer has numerous options for redaction. For each IP, we select a module for redaction, as shown in Table V.

TABLE IV
CHARACTERISTICS OF THE CASE STUDY IPS

IP	# Modules	Module Interfaces (Range in Bits)	
		Input Bits	Output Bits
AES	9	10–128	8–128
GCD	8	8–45	1–18
GPS	12	6–128	1–256
PicoSoC	10	8–96	4–86
12-bit Adder	1	24	13
2-bit adder	1	4	3

To examine the redaction in the context of an SoC, we use PicoSoC [38], which includes the PicoRV32, a size-optimized RISC-V CPU [38]. As the designer has freedom to chose what to redact, we redact a portion of the design that affects the CPU function—for our experiments, we redact the logic that signals whenever the memory is ready.

For AES, from CEP [37], we redact two modules: the module which generates the valid_out signal (which indicates that the encryption is done and the output is ready to be read) and the rconst value. In AES encryption, we need to generate the key for each round (key_expansion); this function requires the rconst value for each round. This rconst value can be read from the fabric instead of hard-wiring. These modules are used independently in the AES module. From the CEP, we also protect GPS IP; we redact the "C/A Code Civilian Acquisition or Access Code" (CACODE) module. Additionally, to understand our study not only on larger IPs but some general purpose IPs, we redact a 2-bit and 12-bit adder from the GCD IP from OpenRoad project [36]. For the GCD IP, we redact logic that subtracts/compares data to zero.

C. Using OpenFPGA for eFPGA Generation

The OpenFPGA flow allows a designer to specify various FPGA architectures, for instance, by choosing different CLB designs. We selected a simple FPGA architecture that has appeared in prior work [14], [26], [27], [31] that comprises CLBs built with eight 4-input LUTs, which we specify in the .xml architecture file for OpenFPGA (as shown in Fig. 2).

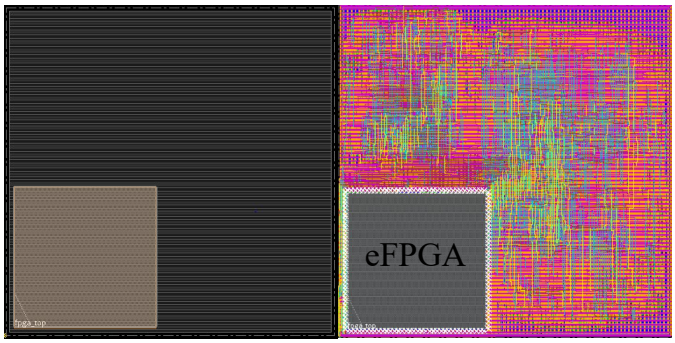
To produce the eFPGA, we replace the "out-of-the-box" I/Os pads with simpler input and output pins since the fabric is embedded in an ASIC; in our redactions, the fabric does not need to be able to communicate with the rest of the SoC or off-chip. This simplification reduces the area overhead tremendously (compared to a non-embedded FPGA fabric). For connections within the fabric we use 2-input multiplexers whose select line is controlled by the configuration bitstream. An added benefit of the OpenFPGA flow [14] is that it automatically generates testbenches for functional verification as well as the required SDC files to disable combinational loops during synthesis of the fabric. The eFPGA fabric is synthesized, placed, and routed separately from the rest of the IP, to produce an eFPGA macro.

D. Redaction Results

For each IP, we redact a module by replacing its instantiation in the original IP's RTL with the fabric macro generated using OpenFPGA, after simulating the eFPGA fabric to ensure that the redacted functionality is implemented correctly. The required fabric size for implementing the redacted module ranges from 4×4 to 8×8. The overall design is then synthesized, placed, and routed as shown in Fig. 5b. For comparison, we also synthesized, placed, and routed each IP without redac-

TABLE V
CHARACTERISTICS OF REDACTING VARIOUS IPS

IP	Module	ASIC-only			OpenFPGA				Redacted IP Area (μm^2)	
		Critical Path?	Module area in ASIC	Area (μm^2)	Fabric	I/O utilization (%)	Resource use (%)	Bitstream	eFPGA Portion	Total
GPS	cacode	No	296.1	273806.0	6×6	21	81	9237	102312.1	570964.5
GCD	zero_comparator	No	7.1	403.2	5×5	71	25	4184	45872.5	68531.4
	mux	No	29.8	403.2	8x8	68	6	16341	185230.4	264534.5
	subtractor	No	74.5	403.2	8x8	80	20	16341	185230.4	264614.3
AES	valid_output/rconst	No	84.4	283944.9	6×6	67	89	9237	102312.1	562648.1
PicoSoC	memory_write	No	259.0	82705.8	4×4	75	100	1954	21182.4	138115.6
ADDER	12-bit Adder	No	227.8	227.8	7x7	71	19	11164	128535.2	128535.2
	2-bit adder	No	12.1	12.1	3×3	58	100	564	6207.1	6207.1



(a) (b)

Fig. 5. (a) 4×4 eFPGA fabric in the bottom-left corner of the floor plan before placing the rest of the IP. (b) Complete PicoSoc design with the eFPGA fabric and remaining logic.

TABLE VI
COMPARISON OF AREA, POWER AND DELAY OVERHEAD WITH INTEGRATION OF DIFFERENT FABRIC SIZES IN PICOSOC

Design	Area (μm^2)	Power (mW)	Delay (ns)
PicoSoC	108307	30.0	1.284
PicoSoC + 3×3	116316	40.2	1.878
PicoSoC + 4×4	137330	49.3	2.261
PicoSoC + 5×5	173725	56.3	3.860
PicoSoC + 6×6	259508	68.7	4.708

tion. The post-synthesis area results are shown in Table V, from reports produced by Cadence Innovus.

1) Area: Table V shows that there is a significant amount of area overhead associated with the redaction method in general. This places a burden on the designer to properly select the best module(s) to achieve their security level with a reasonable overhead. Depending upon the size of the original design, the impact can be relatively characterised as practically possible or not feasible at the allocated budget.

To better understand the area overhead, we take the PicoSoC IP and randomly pick a module to redact such that the fabric required will have different sizes. The result of this exploration is shown in Table VI. Area increases in the range from 10% to 140% suggesting that redacting a function and then integrating it with ASIC is not a simple addition of two IPs. It requires different placement, floorplan, and routing of the design due to the constraints added by the addition of eFPGA and its timing requirement. Thus the area increases as a non-linear function of fabric sizes (Fig. 6).

Moreover, during our experiments, we found that some modules require larger fabrics due to the number of input and outputs of that module. This forces us to increase the size of the fabric and impacts the resource utilization in the fabric. Table V points out this issue; consider the 12-bit Adder, where due to limited number of I/Os the fabric needs to be expanded up to 7×7, resulting in only 19% CLB usage. In other cases, utilisation is better—redaction of AES [37] module, suggests more efficient fabric utilisation (67% I/O and 89% CLBs) which is quite acceptable from the designer perspective.

2) Power: Table VI shows the variation in the power consumption as FPGA fabric varies, as reported by Cadence Innovus. When we compare the power consumption of the module to be redacted in ASIC implementation with the same function mapped to an eFPGA fabric; there is an increase in the expected power consumption because of the extra switching gates and routing multiplexers to connect the fabric, resulting in 30%-130% increases over the original design. In terms of power consumption, there is an inevitably large

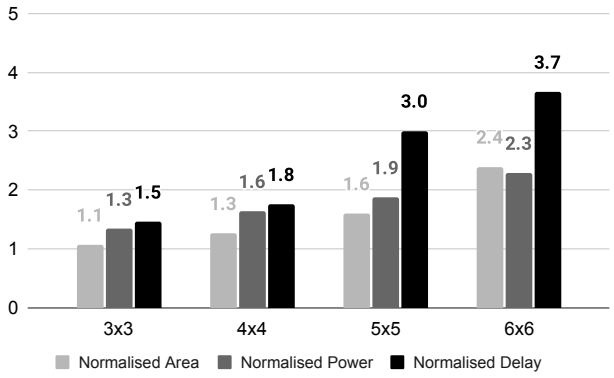


Fig. 6. Comparison of area, delay and power overhead of integration of different fabric size with the original design in PicoSoC. Area normalized to 0.11mm^2, Power normalized to 30 mW and Delay normalized to 1.284 ns

penalty, even for small fabrics.

3) Delay: For better understanding the impact of fabric sizes on delay of the overall IP, we perform a similar experiment as for area Table VI. Delay has a more prominent impact compared to the impact on area and power, i.e., 50% to 270% increase. This is apparent from the FPGA architecture, where we need more routing channels to connect tiles to every other tile in a fabric. This contributes to the additional delay. Fig. 6 illustrates a comparison of normalised delay with for the different fabric sizes.

VI. Discussion and Future Outlook

In this work, we explored the feasibility and other practical issues of eFPGA-based redaction. We performed a security analysis to investigate the characteristics of eFPGAs that contribute towards its SAT attack resilience, and characterized the impact of redacting different modules in a variety of IPs at the register transfer level.

As discussed in Section V-A, we explored the redaction of a variety of IPs for insights into the practicality of eFPGA-based redaction. Generally, for bigger designs like GPS, AES, and PicoSoC, the overhead introduced by integrating the eFPGA is feasible. However, for smaller IPs, the approach is not feasible; for instance, GCD's total design area is smaller than the smallest available fabric (shown in Table V, resulting in a drastic increase in area. Thus, framing eFPGA-based redaction as a general IP-level protection mechanism might not be practical, despite the high-level SAT-attack resilience.

With eFPGA-based redaction, we are "overallocating" resources for redaction (and increasing an attacker's uncertainty). In future, one may explore feasibility of eFPGA redaction for multiple IPs at the register transfer level, i.e., where modules from different IPs share the same redaction fabric. Clearly, there is a complex interplay between fabric size and interface width, fabric utilization by the redacted module, and impact on ASIC quality-of-results, which entails the need for an automated approach to assist with redaction decisions.

In some respects, our case study emulates a "designer-directed" or module-driven approach, in that the module to be redacted is first selected to decide the fabric. During our experiments, we observed that even though a module is not in a critical path for a design, but after integrating it as an eFPGA with rest of the design, it can dominate the timing. The full design flow without any prior understanding of the impact can be very time-consuming—for example, it requires us $\sim$8hr for PicoSoC with 6$\times$6 fabric—this suggests that we need a good way to predict the downstream impact of redaction decisions.

Our security analysis (Section IV-D) suggests that the attack time depends on the fabric and does not depend on the component redacted, at least in the context of our threat model. This suggests that a wider variety of heuristics could be considered in deciding what to redact. For example, if a designer is redacting with the intent to "corrupt" the output for an unauthorized user without the correct bitstream, identifying what to redact based on the "highest value" portion of the design (as suggested by Chen *et al.* [5]) could instead focus on the part of the design with the most impact on the outputs (e.g., identified perhaps through fault analysis). Exploring these alternative approaches to redaction is our future work.

Finally, our study focused on a single eFPGA architecture and we expect the findings to apply for other eFPGA implementations. However, it is possible to vary the complexity of the fabric in terms of blocks and routing, which affects the area, timing, and power characteristics. Our preliminary results in Section IV-E point to the fact that different parts of the eFPGA bitstream potentially impact the attack difficulty in different ways—in future, we will explore the possibility of tailoring eFPGA architectures for redaction.

Resources

Data for the study in this paper can be found at [39].

References

[1] C. Pilato, A. B. Chowdhury, D. Sciuto, S. Garg, and R. Karri, "ASSURE: RTL Locking Against an Untrusted Foundry," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, pp. 1–13, 2021.

[2] B. Tan *et al.*, "Benchmarking at the frontier of hardware security: Lessons from logic locking," *CoRR*, vol. abs/2006.06806, 2020.

[3] 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 Transactions on Design Automation of Electronic Systems*, vol. 24, no. 6, pp. 65:1–65:36, Sep. 2019.

[4] P. Mohan, O. Atli, J. Sweeney, O. Kibar, L. Pileggi, and K. Mai, "Hardware Redaction via Designer-Directed Fine-Grained eFPGA Insertion," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*. Virtual: IEEE, 2021, p. 6.

[5] 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 *2020 57th ACM/IEEE Design Automation Conference (DAC)*. San Francisco, CA, USA: IEEE, Jul. 2020, pp. 1–6.

[6] B. Hu *et al.*, "Functional Obfuscation of Hardware Accelerators through Selective Partial Design Extraction onto an Embedded FPGA," in *Proceedings of the 2019 on Great Lakes Symposium on VLSI*. Tysons Corner VA USA: ACM, May 2019, pp. 171–176.

[7] H. M. Kamali, K. Z. Azar, H. Homayoun, and A. Sasan, "InterLock: an intercorrelated logic and routing locking," in *Proceedings of the 39th International Conference on Computer-Aided Design*. Virtual Event USA: ACM, Nov. 2020, pp. 1–9.

[8] G. Kolhe *et al.*, "Security and Complexity Analysis of LUT-based Obfuscation: From Blueprint to Reality," in *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. Westminster, CO, USA: IEEE, Nov. 2019, pp. 1–8.

[9] B. Liu and B. Wang, "Embedded reconfigurable logic for ASIC design obfuscation against supply chain attacks," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2014*. Dresden, Germany: IEEE, 2014, pp. 1–6.

[10] 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*, pp. 1–1, 2020.

[11] P. Subramanyan, S. Ray, and S. Malik, "Evaluating the security of logic encryption algorithms," in *2015 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, May 2015, pp. 137–143, iSSN: null.

[12] J. Chen and B. C. Schafer, "Area Efficient Functional Locking through Coarse Grained Runtime Reconfigurable Architectures," in *Proceedings of the 26th Asia and South Pacific Design Automation Conference*. Tokyo Japan: ACM, Jan. 2021, pp. 542–547.

[13] M. M. Shihab *et al.*, "Design Obfuscation through Selective Post-Fabrication Transistor-Level Programming," in *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. Florence, Italy: IEEE, Mar. 2019, pp. 528–533.

[14] 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.

[15] J. A. Roy, F. Koushanfar, and I. L. Markov, "EPIC: Ending Piracy of Integrated Circuits," in *Proceedings of the Conference on Design, Automation and Test in Europe*. New York, NY, USA: ACM, 2008, pp. 1069–1074.

[16] J. Rajendran, Y. Pino, O. Sinanoglu, and R. Karri, "Security analysis of logic obfuscation," in *Proceedings of the 49th Annual Design Automation Conference*, ser. DAC '12. San Francisco, California: Association for Computing Machinery, Jun. 2012, pp. 83–89.

[17] ——, "Logic encryption: A fault analysis perspective," in *2012 Design, Automation Test in Europe Conference Exhibition (DATE)*, Mar. 2012, pp. 953–958.

[18] 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.

[19] Y. Shen, Y. Li, A. Rezaei, S. Kong, D. Dlott, and H. Zhou, "BeSAT: behavioral SAT-based attack on cyclic logic encryption," in *Proceedings of the 24th Asia and South Pacific Design Automation Conference*. Tokyo Japan: ACM, Jan. 2019, pp. 657–662.

[20] 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.

[21] K. Shamsi, M. Li, D. Z. Pan, and Y. Jin, "KC2: Key-Condition Crunching for Fast Sequential Circuit Deobfuscation," in *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. Florence, Italy: IEEE, Mar. 2019, pp. 534–539.

[22] L. Li and A. Orailoglu, "Piercing Logic Locking Keys through Redundancy Identification," in *2019 Design, Automation Test in Europe Conference Exhibition (DATE)*, Mar. 2019, pp. 540–545, iSSN: 1530-1591.

[23] Z. Han, M. Yasin, and J. J. Rajendran, "Does logic locking work with EDA tools?" in *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, Aug. 2021, available: https://www.usenix.org/conference/usenixsecurity21/presentation/han-zhaokun.

[24] "Intel Xeon+FPGA Platform for the Data Center," https://reconfigurablecomputing4themasses.net/files/2.2%20PK.pdf.

[25] P. D. Schiavone *et al.*, "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.

[26] 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.

[27] 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.

[28] 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.

[29] 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.

[30] C. Wolf, J. Glaser, and J. Kepler, "Yosys-a free verilog synthesis suite," in *Proceedings of Austrochip*, 2013.

[31] K. E. Murray *et al.*, "VTR 8: High-performance CAD and Customizable FPGA Architecture Modelling," *ACM Transactions on Reconfigurable Technology and Systems*, vol. 13, no. 2, pp. 1–55, Jun. 2020.

[32] G. Gore, X. Tang, and P.-E. Gaillardon, "A Scalable and Robust Hierarchical Floorplanning to Enable 24-hour Prototyping for 100k-LUT FPGAs," in *Proceedings of the 2021 International Symposium on Physical Design*. Virtual Event USA: ACM, Mar. 2021, pp. 135–142.

[33] M. T. Rahman, S. Tajik, M. S. Rahman, M. Tehranipoor, and N. Asadizanjani, "The Key is Left under the Mat: On the Inappropriate Security Assumption of Logic Locking Schemes," in *2020 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. San Jose, CA, USA: IEEE, Dec. 2020, pp. 262–272.

[34] P. Baumgartner, "A first-order logic davis-putnam-logemann-loveland procedure," 08 2001.

[35] "FreePDK45:Contents - NCSU EDA Wiki," https://www.eda.ncsu.edu/wiki/FreePDK45:Contents.

[36] T. Ajayi *et al.*, "Toward an Open-Source Digital Flow: First Learnings from the OpenROAD Project," in *Proceedings of the 56th Annual Design Automation Conference 2019*. Las Vegas NV USA: ACM, Jun. 2019, pp. 1–4.

[37] MIT Lincoln Labs. (2020) Common evaluation platform. https://github.com/mit-ll/CEP/tree/master/hdl_cores/gps.

[38] Picorv32. https://github.com/cliffordwolf/picorv32.

[39] https://github.com/NYU-Hardware-Security/FPGA-Fabrics-for-Redaction-ICCAD.