Hardware Redaction via Designer-Directed Fine-Grained eFPGA Insertion

Prashanth Mohan, Oguz Atli, Joseph Sweeney, Onur Kibar, Larry Pileggi and Ken Mai
Department of Electrical and Computer Engineering, Carnegie Mellon University
Pittsburgh, PA, USA
{*pmohan, aatli, jsweene1, okibar, pileggi, kenmai*}*@andrew.cmu.edu*

***Abstract*—In recent years, IC reverse engineering and IC fabrication supply chain security have grown to become significant economic and security threats for designers, system integrators, and end customers. Many of the existing logic locking and obfuscation techniques have shown to be vulnerable to attack once the attacker has access to the design netlist either through reverse engineering or through an untrusted fabrication facility. We introduce soft embedded FPGA redaction, a hardware obfuscation approach that allows the designer substitute security-critical IP blocks within a design with a synthesizable eFPGA fabric. This method fully conceals the logic and the routing of the critical IP and is compatible with standard ASIC flows for easy integration and process portability. To demonstrate eFPGA redaction, we obfuscate a RISC-V control path and a GPS P-code generator. We also show that the modified netlists are resilient to SAT attacks with moderate VLSI overheads. The secure RISC-V design has 1.89x area and 2.36x delay overhead while the GPS design has 1.39x area and negligible delay overhead when implemented on an industrial 22nm FinFET CMOS process.**

***Index Terms*—hardware security; design obfuscation; field-programmable gate array; FPGA**

I. Introduction

Due to the rapidly increasing cost of building and maintaining silicon manufacturing facilities with technology scaling, there has been a steady decline in the number of IC fabrication facilities offering state-of-the-art process technologies. As a result, both companies and governments have been forced to rely on untrusted third-party silicon fabs. Reliance on untrusted fabrication reduces the control over the security of chips and introduces a number of threats such as reverse engineering, IP theft, overproduction, trojans, and counterfeiting [1].

There has been significant effort in developing new techniques to design hardware that is resistant to such IP threats. Hardware obfuscation methods such as logic locking, gate camouflaging, split-manufacturing, and reconfigurable logic are a few of the proposed solutions [2]. While logic locking has gained attention as a possible low overhead obfuscation solution, many of the existing logic locking schemes have been broken by satisfiability-based (SAT) attacks [3]. Additionally, many SAT-resilient locking methods suffer from low output corruptibility when the wrong key is applied to unlock the circuit [4].

We introduce an alternative approach based on using reconfigurable logic called hardware redaction. In the intelligence community, it is common practice to redact sensitive information from documents before release to untrusted parties. This includes obfuscation of entire sections, paragraphs, sentences, or even single words/numbers. This allows fine grained obfuscation of critical information from public release. We endeavor to employ analogous capability to hardware designers enabling them to obfuscate selected portions of a hardware design as shown in Fig. 1. This method differs significantly from other types of logic locking/obfuscation where an entire block is obfuscated and a tool chooses which specific gates to camouflage/obfuscate. Those techniques have been shown to be vulnerable to a variety of attacks using SAT-solvers and ATPG-based methods, because there is still information leakage from the un-camouflaged/locked gates. In the proposed method, only the critical IP is fully obfuscated by replacing it with an eFPGA, entirely concealing the logic and interconnect from reverse engineers and the untrusted fabrication facility. Additionally, by not altering the non-critical IP, the overall VLSI overheads (e.g., area, power, and performance) can be kept to a minimum.

In Section II, we introduce hardware obfuscation and discuss related work on reconfigurable logic based hardware obfuscation techniques. In Section III, we introduce soft eFPGA based redaction and the tool flow for generating and integrating soft eFPGA fabrics with ASIC designs. In Section IV, we apply our proposed soft eFPGA redaction flow and obfuscate RISC-V control path and the GPS P-code generator. We provide post-layout experimental results on a 22nm FinFET CMOS process.

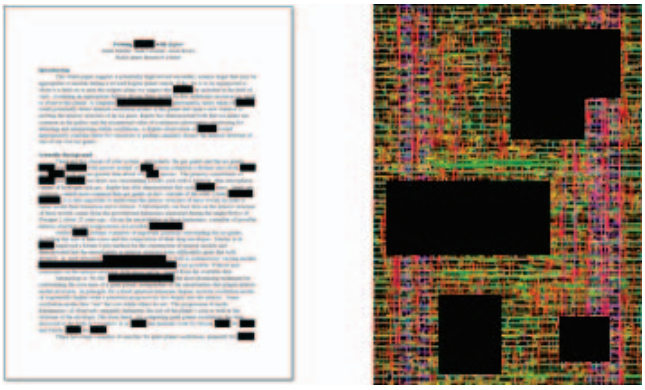


Fig. 1: Hardware redaction concept. Analogous to text document redaction, but in hardware, sections are removed and replaced with synthesizable eFPGA.

This work was supported in part by DARPA under contract FA8750-17-1-0059 "Obfuscated Manufacturing for GPS (OMG)" and also by Sandia National Laboratories.

In Section V, we show that the proposed eFPGA redaction is SAT resilient by performing SAT attack experiments on redacted netlists.

II. Background

Hardware obfuscation techniques aim to hide the design intent while maintaining correct circuit behavior by concealing the structure and the functionality of the circuit from adversaries. A number of obfuscation techniques such as split manufacturing, gate camouflaging, logic locking, and reconfigurable logic have been proposed in literature [2]. Split manufacturing schemes aim to reduce the design information available to the untrusted facility by manufacturing some of the metal layers in a trusted facility. This can protect the design from untrusted fabrication but it does not offer protection against reverse engineering after product deployment. Gate camouflaging on the other hand, makes reverse engineering challenging by using layout and circuit level techniques that make it harder to distinguish different logic gates from each other, but does not offer protection against an untrusted foundry. Logic locking has also been used as a hardware obfuscation scheme that offers IP protection both against untrusted fabrication and reverse engineering. Logic locking techniques introduce an additional key input to lock the design by modifying the input/output relationship of the design. The end user needs the correct key to restore the correct circuit functionality.

Several proposals have been put forward in literature to use reconfigurable logic for hardware obfuscation. Baumgarten et. al. [1] use reconfigurability to insert a logical barrier between the primary inputs and primary outputs of a combinational circuit. While this paper proposes using Look Up Tables (LUTs) to replace logic gates for obfuscation, it does not discuss the VLSI overheads of adding reconfiguration nor quantifies the security against an attack. Later, Kamali et. al. [5] prove that reconfigurable barriers [1] are weak against SAT attacks and propose LUT-Lock. LUT-lock modifies the gate to LUT replacement strategy to improve SAT resilience but at the cost of reduction in output corruptibility. Even though LUT-Lock provides resilience against the SAT attack, low output corruptibility may result in correct input-output relationship for large fraction of the input space even when an incorrect key is applied [4]. The work by Kolhe et. al. [6] explores custom LUT structures and analyzes how VLSI overheads trade off with SAT resilience. They show that using smaller LUTs with additional muxes to obfuscate the LUT inputs can provide similar security compared to using larger LUTs but with much less area overhead. This can also be interpreted as augmenting LUT obfuscation with interconnect obfuscation which is similar to the structure of an FPGA fabric.

FPGAs are the most widely used reconfigurable circuits due to their versatility and the wide range of logic functions that they can perform. Structurally, FPGAs consist of a 2D array of complex logic blocks (CLBs) that are interconnected through a routing fabric. The CLBs contain look-up tables (LUTs) that can perform arbitrary logic functions, and the routing fabric contains routing channels and interconnect muxes to connect the CLBs together. The FPGA also contains a large number of configuration bits that are used to program the LUTs and the interconnect muxes. The configuration bits of the FPGA is analogous to the key bits of a logic locked circuit. An embedded FPGA (eFPGA) has the same architectural features as an FPGA, but its main difference is that while FPGAs are stand-alone parts, eFPGAs interact with other IP blocks on the same die, most often within a system-on-chip (SoC) environment. Therefore, eFPGA is a natural candidate for reconfigurable logic based hardware obfuscation.

III. Soft eFPGA Redaction

Though the reconfigurablity of eFPGA makes it a natural candidate for hardware obfuscation, the use of eFPGA fabrics for fine-grained hardware redaction comes with its own set of challenges.

1) Inflexible eFPGA hard macros: Hu et. al [7] explored the use of eFPGA for obfuscation but they assume the availability of a ready-made eFPGA hard macro IP to obfuscate the designs. While eFPGA hard macro IPs can be purchased from commercial vendors such as Achronix, QuickLogic, and FlexLogix, these are not suitable for fine-grained obfuscation because they are typically organized into very large blocks (1K - 4K LUTs) and the end user cannot fine-tune the eFPGA architecture parameters to match the redacted design [8], [9]. To enable fine-grained eFPGA redaction, we need the ability to rapidly and automatically generate eFPGA fabrics of different sizes and architectural parameters that match the redacted portions of the design.

2) Mixed ASIC/FPGA tool flow: There is a lack of CAD tool flows that support combined logic synthesis, timing analysis, and optimization of a mixed ASIC and FPGA design. To address this issue, Shihab et. al [10] proposed the use of a specialized transistor-level programmable fabric along with its own tool-flow for obfuscation. While specialized programmable fabrics can decrease area and delay overheads, designing custom layout in modern process technology requires extensive design time/effort and has poor process portability, which are not compatible with typical SoC design schedules, budgets, and fab diversity requirements. Thus, using "soft" eFPGA based

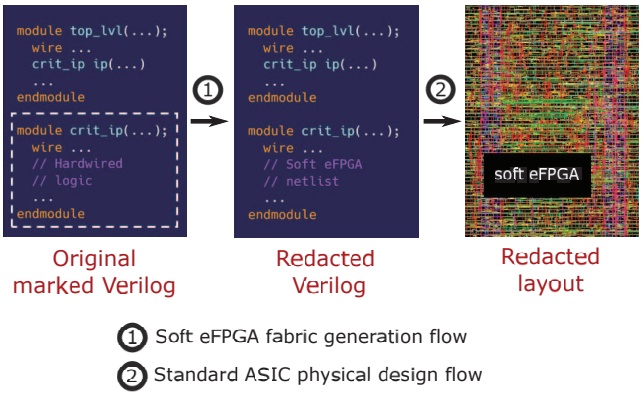


Fig. 2: Overview of the soft eFPGA redaction flow starting from the original marked Verilog to the final redacted layout.

designs that specify the eFPGA fabric in RTL and use the conventional standard cell based CAD tool flow to implement the physical design are more appropriate for many SoC designs.

To address the aforementioned challenges and realize fine-grained hardware redaction, we propose a soft eFPGA based redaction flow. Our soft eFPGA redaction flow uses multiple open-source tools in tandem to generate soft eFPGA fabrics of various sizes with different architectural parameters. In addition to that, our flow can perform logic synthesis, timing analysis, and optimization of the mixed ASIC and soft eFPGA design as a whole without introducing any significant deviations to the standard ASIC physical design flow.

A. CAD tool flow

Soft eFPGA redaction allows hardware designers to obfuscate selected portions of an IP in a fine-grained manner, analogous to how sensitive information in documents is redacted as illustrated in Fig. 1. An overview of the soft eFPGA redaction flow is show in Fig. 2. The designer starts with a Verilog description of the design and marks the critical IP that needs to be redacted. Then, the fabric generation flow automatically determines the smallest eFPGA fabric required to map the redacted module on to an eFPGA. The corresponding eFPGA module is generated using our Chisel based fabric generator and the marked critical IP module is replaced with the eFPGA module to obtain the redacted design. The eFPGA module does not contain any design details of the security critical IP, and the desired functionality can only be achieved by loading the eFPGA with the correct configuration bitstream. Our Chisel based fabric generator also generates Static Timing Analysis (STA) constraints that can be used to synthesize and optimize the eFPGA fabric. The redacted design along with the generated STA constraints are passed on to the standard ASIC physical design flow to generate the redacted layout.

1) Fabric generation flow: Fabric generation flow is responsible for generating the redacted design with the eFPGA fabric, the bitstream required to map the critical IP onto the eFPGA fabric, and the STA constraints required for the physical design tools. The first step in the fabric generation flow is to synthesize the marked critical module into LUTs using the open source Yosys synthesis tool [11]. The synthesized LUT netlist is then passed on to the VTR/VPR tool [12] to place and route the redacted module. During the place and route process, VPR identifies the smallest fabric size and routing channel width required to map the design on to the eFPGA fabric. The VPR output files which include information about the fabric size, channel width, and other fabric architecture parameters are passed on to our fabric generator designed using the Chisel HDL [13]. The Chisel fabric generator outputs the eFPGA Verilog RTL along with the STA timing constraints that are necessary for downstream physical design tools. Bitstream generation is done by processing VPR output files through custom Python scripts. The resulting bitstream is used to verify the functionality of the redacted module using Cocotb testbenches.

2) Mixed ASIC/eFPGA physical design flow: A generic eFPGA IP has to offer satisfactory delay, power, and area metrics for any application mapped on to the fabric. Generic timing optimization is necessary for the paths between the logic blocks so that the logic synthesis is application agnostic. This is not the case for an eFPGA that will be used for redaction, as the design that will be implemented is known prior to its synthesis. This means that the paths within the fabric that will be exercised during operation are also known, and the CAD tools can make use of this information to optimize the fabric further. Although this optimization would minimize the overhead of redaction, the buffering and gate sizing efforts by the tools might make the optimized paths easier to determine, and help the attackers narrow down some of the routing configuration bits. Even if the attacker can narrow down a portion of the routing configuration bits, experimental results in Section V show that the design is still highly resilient to SAT attacks.

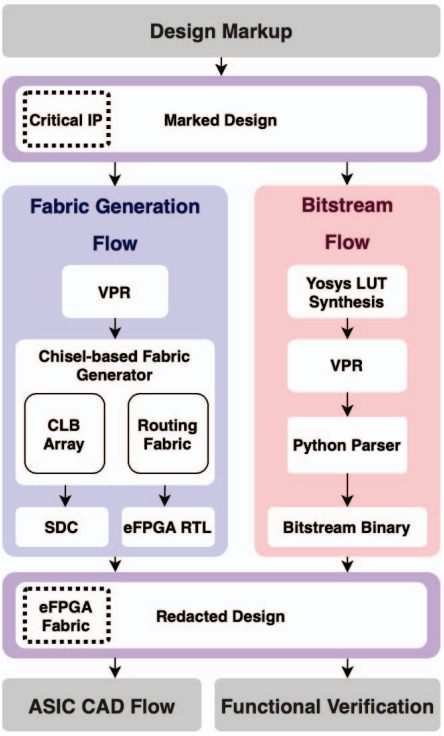


Fig. 3: Detailed design flow for redacted RTL generation and verification. The redacted RTL generation is automated using custom Python scripts and a number of open source tools including Yosys, VTR/VPR, Chisel, and Cocotb.

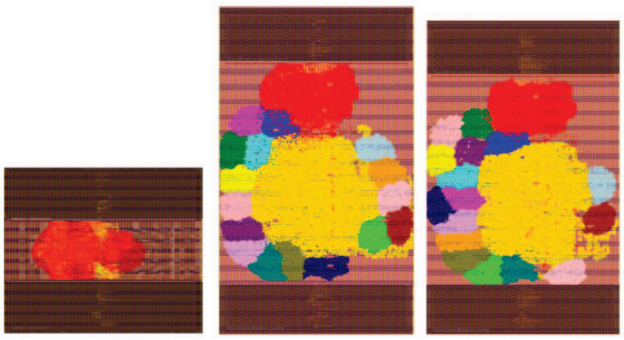
To explore the differences between generic and design-specific eFPGA fabrics, we implemented two different versions of the redacted designs. For the first version, we generate STA constraints to optimize all the channel wires of the fabric in a bitstream-agnostic manner. We call this version *eFPGA_generic*. For the second version, we further optimized the paths that are exercised in the redacted design that will be mapped to the eFPGA by translating the bitstream into additional STA constraints. We call this design-specific version *eFPGA_opt*.

IV. Experimental results

In order to evaluate the overheads and the effectiveness of eFPGA redaction, we obfuscated two different designs: a RISC-V CPU core and a GPS P-code generator. We mapped the security-critical modules in the designs to *eFPGA_opt* and *eFPGA_generic* fabrics and implemented the designs on an industrial 22nm FinFET CMOS process using the foundry supplied standard cells libraries. The VLSI metrics are derived from post-layout extracted designs using Cadence Innovus. We also perform SAT attacks on the eFPGA fabric and show that the designs are resilient to SAT attacks.

A. RISC-V core

The first design that we obfuscated is a 32-bit RISC-V CPU core compatible with the RV32I instruction set. It is a 5-stage pipelined architecture with single-port 6T-SRAM based 16KB instruction and data memories. A scalar CPU core has many elements that are relatively standard across different designs, such as the adder and the multiplier, but the control logic is unique to the architecture. For this reason, in order to obfuscate the RISC-V core, we redacted the control path using a soft eFPGA fabric. The control path was mapped onto a 4x4 eFPGA fabric with 8 LUT4s and 44 channels.



(a) Original **(b)** Generic eFPGA **(c)** Optimized eFPGA

Fig. 4: (a) Layout of the original RISC-V design. The datapath cells and control path cells are highlighted in red and yellow respectively. Layout of (b) *eFPGA_generic* and (c) *eFPGA_opt* designs. The eFPGA tiles are highlighted with different colors and the routing fabric is highlighted with yellow in (b) & (c).

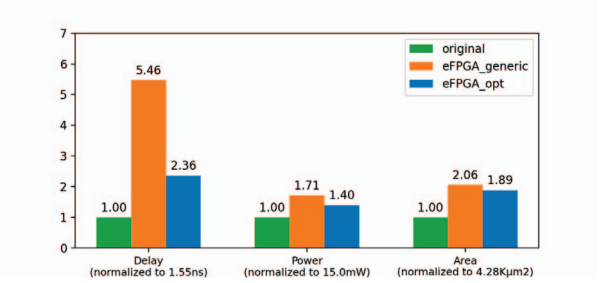


Fig. 5: Delay, power, and area overheads for the original and redacted RISC-V designs with *eFPGA_generic* and *eFPGA_opt* implementations normalized to the original designs from post-layout implementation. The delay overhead of the *eFPGA_opt* version reduces by more than 2X compared to the *eFPGA_generic* version.

Fig. 4 shows the final layout highlighting the tile locations and routing fabric locations for the *eFPGA_generic* and *eFPGA_opt* designs respectively. Configuration storage in the scan chains accounts for approximately half of the area overhead. The control path area shown in yellow in Fig. 4(a) increased by a considerable amount after the redaction, but Fig. 5 shows that the overall area overhead is less than 2x with the *eFPGA_opt* redaction. This overhead factor is larger in RISC-V compared to the GPS P-code generator, because in the RISC-V, the number of redacted cells is a larger percentage of the overall system cell count. In this particular case, the critical path goes through the stall logic, which lies in the control path. This results in delay overhead even in *eFPGA_opt* method, unlike the GPS test case. However, the overhead is less than the *eFPGA_generic* design, due to design-specific optimization.

B. GPS P-code generator

The second design we obfuscated is the GPS code generator found in the MIT Lincoln Labs Common Evaluation Platform (CEP). This is a reference implementation of the circuit that is designed to evaluate different obfuscation methods that can protect a GPS module from untrusted fabrication. Our approach obfuscates the following three security critical attributes of a GPS module as per the CEP documentation:

- Length of P-code generator
- Position of taps for P-code generator
- Initial value used by P-code generator

To conceal this information, we redacted the next state logic of the linear-feedback shift registers (LFSRs) that are used to generate the P-code. The fabric hides the reset initial value and feedback logic function within the configuration bits. We also added the rollover value of the LFSRs on the configuration scan chain, which obfuscates the length of the P-code. A 4x4 eFPGA fabric with 8 LUT4s per CLB was able to implement this desired functionality.

Fig. 6 shows the final layout highlighting the tile locations and routing fabric locations for the *eFPGA_generic* and *eFPGA_opt* designs respectively. Fig. 7 shows that *eFPGA_opt* and *eFPGA_generic* versions are very similar in terms of area, but there is a significant difference in delay. *eFPGA_generic* delay incurs 3.21x delay overhead, while *eFPGA_opt* version does not incur any delay overhead. This is because the critical path of the design is not through the eFPGA, and the *eFPGA_opt* version can optimize the fabric delay below the critical path in the design. Since the *eFPGA_generic* version has no information on which paths will be exercised, the same optimizations can not be performed by the tool, and the generic fabric ends up having higher delay than the other paths in the GPS core.

V. Security analysis

To demonstrate the security efficacy of eFPGA redaction, we performed a series of experiments in which we mounted the SAT attack on the redacted netlists. We performed the attacks

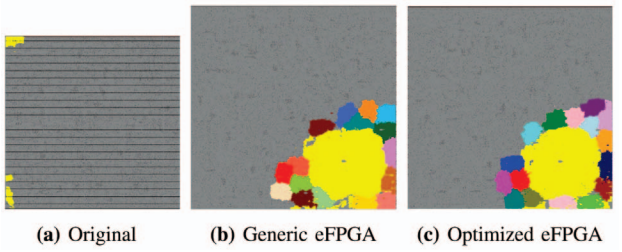


Fig. 6: Layout of the original (a), *eFPGA_generic* (b), and *eFPGA_opt* (c) redacted GPS designs. The P-code generator is highlighted with yellow in the original design. In the redacted designs, the routing fabric is highlighted with yellow and the logic blocks are highlighted in other colors.

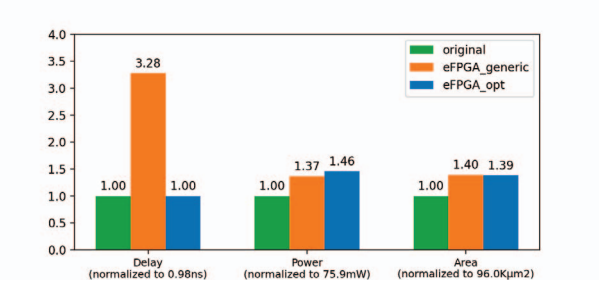


Fig. 7: Delay, power, and area overheads for the original and redacted GPS designs with *eFPGA_generic* and *eFPGA_opt* fabric optimization. *eFPGA_opt* design incurs no delay overhead as the critical path does not pass through the redacted portion of the design.

through the eFPGA IO. We assume that the attacker would have access to the eFPGA IO, because a DFT scan chain is necessary to allow visibility into the faults inside the eFPGA during testing. All SAT attack trials ran with a timeout of 12 hours on a workstation with a 32-core 2.5 GHz AMD EPYC 7502P CPU and 1TB DDR4 memory.

A. SAT Attack setup

We used a miter-based SAT attack to evaluate the attack resilience of eFPGA redaction. This attack uses the redacted netlist and unlocked circuit to iteratively produce input-output (IO) relationships [14] and rule out keys that don't produce the same behavior. If a circuit contains stateful cycles under any input or key combination, miter-based SAT attacks will potentially get stuck in an infinite loop [15]. Since eFPGA fabrics contain numerous cycles due to their flexible interconnect network, the miter-based SAT attacks do not terminate. To enable our experiments, we implemented the CycSAT algorithm [15] that allows the attack to run to completion in cyclic circuits by adding constraints to the attack formulation. These constraints rule out any key that creates cycles in the design.

B. Attack results

We evaluate the attack resistance of eFPGA redaction by mounting the SAT attack on the redacted RISC-V, which contains a total of 8119 configuration bits. The SAT formulation contains approximately 600 million clauses and 250 million variables, most of which are due to acyclic constraints generated by CycSAT. The attack unsurprisingly times out in this setting. In order to better understand how the attack time scales with the number of eFPGA configuration bits, we start by running a modified SAT attack where we retain a fraction of the key bits as variables to be solved by the attack and hard code the rest of the key bits to their correct value. We run multiple experiments by slowly increasing the number of unknown key bits and plotting the corresponding SAT attack run times in Fig. 8. The unknown key bits are chosen randomly, distributed equally between LUT and routing configuration bits. Multiple trials are run for each experiment to account for the variability in SAT attack run times due to random selection. The various points marked in Fig. 8 represent the SAT attack runtime for each of these random trials. We plot the lower bound of the SAT attack runtime for each experiment to visualize the growth of SAT run times with increasing number of key bits.

Another way to interpret Fig. 8 would be to consider the case where an adversary is able to reverse engineer a fraction of the total configuration bits of the eFPGA through side channels or probing methods. The exponential increase in SAT attack runtime shows that the redacted RISC-V design is secure even when the attacker can reverse engineer a sizeable chunk of the configuration key space. All the trials time out when the number of key bits is equal to or greater than 1024 bits, which is 12.6% of the total configuration bits in the 4x4 tile eFPGA used in the example design. We place the timed out datapoints on the 12 hour timeout line in the plot. In reality the time taken to solve the designs can be much greater. As an example, we ran a trial with 1024 key bits without any timeout as an additional experiment and found that it took 3 days to finish which reinforces the exponential growth of the SAT run time.

To address possible concerns with respect to information leakage in *eFPGA_opt*, we ran a set of additional SAT experiments in which the variable key bits are selected from a reduced configuration key space that includes only the routing muxes and LUTs that are used by the RISC-V design mapped

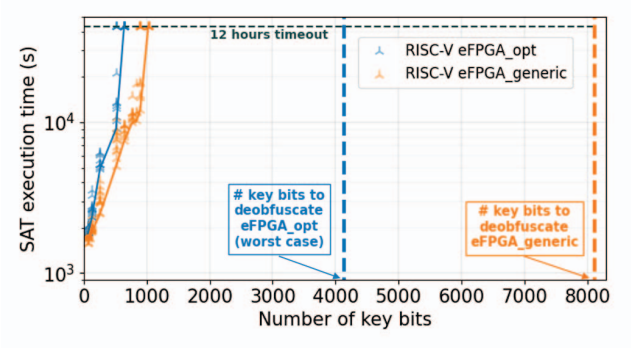


Fig. 8: SAT attack runtimes for redacted RISC-V design mapped onto a 4x4 soft eFPGA fabric increase exponentially with the number of key bits. The blue and orange vertical lines indicate that the SAT attack run times required to deobfuscate the RISC-V design will take multiple orders of magnitude longer than the timeout period of 12 hours.

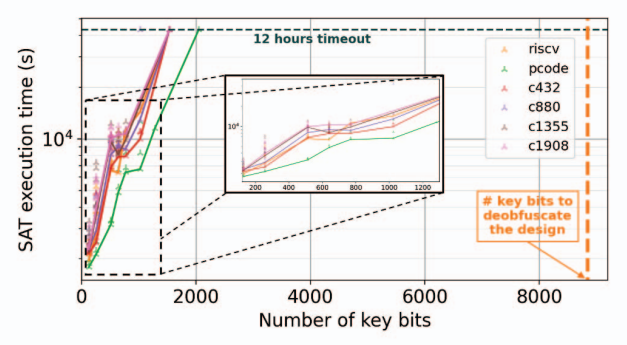


Fig. 9: SAT attack runtime for the RISC-V, P-code generator, and ISCAS'85 benchmarks mapped on to a 4x4 eFPGA fabric. The orange vertical line indicates that the SAT attack run times required to deobfuscate the benchmark designs will take multiple orders of magnitude longer the 12 hour timeout.

onto the eFPGA. In this case, we assume that the attacker is able to extract all potential information from the structure of *eFPGA_opt*. This will require the attacker to identify the timing target, extract the timing of every path inside the eFPGA fabric, and analyze which paths are not optimized to that target. Those paths will consist of the routing muxes and LUTs that are not used by the mapped RISC-V design. However, by reducing the configuration key space to 4133 used configuration bits, we can account for the information leakage in the *eFPGA_opt* design and estimate the worst-case SAT attack runtimes by providing maximum advantage to the attacker. Even under this worst-case conditions, the *eFPGA_opt* design is highly resilient to SAT attacks as the SAT attack runtimes still follow an exponential trend as shown in the blue *eFPGA_opt* plot in Fig.8.

To extend the attack results to other types of circuits, we performed the attack on the GPS P-code generator described in Section IV and a set of ISCAS'85 benchmark circuits (c432, c880, c1355, c1908) that fit on the 4x4 eFPGA fabric that is used in the redacted RISC-V CPU. We performed these attacks on an eFPGA with 54 routing channels as opposed to 44 channels, because we need a minimum of 54 channels to route all of the benchmarks covered in these experiments. As seen in Fig. 9, the SAT attack runtime trends of the ISCAS'85 benchmarks are similar to the trend of the RISC-V, which was shown to be SAT-resilient. This result shows that the redaction approach can be used to secure a variety of circuit types and is not limited to RISC-V control logic. The P-code design has a relatively lower runtime compared to the other designs, but the SAT attack runtimes still show an exponential trend. The relatively lower SAT attack runtime is likely caused by the P-code's low logic depth, which is known to have a strong effect on the SAT attack runtime [16].

VI. Conclusion

eFPGA redaction is a promising direction in hardware security that ensures that no critical IP is leaked to an untrusted fabrication source. We demonstrated that it can be used to secure a design without disrupting the ASIC design flows by redacting a RISC-V control path and a GPS P-code generator. The redaction methodology incurred 1.89x area and 2.36x delay overhead for the RISC-V block and 1.39x area and no delay overhead for the GPS block when the designs are implemented on an industrial 22nm FinFET CMOS process. We also showed that the hardware redaction using soft eFPGA is SAT resilient by mounting SAT attacks on a number of different circuits mapped on to the soft eFPGA fabric. Future work includes open-sourcing the hardware redaction flow and introducing enhancements to the eFPGA architecture to reduce the VLSI overheads even further.

References

[1] A. Baumgarten, A. Tyagi, and J. Zambreno, "Preventing ic piracy using reconfigurable logic barriers," *IEEE Design Test of Computers*, vol. 27, no. 1, pp. 66–75, Jan 2010.

[2] B. Colombier and L. Bossuet, "Survey of hardware protection of design data for integrated circuits and intellectual properties," *IET Computers Digital Techniques*, vol. 8, no. 6, pp. 274–287, 2014.

[3] 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.

[4] M. Zuzak and A. Srivastava, "Memory locking: An automated approach to processor design obfuscation," in *2019 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, July 2019, pp. 541–546.

[5] H. Mardani Kamali, K. Zamiri Azar, K. Gaj, H. Homayoun, and A. Sasan, "Lut-lock: A novel lut-based logic obfuscation for fpga-bitstream and asic-hardware protection," in *2018 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, July 2018, pp. 405–410.

[6] G. Kolhe, S. M. PD, S. Rafatirad, H. Mahmoodi, A. Sasan, and H. Homayoun, "On custom lut-based obfuscation," in *Proceedings of the 2019 on Great Lakes Symposium on VLSI*, ser. GLSVLSI '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 477–482. [Online]. Available: https://doi.org/10.1145/3299874.3319496

[7] B. Hu, J. Tian, M. Shihab, G. R. Reddy, W. Swartz, Y. Makris, B. C. Schaefer, and C. Sechen, "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*, ser. GLSVLSI '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 171–176. [Online]. Available: https://doi.org/10.1145/3299874.3317992

[8] P. N. Whatmough, S. K. Lee, M. Donato, H. Hsueh, S. Xi, U. Gupta, L. Pentecost, G. G. Ko, D. Brooks, and G. Wei, "A 16nm 25mm2 soc with a 54.5x flexibility-efficiency range from dual-core arm cortex-a53 to efpga and cache-coherent accelerators," in *2019 Symposium on VLSI Circuits*, 2019, pp. C34–C35.

[9] P. D. Schiavone, D. Rossi, A. D. Mauro, F. 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," 2020.

[10] M. M. Shihab, J. Tian, G. R. Reddy, B. Hu, W. Swartz, B. Carrion Schaefer, C. Sechen, and Y. Makris, "Design obfuscation through selective post-fabrication transistor-level programming," in *2019 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2019, pp. 528–533.

[11] C. Wolf, "Yosys open synthesis suite," http://www.clifford.at/yosys/.

[12] J. Luu, J. B. Goeders, M. Wainberg, A. Somerville, T. Yu, K. Nasartschuk, M. Nasr, S. Wang, T. Liu, N. Ahmed, K. B. Kent, J. H. Anderson, J. Rose, and V. Betz, "Vtr 7.0: Next generation architecture and cad system for fpgas," *TRETS*, vol. 7, pp. 6:1–6:30, 2014.

[13] J. Bachrach, H. Vo, B. Richards, Y. Lee, A. Waterman, R. Avižienis, J. Wawrzynek, and K. Asanović, "Chisel: Constructing hardware in a scala embedded language," in *DAC Design Automation Conference 2012*, June 2012, pp. 1212–1221.

[14] 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)*, 2015, pp. 137–143.

[15] 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.

[16] K. Shamsi, D. Z. Pan, and Y. Jin, "On the impossibility of approximation-resilient circuit locking," in *2019 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, 2019, pp. 161–170.