MANTIS: Machine Learning-Based Approximate ModeliNg of RedacTed Integrated CircuitS

Chaitali G. Sathe, Yiorgos Makris and Benjamin Carrion Schafer
The University of Texas at Dallas
Department of Electrical and Computer Engineering
{chaitaligajanan.sathe,yiorgos.makris,schaferb}@utdallas.edu

***Abstract*—With most hardware (HW) design companies now relying on third parties to fabricate their integrated circuits (ICs) it is imperative to develop methods to protect their Intellectual Property (IP). One popular approach is logic locking. One of the problems with traditional locking mechanisms is that the locking circuitry is built into the netlist that the (HW) design company delivers to the foundry which has now access to the entire design including the locking mechanism. This implies that they could potentially tamper with this circuitry or reverse engineer it to obtain the locking key. One relatively new approach that has been coined as hardware redaction is to map a portion of the design to an embedded FPGA (eFPGA). The bitstream of the eFPGA now acts as the locking key. In this case the fab receives the design without the bitstream and hence, cannot reverse engineer the functionality of the design.**

In this work we propose, to the best of our knowledge, the first attack on eFPGA HW redacted ICs by substituting the exact logic mapped onto the eFPGA by a synthesizable predictive model that replicates the behavior of the exact logic. This approach is particularly applicable in the context of approximate computing where hardware accelerators tolerate certain degrees of error at their outputs. One of the main issues addressed in this work is how to generate the training data to generate the synthesizable predictive model. For this we use SAT/SMT solvers as the potential attacker only has access to primary IO of the IP. Experimental results for various degrees of maximum allowable output errors show that our proposed approach is very effective finding suitable predictive models.

I. Introduction

The hardware (HW) design industry has been undergoing a significant transformation over the last two decades. In the past, companies were vertically integrated and design, verification and manufacture were all done in-house. This has rapidly changed due to the increase in complexity of designing today's multi-billion transistor integrated circuits (ICs). Most semiconductor companies are now fabless and rely extensively on third party IPs (3PIPs). Moreover, they often outsource significant portions of the design efforts to external third party companies, e.g., the physical design stage and/or verification. This leaves these companies extremely vulnerable to multiple security threats. Some of these include the malicious alteration of their hardware to include Hardware Trojans. Other threats includes the ability of third parties to *steal* their IP which represent their main value added. It is therefore extremely important to introduce efficient methods to enable these companies to protect their IPs.

One way to protect their IP is trough functional locking. In traditional functional locking, additional *locking* gates are added to the circuit. A logic locking key is in turn stored in a tamper-proof memory [1]. Without the correct key the locked circuit does not work as specified, either the output is incorrect (logic locking) or the performance is degraded (parametric locking). The main problem with this approach is that the locking mechanism is embedded into the design and hence can be tampered with at consecutive VLSI design stages or the fab.

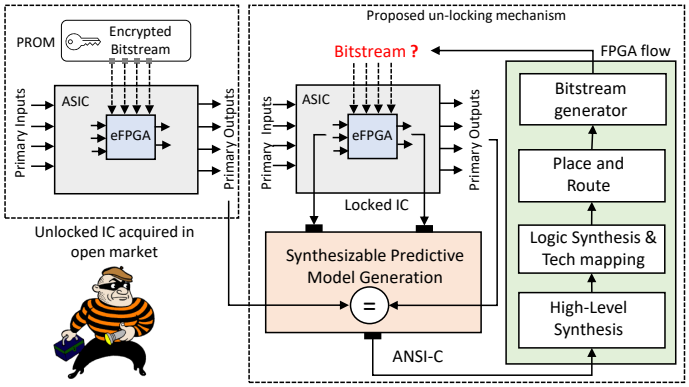

Fig. 1: HW redaction locking mechanism and proposed attack.

One relatively new way to lock hardware circuits is through hardware redaction. In this case, a portion of the circuit is mapped to an eFPGA. The eFPGA bitstream now acts as the logic locking key and the un-programmed eFPGA design is sent to the foundry for fabrication. This approach has been shown to be more secure as the search space is much larger than traditional locking methods considering that the bitstream not only determines the functional logic mapped onto the eFPGA but also the interconnect [2], [3], [4]. Thus, mapping a smaller portion of a design to an eFPGA can serve as a strong locking mechanism.

At the same time most of the ICs are now heterogeneous System-on-Chips (SoCs) that are composed of embedded processors, on-chip memory, various interfaces and a variety of hardware accelerators. Most of these components are often standard off-the-shelf IPs licensed from third party IP vendors (like ARM cores). Often the only differentiating components are the hardware accelerators. Thus, it is particularly important to protect these hardware accelerators from being reversed engineered.

The question that we aim to address in this work is if this new type of IC protection mechanisms can be broken, especially considering that most of these HW accelerators implement DSP or image processing applications that inherently tolerate some errors at their outputs. In the context

978-3-9819263-7-8/DATE23/© 2023 EDAA

of approximating computing, it has been shown that these type of accelerators can often be simplified trading-off lower power and performance with certain output error [5]. Thus, in this work we propose an attack mechanism for eFPGA HW redacted ICs by generating synthesizable (High-Level Synthesis) predictive models that can replicate the behavior of the portion of the circuit mapped onto the eFPGA. Although this will not lead to the exact output, it will generate outputs that are within a given maximum error threshold (E_{max}). We call this method MANTIS: **M**achine Learning-Based **A**pproximate Modeli**N**g of Redac**T**ed.

Fig. 1 shows an overview of how functional locking through hardware redaction works and our proposed attack mechanism, where one part of the design is mapped to the eFPGA and the rest to the ASIC. The proposed attack requires to acquire a fully working IC from the open market that will be used as an oracle (golden outputs), and a locked IC. The main ideas is to capture the inputs and outputs of the eFPGA and train a synthesizable predictive model that can then be synthesized on the eFPGA as shown in the figure. One important issue that we will address in this work is how to obtain this training data. It is impractical to assume that the attacker has access to the eFPGA IOs, and only the primary IOs of the locked hardware module. We therefore propose a Boolean Satisfiability (SAT) or Satisfiability modulo theories (SMT)-based approach to restore the outputs of the eFPGA given the primary outputs which are observable by the attacker. In summary, the main contributions of this work are:

- Present a predictive model-based attack to circumvent hardware redaction functional locking mechanisms.
- Introduce a SAT/SMT-based approach to obtain the training data for the eFPGA portion of the design.
- Present extensive experimental results to show the effectiveness of our proposed approach.

II. Motivational Example

Fig. 2 shows a motivational example for this work. In this particular it shows a flow diagram of the main computational steps involved in the JPEG encoder that takes as input an image to be compressed (input image) and performs a DCT, quantization, run-length encoding (RLE) and finally Huffman coding on it. The result is the compressed image (output image).

Fig. 2(a) shows the original unprotected ASIC implementation. Fig. 2(b) shows one possible obfuscation partitioning, which consists of mapping the RLE stage to an eFPGA. This design is now protected against reverse engineering when sent to a third-party fab as the fab does have access to the bitstream that configures the eFPGA portion of the design. The entire design is hence, partitioned into an ASIC portion that contains the DCT, quantization and Huffman coding and an eFPGA portion which contains RLE: Design=ASIC(DCT,quant,Huffman) $\cup$ eFPGA(RLE). Fig. 2(d) shows the power overhead introduced by this obfuscation approach, which for this example we measured as 9.3$\times$ the ASIC-only power (targeting 40nm GF and Quicklogic eFPGA).

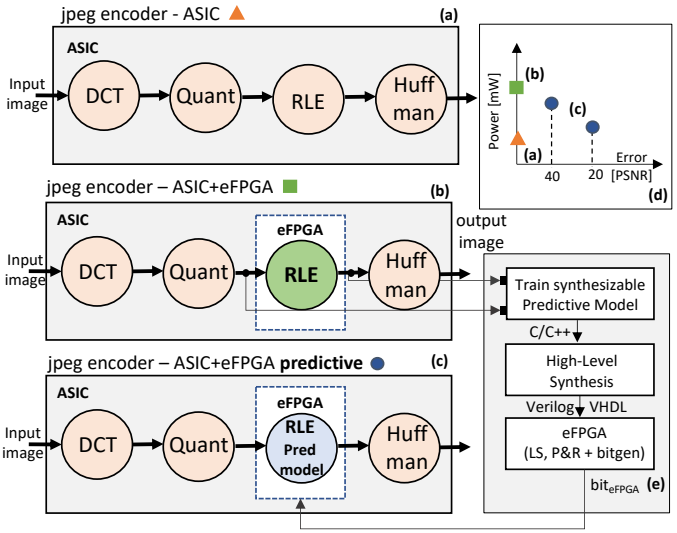

Fig. 2: Motivational example of a jpeg encoder. Original ASIC-only implementation; (b) HW redacted RLE stage mapped to eFPGA; (c) Unlocked implementation substituting obfuscated RLE stage by predictive model; (d) Power vs. error trade-offs of different implementations; (e) Proposed attack.

Fig. 2(c)+(e) show our proposed attack. The idea is to obtain the input and output data from the eFPGAs' IOs from the fully working hardware obtained legally from the open market and create a synthesizable predictive model for HLS. Because these predictive models only *approximate* the output of the eFPGA they will inevitably introduced an error in the HW module. Based on the application, the predictive model could be a simple linear regression (LR) model, or a more complex ANN-based model, with the complexity of this model also depending on the maximum allowable error (E_{max}) that the circuit tolerates.

Fig. 2 (d) shows two examples, when a Peak Signal to Noise Ratio (PSNR) of 40db is allowed vs. a PSNR of 20db. The lower the PSNR is the larger the error is, which in turn implies that the predictive model can be further simplified. In this example, we substituted FIR_2 with a LR model. This model is further simplified for the E_{max}=10 PSNR case. These predictive models are synthesized through HLS into Verilog and the bitstream for the target eFPGA generated. The result was that in both cases smaller circuits than the exact RLE design, leading at the same time to a power reduction of 17% and 27% as compared to the exact obfuscated solution (see Fig. 2(d)).

The main challenges of the proposed attach methods are: (1) How to obtain the training data for the predictive model considering that the eFPGA is integrated into the ASIC and hence, the attacker can most likely only probe the primary inputs and outputs of the entire module (jpeg encoder in this case)? (2) How to ensure that the synthesizable predictive model will fit into the eFPGA?

It should be noted that our proposed attack flow makes the following assumptions that we believe are reasonable and that will not affect the generality of our approach. Assumption 1: We can access the primary inputs and outputs of the hardware

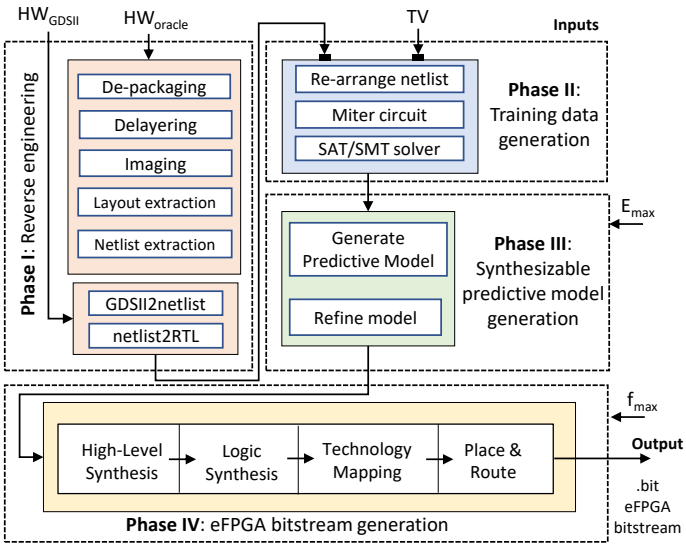

Fig. 3: Overview of proposed MANTIS attack flow composed of four main phases.(a) Phase I: ASIC part reverse engineering; (b) Phase II: Data generation; (c) Phase III: Synthesizable predictive model generation; (d) Phase IV: eFPGA bitstream generation.

accelerator e.g., directly sending and receiving data to that particular accelerator or through the scan-chain in the IC. Assumption 2: We have access to the CAD flow to program the eFPGA. This assumption is reasonable, because it is highly unlikely that the HW design company also created their own eFPGA fabric and tool flow. Most likely it licensed the eFPGA from a third-party vendor like Quicklogic or Achronix. The attacker could easily contact these third-party eFPGA vendors and obtain their tool flow. Assumption 3: The ASIC portions of the circuit can be reverse engineered either directly from the GDSII [6], or by acquiring a fully working IC from the open market and then reverse engineer it through de-packaging, delaying, imaging and layout extraction [7].

III. Related Work

Logic obfuscation can be describe as the process that transforms a circuit into another functional equivalent version that is significantly, ideally impossible, to reverse engineer. This research area has recently received significant attention due to the importance of this topic, especially considering that most HW design companies are now fabless.

Some of this research includes active metering [8], state obfuscation [9], logic encryption [10], split manufacturing [11] and design camouflaging [12]. Each of these proposed solutions has its strengths and weaknesses with no single solution successfully addressing the security and trust challenges in a cost-effective manner.

One common problem though with most of these approaches is that the fab still has access to the entire circuit including the obfuscated logic (except for split manufacturing). Thus, to address this, one relatively new approach has been to selectively extract a small portion of the circuit and mapping it to an eFPGA. This allows designers to *hide* a portion of their design and make the circuit unusable without the correct eFPGA bitstream. To the best of our knowledge, this approach was first introduce in [2], where the authors present a dedicated eFPGA fabric that they call TRAP to minimize the overhead introduced by conventional eFPGAs. Although the authors showed that this approach works well, the dedicated fabric is only usable to hide simple combinational logic. In this work the authors also present a partitioning flow for RTL descriptions. More recently the authors in [4] proposed ALICE, an automated flow that partitions the RTL modules between one or more reconfigurable fabrics and the rest of the circuit, automating the generation of the corresponding redacted design.

Other work raises the level of design abstraction by partitioning the design at the behavioral level. In [3] Bo et al. presented an automated partitioning flow for behavioral descriptions for HLS, where the authors encapsulate different portions of the behavioral description into functional operators that they then map to the eFPGA. Zi et al. presented a similar approach but doing a finer partition at the CDFG generated after parsing the behavioral description in [13]. Finally, this ASIC+eFPGA flow was also shown to be effective to *hide* implementation details of two functionally equivalent designs, e.g., ANN activation functions [14]. In [15] the authors studied how to reduce the area overhead introduced by the eFPGA by using runtime reconfigurable coarse grain FPGAs. Closer to this work, the authors in [16] proposed a similar approach buy assumed that the inputs and outputs of the eFPGA are fully accessible, hence, significantly simplifying the problem. This work extends this work proposing a method to obtain the training data efficiently.

IV. Proposed Attack Flow

Fig. 3 shows an overview of the complete attack flow, which consists of four main phases. The inputs to the flow are the fully working IC with a locked component obtained legally from the market (HW_{oracle}) or a GDSII file of the same (HW_{GDSII}). This IC will be used as an oracle to verify if our predictive model works within the maximum specified output error E_{max}, which must be specified as another input to our flow. This E_{max} is obviously application dependent and is typically specified in Mean Absolute Percentage Error (MAPE) for signal processing applications or PSNR for image processing applications. MAPE calculates the error between the output of the approximate circuit and the golden output from the exact solution as follows: $\text{MAPE} = \frac{1}{N}\sum_{i=1}^{N}\left|\frac{GO_i - APO_i}{GO_i}\right|$, where GO_i is the error-free output (golden output) obtained from the fully functioning oracle, and APO_i is the approximate output when the eFPGA is programmed with our predictive model. N is the total number of outputs observed. Our flow also requires the test vectors or actual workload that the IC will be subject to under normal conditions (TV), the operating frequency (f_{max}) and finally, the CAD flow to generate a bit stream for the eFPGA. The output of the flow is the eFPGA bitstream that replicates the behavior of the original logic mapped onto

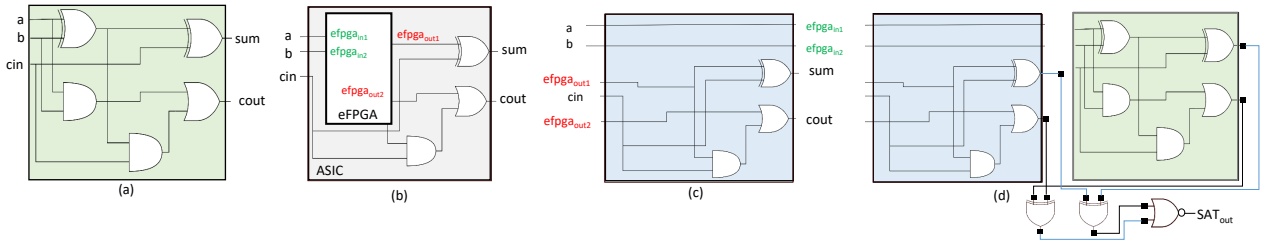

Fig. 4: Example of SAT solver used to obtain training data for predictive model. (a) Full adder gate netlist. (b) Full adder with HW redaction example. (c) Gate netlist re-arrangement to prepare for SAT solver. (d) Miter circuit including fully working design to obtain training data.

the eFPGA ($efpga_{bit}$). The next subsections describe the four phases of our proposed attack in detail:

Phase I: ASIC part reverse engineering: The first phase of the proposed attack reverse engineers the ASIC portions of the circuit, either by obtaining a fully working IC from the market and then de-packaging, delayering, imaging and extracting its layout as shown in Fig. 3 [7] or reverse engineering the GDSII description if available [6]. This work does not deal with this and assumes that the ASIC portion of the circuit can be reversed engineered into a gate netlist. This has been previously shown in other work including [7], [6]. The reversed engineered gate netlist of the ASIC portion can be further converted into a functional equivalent RTL description [17], [18]. One of the advantages of working at the RT-level instead of at the gate netlist level is that more complex sequential circuits can be analyzed faster.

Phase II: Training Data Generation: This second phase generates the training data to generate the predictive model. Unfortunately, under normal circumstances the attacker only has access to the data at the primary IOs of the hardware accelerator, but not at the inputs and outputs of the eFPGA, which is what our flow needs to train the predictive model.

To address this we will use the reverse engineered gate netlist or RTL obtained in phase 1. This phase starts by running a traditional workload that uses the locked HW accelerator and by annotating the values at the primary inputs (TV) and primary outputs (GO Golden outputs). These TV are then applied to the gate netlist or RTL obtained in phase 1 in order to retrieve the inputs to the eFPGA ($TV_{ineFPGA}$). The main problem is how to obtain the outputs of the eFPGA module. For this we make use of a SAT solver in the case of having only the gate netlist and an SMT solver in the case of the RTL. The solvers return the outputs of the eFPGA which are basically the inputs of the solver, knowing the primary outputs of the module (GO). SMT solvers provide a much richer modeling capability than SAT solvers by adding equality reasoning, arithmetic, bit-vectors, quantifiers, arrays and other useful first-order theories. Thus, SMT solvers are well suited to find the eFPGA outputs for complex larger ASIC circuits.

Fig. 4 shows how this is accomplished using a full-adder as an example. Fig. 4 (a) shows the gate netlist of a typical full adder. Fig. 4(b) shows the HW redacted circuit with two gates mapped to the eFPGA. Fig. 4 (c) shows how the ASIC portion of the netlist is re-arranged in order to obtain training data of the eFPGA (inputs and outputs of the eFPGA portions only). As shown, the inputs of the eFPGA ($efpga_{in}$) are now primary outputs of the resultant gate netlist and the outputs of the eFPGA ($efpga_{out}$) inputs to the netlist. By simulating this netlist alone our method can only find the input values of the efpga which is part of the training data required.

To obtain the eFPGA outputs, which are now primary inputs of the newly re-arranged netlist, we make use of a SAT solver. For this we build as shown in Fig. 4 (d) a miter circuit with the fully working design and our re-arrange netlist. The outputs are xor'ed and the outputs of the xor gates or'ed and inverted. Basically, the idea is to find the inputs of the circuit that make SAT_{out}=1. This will reveal the values of $efpga_{in}$. The SAT solver is called for different TVs, and although this process can be time consuming, it can also be easily parallelized. SMT solvers work similarly, but much faster as they work at the word-level.

The data obtained serves as the training data to generate the synthesizable predictive model $data_{train} = \{efpga_{in}, efpga_{out}\}$. In order to make sure that there are no timing issues later when our method substitutes the eFPGA portion by a predictive model, this step also records the latency (clock cycles required to produce a new output) of the eFPGA, such that the predictive model can fully replicate the timing (L_{eFPGA}).

Phase III: Synthesizable Predictive Model Generation: This phase takes the training data extracted in phase II as input and searches for a predictive model that can replicate the behavioral of the original circuit. The predictive model needs to fulfill three major conditions: **Condition 1:** The synthesized model has to fit in the given eFPGA fabric. **Condition 2:** The accelerator's outputs should be within the maximum specified error threshold (E_{max}). **Condition 3:** The approximated circuit should operate at the same frequency and have the same latency. As shown in Fig. 3, this phase is further sub-divided into two steps:

Step 1: Generate Predictive Model: This step takes the training data obtained from phase 1 ($data_{train}$) and performs a predictive model fitting. To accomplish this, the data is passed to a well-known predictive model toolkit (scikit-learn [19]), which in turn returns the predictive model and the statistical information regarding this model. Different predictive models are considered in this work and the simplest that meets the conditions listed previously is used. E.g., the simplest one is a linear regression (LR) model. Other models used include regression trees, which consists of multiple LR models, and multi-layer perceptron (MLP). This last model is more com-

plex than the LR models but is more efficient when the outputs are highly un-correlated as MLPs can better model those non-linear behaviors. MLPs also scale very well when the HW redacted portions has a larger number of inputs and outputs.

The predictive model generator starts by generating the simples possible predictive model and then proceeds by computing the predictive model's output, and by computing its error (MAPE) and comparing it with E_{max}. If the error is smaller, then the model is valid and the process stops. If not, the search continues by moving to the next more complex predictive model. Once a suitable model is found, the model is translated into synthesizable ANSI-C code for HLS. The output of this step is a synthesizable C description of the simplest predictive model that operates within E_{max} (C_{pred}).

Step 2: Predictive Model Refinement: The predictive model generated in step 1 is further optimized in this step by obtaining the smallest possible model that operates within E_{max}. For this, this step starts by deleting the smallest terms in the predictive model. E.g., in the LR model (based on the coefficients obtained) or neurons in the MLP case that have the smallest weights and biases. This step is done sequentially and after each optimization the proposed flow checks if the output error is still within E_{max} by running a full simulation. This refinement step exits when E_{max} is reached and returns the smallest model that does not exceed the given error threshold. The output of this steps is a new refined behavioral description of the synthesizable predictive model (C_{opt}). Moreover, if the latency of the HW redacted circuit is not one (it is not purely combinational), then our method inserts a delay loop into the C_{opt} such that $L_{opt} = L_{eFPGA}$.

Phase IV: eFPGA Bitstream generation: This last phase takes as input the optimized predictive model that was generated in phase 2 (C_{opt}) and generates the eFPGA bitstream ($efpga_{bit}$) to configure the eFPGA with the predictive model. Considering that the model is generated in synthesizable C code for HLS, the entire process, as shown in Fig. 3, consists of HLS, logic synthesis, technology mapping, place and route and finally eFPGA bitstream generation. In all cases the target synthesis frequency should be set to the same frequency at which the exact obfuscated circuit works. This will now enable the *unlocking* of every manufactured chip.

TABLE I: Experimental Setup

HLS Tool	NEC CyberWorkBench 6.1.1
Synthesis frequency	500MHz
RTL Simulator	Synopsys VCS 2018.06
eFPGA technology	Quicklogic ArticPro
SMT solver	pySMT 0.9.5 Z3-Solver
Machine learning tool	scikit-learn 0.24.2 [19]

V. Experimental Results

Table I shows an overview of the experimental setup used to test our proposed flow. Because our proposed MANTIS flow works works at the gate netlist and RT-level, we choose RT-level and thus, use pySMT 0.9.5 Z3-Solver SMT solver to obtain the eFPGA training data.

Six benchmarks from the S2CBench benchmark suite [20] from different domains and different complexities amenable to approximate computing are used to measure the effectiveness of our proposed flow. We use the testbench provided with these benchmarks to measure the output error which based on the benchmark. We set the maximum output error (E_{max}) to 10% MAPE further relax it to 20% MAPE to investigate how relaxing the output error constraint affects the results.

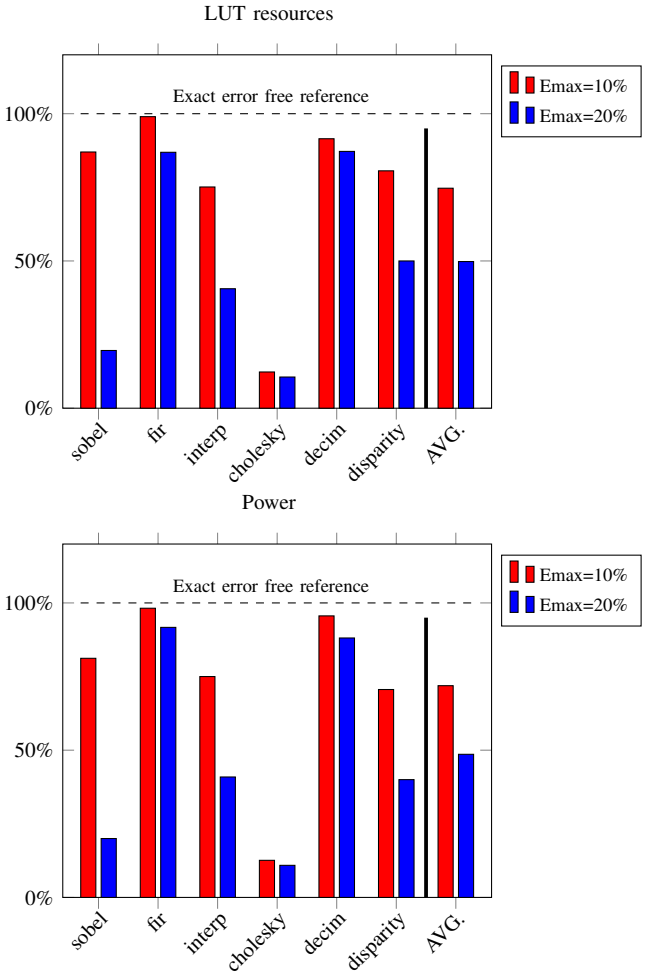

Fig. 5: Reconfigurable computing resources (LUTs) and power consumption comparison of our proposed approximated accelerators for two different error thresholds (Emax=10% and Emax=20%) vs. the exact, error free implementation.

As baseline exact HW redacted design (ASIC+eFPGA), we follow the instructions from [3] to partition the benchmarks into an ASIC and an eFPGA partition. Because there are multiple possible partitions, we choose the smallest partition that has a *time-to-break* (TTB) for SAT and brute-force attacks of at least 1 year.

Fig. 5 shows the experimental results for the two different error thresholds compared to the exact partitioned solution. In all cases the latencies match the latencies (input to output clock cycles of the eFPGA portion) of the exact version. The percentage in each bar indicate the area reduction/increase of our proposed method considering the exact solution as 100% (e.g., in the interp benchmark for E_{max}=10% the total area is 75% of the area required by the exact solution and for

E_{max}=20% only 41%. Thus, the area savings are 25% and 59% for the corresponding maximum error thresholds.

Several interesting observations can be made from these results. Observation 1: In all cases, the predictive model substitution method was able to find a valid solution that was equal or smaller than the exact solution. Observation 2: Relaxing the output error threshold (E_{max}) leads to larger area and power savings. On average, the area and power savings increased by 33% and 31% when the maximum error was relaxed from 10% to 20% MAPE. Observation 3: One interesting side effect of our proposed approximation approach is that it leads to designs with lower power. On average the power (for the eFPGA portion only) was reduced by 28% and 60% for the different error thresholds.

TABLE II: Runtime summary of our proposed MANTIS flow (Phase 2 and 3) with different E_{max}.

Bench	E_{max}**=10%MAPE Run[min]**	E_{max}**=20%MAPE Run [min]**	**Predictive Model**
sobel	1.92	2.72	LR
fir	2.75	4.25	LR
interp	2.15	3.15	LR
cholesky	2.41	2.84	LR
decim	5.12	7.56	PolyReg
disparity	7.65	9.43	MLP
Geomean	**3.19**	**4.41**	

Table II reports the running time of our proposed flow (Phase 2 and 3 only) for the two different E_{max}, and also the predictive model finally used for each of the benchmarks, where the geometric average is used to account for the benchmarks size differences. The results show that the running time is relatively low averaging an average of 3.19 min and 4.41 min for the different error thresholds, with at most taking 9.43 min in the disparity case. One of the reasons that the runtime is relative low is because we used the SMT solver instead of the SAT solver. We believe that this is very reasonable especially considering that this flow only needs to be execute once. From the results it can also be observed that relaxing the error constraints leads to slightly larger running times due to the larger search space when the predictive model is fine tuned. The table also shows that there is not a unique predictive model that works for every benchmark. Thus, it is important to have an automated flow that will automatically try different predictive models like in our work and choose the best (smallest) that meets the given constraints.

These results allow us to conclude that our proposed method is effective in finding a predictive method that can substitute the exact circuit portion mapped onto the eFPGA to obfuscate the design under different error thresholds.

that is in turn mapped onto the eFPGA using HLS. Our

VI. Conclusion

In this work, we have introduced a framework to reverse engineer a hardware redacted circuit that has been partitioned into an ASIC and an eFPGA part through synthesizable predictive models. Because most of these circuits are hardware accelerators, they often tolerate errors at their outputs in the context of approximate computing. Thus, we present a method based on generating a synthesizable predictive model method uses SAT/SMT solvers to obtain the training data as the attacker will only have access to the primary IOs of the accelerator and not the eFPGA and automatically tries different predictive models as we have shown that for different benchmarks different models lead to better results. To the best of our knowledge this is the first attack that tries to break these type of locked circuits. We hope that the hardware design community will react to the ideas presented in this paper to make make HW reduction more secure.

VII. Acknowledgements

This work is partially supported by the NSF Industry/University Cooperative Research Center on Hardware and Embedded Systems Security and Trust (CHEST) through project #P16_22.

References

[1] M. T. Rahman *et al.*, "Defense-in-depth: A recipe for logic locking to prevail," *Integration, the VLSI Journal*, vol. 72, pp. 37–57, Jan 2020.

[2] M. M. Shihab *et al.*, "Design obfuscation through selective post-fabrication transistor-level programming," in *DATE*, 2019, pp. 528–533.

[3] B. Hu *et al.*, "Functional obfuscation of hardware accelerators through selective partial design extraction onto an embedded fpga," in *GLSVLSI*, 2019, p. 171–176.

[4] C. M. Tomajoli *et al.*, "ALICE: An Automatic Design Flow for eFPGA Redaction," in *Design Automation Conference*. ACM, jul 2022.

[5] Q. Xu, T. Mytkowicz, and N. S. Kim, "Approximate computing: A survey," *IEEE Design Test*, vol. 33, no. 1, pp. 8–22, Feb 2016.

[6] R. S. Rajarathnam, Y. Lin, Y. Jin, and D. Z. Pan, "Regds: A reverse engineering framework from gdsii to gate-level netlist," in *2020 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, 2020, pp. 154–163.

[7] U. J. Botero *et al.*, "Hardware trust and assurance through reverse engineering: A tutorial and outlook from image analysis and machine learning perspectives," *J. Emerg. Technol. Comput. Syst.*, vol. 17, no. 4, jun 2021.

[8] J. A. Roy *et al.*, "Epic: Ending piracy of integrated circuits," in *DATE*, 2008, pp. 1069–1074.

[9] R. S. Chakraborty and S. Bhunia, "HARPOON: An Obfuscation-Based SoC Design Methodology for Hardware Protection," *IEEE TCAD*, vol. 28, no. 10, pp. 1493–1502, Oct 2009.

[10] J. Rajendran *et al.*, "Fault analysis-based logic encryption," *IEEE Transactions on Computers*, vol. 64, no. 2, pp. 410–424, Feb 2015.

[11] ——, "Is split manufacturing secure?" in *2013 DATE*, 2013, pp. 1259–1264.

[12] ——, "Security analysis of integrated circuit camouflaging," in *SIGSAC Conference on Computer & Communications Security*, ser. CCS '13, 2013, pp. 709–720.

[13] Z. Wang *et al.*, "Functional locking through omission: From hls to obfuscated design," in *ICCD*, 2021, pp. 591–598.

[14] J. Chen *et al.*, "DECOY: DEflection-Driven HLS-Based Computation Partitioning for Obfuscating Intellectual PropertY," in *DAC*, 2020, pp. 1–6.

[15] J. Chen and B. Carrion Schafer, "Area efficient functional locking through coarse grained runtime reconfigurable architectures," in *ASP-DAC*, 2021, p. 542–547.

[16] P. Chowdhury, C. Sathe, and B. Carrion Schaefer, "Predictive model attack for embedded fpga logic locking," in *ISLPED*, 2022.

[17] D. Watkins *et al.*, "Gate netlist to register transfer level conversion tool," U.S. Patent US08/668,064 Jun. 1996.

[18] P. Subramanyan *et al.*, "Reverse engineering digital circuits using structural and functional analyses," *IEEE Transactions on Emerging Topics in Computing*, vol. 2, no. 1, pp. 63–80, 2014.

[19] F. Pedregosa *et al.*, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[20] B. Carrion Schafer and A. Mahapatra, "S2CBench:Synthesizable SystemC Benchmark Suite," *IEEE Embedded Systems Letters*, vol. 6, no. 3, pp. 53–56, 2014.