MAAS: Hiding Trojans in Approximate Circuits

Qazi Arbab Ahmed[1,2], Muhammad Awais[1,3], and Marco Platzner[1]

[1]Paderborn University, Germany
[2]University of Azad Jammu and Kashmir, Muzaffarabad, Pakistan
[3]Quaid-e-awam University of Engineering, Science & Technology, Nawabshah, Pakistan

***Abstract*—Automated frameworks for approximate accelerator synthesis employ an iterative search-based approach to generate approximate instances of hardware. While offering distinct savings in terms of hardware area and power consumption, approximate circuits are potentially at risk of being infected with hardware Trojans mainly due to the fact that the approximation is typically provided by third-party approximate accelerator synthesis frameworks which utilize components libraries to perform substitutions during the design space exploration phase. In this paper, we propose a threat model that discusses the potential of hardware Trojans insertion during the approximate accelerator synthesis. Moreover, we present *MAAS*, a framework that exploits a search-based approximate accelerator synthesis technique to demonstrate the applicability of our threat model by hiding Trojans in approximate circuits. The experimental results show that the approximate circuits generated by *MAAS* containing infected hardware Trojans are slightly larger than the approximate designs and are hard to identify via conventional area and power measurement techniques. To the best of our knowledge, this is the first effort to demonstrate the hardware Trojan insertion in the third-party approximate accelerator synthesis flow via library component substitution.**

***Index Terms*—Approximate Computing, Hardware Trojans, Approximate Circuit Synthesis**

I. Introduction

Approximate Computing (AC) has emerged as an effective alternative for performance improvement in computing systems. It targets the intrinsic error resilience present in several applications, e.g., image and signal processing, computer vision, and machine learning. Approximate computing has been largely applied to generate hardware accelerator circuits that offer substantial benefits of power consumption, area, and delay while occasionally providing erroneous yet acceptable outputs. Existing works in approximate accelerator circuits synthesis provide various automated frameworks, such as the one proposed in [1], which accepts the original circuit description along with an error bound defined by the user and generates an approximate version of the circuit that adheres to the given bounds. The main target of approximation is arithmetic components, such as adders and multipliers. However, security breaches due to the inclusion of third-party's malicious arithmetic components in a circuit have not been investigated yet.

Recent advancements in automated approximation circuit synthesis, like the MCTS-based framework [1] and the CIRCA framework [2], have demonstrated significant reductions in area, power, and delay. These frameworks could, however, be a simple target for an attacker to infiltrate the approximate synthesis flow, either by adding malicious code via a compromised tool chain or by a bad employee binding a Trojan during the synthesis.

Third-party EDA tools may be untrustworthy [4] and subverted EDA tools can be used to insert malicious logic into a circuit [5]. Recently, Ahmed et al. in [6] have demonstrated an FPGA design flow attack to leak secret information from an encrypted module by using the compromised EDA tools. This is considered to be a powerful attack since it maintains control over the circuit throughout the design flow while still being sneaky. In such attacks, the executable binaries of certain tools in an EDA tool chain are replaced by malicious ones to carry out the successful attack. There may also be a hidden system of communication between the tools for the insertion and activation of malicious logic if more than one tool is engaged. These kind of attacks can be engineered for a targeted user/device or the entire design batch could be undermined over the internet to perform higher-level attacks and can circumvent techniques such as Proof-Carrying Hardware (PCH) [7] used for the verification of tools' binaries.

To investigate the aforementioned threat model, the state-of-the-art approximate circuit synthesis flows [1], [2] can be manipulated with malicious approximate components, which likely meet the constraints such as error and performance parameters but may result in catastrophic results later when the Trojan is activated. Furthermore, Trojan insertion in approximate circuits has so far been limited to the netlist level, whereas state-of-the-art approximate circuit synthesis frameworks, i.e., [1], target higher levels of abstraction, such as mirco-architectural level e.g., a design written in SystemC.

In order to evaluate hardware Trojan insertion in the approximate accelerator circuits synthesis flow at a higher level of abstraction, we propose a Malicious Approximate Accelerator Synthesis *(MAAS)* framework, which conceals a Trojan in approximate circuits in a way that keeps it below the error threshold for circuit verification and activates under specific conditions.

In short, *MAAS* demonstrates how a third-party synthesis flow can insert hardware Trojan during the generation of approximate circuit synthesis. The main contributions of this paper are as follows:

- We propose a threat model for hardware Trojan insertion

This work has been partially supported by the German Research Foundation (DFG) within the Collaborative Research Centre 901 "On-The-Fly Computing" under the project number 160364472.

in approximate accelerator synthesis via an automated search-based synthesis framework. To the best of our knowledge, this is the first work that inserts hardware Trojan in approximate circuits at the micro-architectural level, e.g., SystemC design.

- We propose a novel Trojan insertion framework, *MAAS*, that exploits a search-based framework to generate approximate circuits with hardware Trojan insertion via library component substitution and experimentally demonstrate its applicability on various practical benchmarks with marginal differences in the area and power consumption results of approximate and approximate+infected designs.

The rest of the paper is organized as follows: Section II lays out the previous efforts in the field of approximate accelerator synthesis as well as of hardware Trojan insertion. In Section III, the threat model and the proposed *MAAS* framework are explained. Experimental results and discussions are provided in Section IV. Finally, the paper is concluded in Section VI.

II. Related Work

The development of real-time data-intensive and intelligent applications has given a boost to both academia and industries to modernize the current architecture to a new paradigm that could enhance the performance and reduce the power consumption and area of a circuit. The approximate computing paradigm has been in competition for a few years and performs exceptionally well in terms of area saving and power efficiency of a circuit while maintaining an acceptable output. However, security is the major concern to adopt yet another paradigm for future computing which has largely been ignored so far. Besides, confidentiality, integrity, and design theft, a circuit can be infiltrated with additional/malicious circuitry, also known as hardware Trojan [8]–[10]. Under particular conditions, a hardware Trojan could alter the functionality of the circuit, leak information, or introduce a denial of service attack.

Regazzoni et al. [11] have discussed some of the possible security outcomes and concluded that the Approximate Circuit (AxC) could be an easy target for hardware Trojans insertions. However, no practical example of a hardware Trojan threat has been demonstrated. Considering the implementation, approximate computing can be classified and separated into four different schemes based on abstraction levels such as system-level approximation, software-level approximation, circuit-level approximation, and storage approximation.

Yellu et al. in [12] have generally assessed the vulnerability of AxC to each of the levels, however, the authors focused on the potential attacks on the storage, while the hardware Trojan attack on higher levels has been overlooked. Ariful Islam in [13] presented a hardware Trojan attack on the approximate computing system during the synthesis of an approximate module. The author analyzed each of the approximate modules with respect to architecture and the required objective function to insert the hardware Trojan. However, the approach works only on the netlist level. The approximation on the higher levels of the circuit design, i.e., the microarchitectural level has a higher impact on the objective function, therefore, could be an attractive abstraction for an attacker to target. Wang et al. in [14] evaluated the data modification vulnerability in the AxC modules and demonstrated it on the Approximate Error Tolerant Adder Type I (ETAI), where the error produced by the infected circuit remains inside the boundary of the error threshold set by the user. The precision flag is not modified, thus, the error produced by modifying the data bits would not be distinguishable from the original error. However, the attack is not evaluated for other approximate components used in approximate accelerator circuits synthesis.

Arithmetic circuits such as adders and multipliers are the main target in approximate computing where the approximated bits directly affect the transition probability of the circuit. A lower transition probability could be an exciting opportunity for an attacker to hide a Trojan circuit/trigger, which can be exploited in later stages to jeopardize the functionality of the circuit. Dou et al. in [15] explored and assessed the transition probability of an exact low-part-or-adder (LOA) circuit and the approximate version of that which shows the significant reduction in the transition probability between the exact and the approximate circuit when the number of modules is increased.

III. Methodology

This section introduces the threat model that is considered to demonstrate our attack, followed by an explanation of the approximate accelerator synthesis and the proposed Trojan insertion framework in approximate circuits.

A. Threat Model

Approximate computing provides an opportunity for an attacker to hide a Trojan circuit due to its inexact nature of components and lack of a powerful testing mechanism. On the other hand, it is challenging for an attacker to reverse engineer the circuit to retrieve the original design as some of the components are modified or approximated. There may

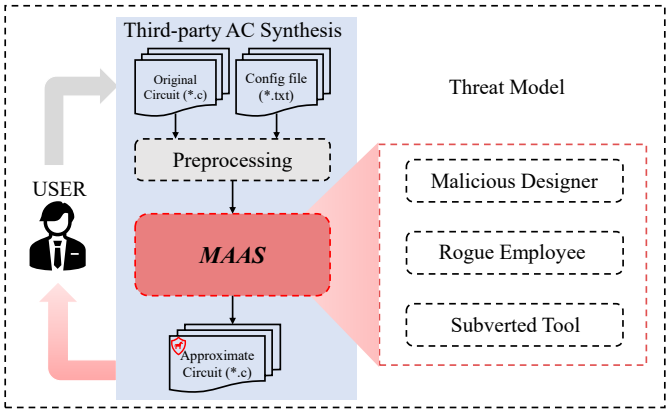

Fig. 1: Threat Model: Hardware Trojan insertion possibilities in approximate circuits.

be multiple ways to insert a hardware Trojan into an approximate circuit, however, in this paper we target the Trojan insertion by a third party during the approximate accelerator circuits synthesis. Figure 1 illustrates the threat model we have proposed and followed in this paper. Our threat model in a third-party approximate accelerator circuits synthesis flow is based on the following assumptions. a) malicious designer could replace the approximate components with the Trojan inserted/compromised approximated components. b) a rogue employee can deliberately change the configurations of approximate components and c) a subverted approximate accelerator synthesis tool can automatically replace the approximate components with the compromised ones during the synthesis of a circuit, without the knowledge of a designer or the user. This work is based on assumption c) and practically demonstrates the subversion of approximate accelerator synthesis tool to hide Trojans in approximate circuits. Note that, the original components can also be replaced by any of the malicious parties, however, this would have a large impact on the output parameters of the circuit. So, the attacker's aim is to target the approximate circuit to insert Trojans, as the user would not be able to notice any changes in the parameters due to the approximation, while the error bound check would still be verifiable by the user.

Based on our threat model, the existing frameworks for approximate accelerator circuits synthesis can potentially be exploited to insert malicious logic during the synthesis flow. Majority of the existing frameworks follow an iterative search-based flow and employ component substitution via component library provided as input to the framework for approximation [2]. More often, open-source component libraries are available from multiple sources providing large number of implementations offering trade offs of area and power consumption at different error values. Therefore, in this paper, we implement our framework as an iterative search-based approximation flow to show that the third party approximate accelerator circuits tools are vulnerable to the Trojan attacks.

B. Automated Approximate Accelerator Synthesis

Automated generation of hardware accelerators from the original design is a quite challenging task. Typically, the process involves exploring a large design space of possible solutions with varying trade-offs for error and target metric, such as circuit area and power consumption. This iterative process starts with the original circuit configured as the root or seed and then repeatedly improves the hardware cost by applying approximations on the current design and generating new designs from the current design. The process only exits when there is no improvement possible without exceeding the allowed error threshold set by the user.

In the presence of multiple objectives to improve, such as hardware area and power consumption, where the error is included as a constraint, automated synthesis of approximate instances becomes a complex optimization problem. In addition, a large number of arithmetic components, that can be used to substitute exact components of the circuits, spawn an enormously large solution space that cannot be explored exhaustively. To deal with such a massive number of combinations in a reasonable time, often a heuristic (such as greedy-based) approach is used with a downside that it could overlook promising combinations [3]. A more extensive reward-based albeit time-consuming approach is to employ a learning-based algorithm such as Monte Carlo Tree Search (MCTS) to find promising paths in the search space [1]. In this work, we implement a search-based approximation flow based on a rather balanced search policy i.e., a modified version of MCTS [1], as a reference implementation and manipulate its approximation step to demonstrate hardware Trojan insertion via third-party synthesis tools. The modified approximation framework coupled with Trojan infected component library is what we call *MAAS*. It could accept an accelerator design coded in SystemC and generates an approximated version of the design that provides maximum area savings under the given error threshold. *MAAS* follows a two stage design space exploration policy. In the first stage, it generates analytical models from the training data obtained from the simulations of the design variants. In the following stage, it explores the design space using a reward-based MCTS search method to find feasible designs via an iterative process. The designs are then filtered and the best design in terms of area savings estimate is then chosen as the output of the framework.

C. Malicious Approximate Accelerator Synthesis (MAAS)

This paper presents *MAAS*, a new hardware Trojan insertion framework, targeting third-party approximate accelerator circuit synthesis flow to hide malicious logic in approximate circuits. Figure 2 shows the complete flow of the proposed *MAAS* framework. The preprocessing stage of the framework identifies components for approximation and generates training data from the simulations of various configurations of the original design and gathers error information. Based on the obtained data, it creates lightweight analytical models that are capable of faster error estimation than the conventional cycle-accurate simulation.

The main step of iterative approximation runs a search-based flow for a predefined number of iterations. In each iteration, one of the identified components is identified for approximation, and a new approximate configuration is obtained from a node by approximating one of the non-approximated operations in the parent node. However, *MAAS* uses a pre-characterized library of approximate+infected operators which are obtained by inserting malicious logic in an extended version of EvoApproxLib [17], an open-source library of approximate components. Consequently, the new approximate configuration contains the infected component substituted. The subsequent step verifies that the new approximate configuration adheres to the quality bounds set by the user by performing an error estimation step. The error estimation takes place by invoking lightweight error estimation models trained in the preprocessing step. Note that, the error estimation only checks the error magnitude information of the approximate configuration. Any additional logic inserted in the approximate

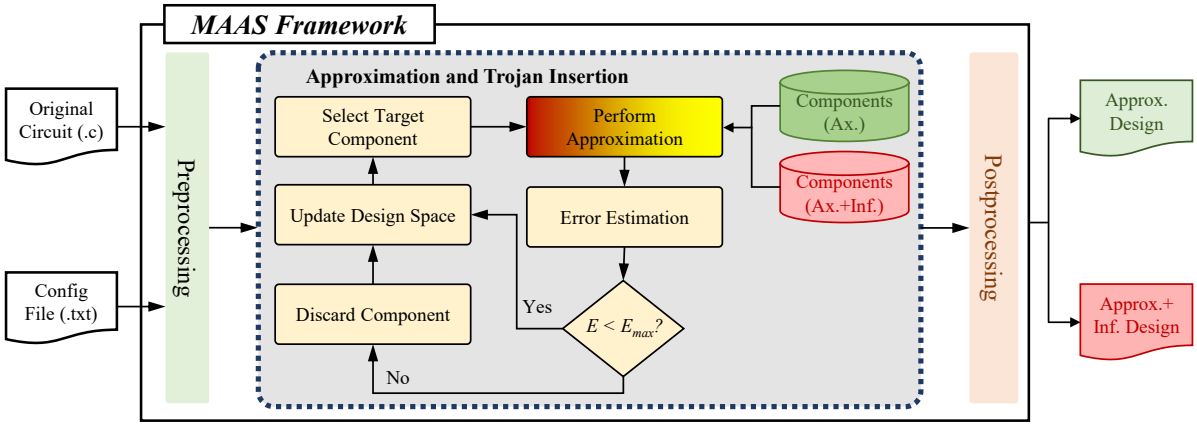

Fig. 2: *MAAS* framework: (i) Accepts original design, along with a configuration file, (ii) performs preprocessing including loop unrolling and component identification, (iii) performs iterative approximation during which the approximation step is manipulated to insert/select malicious components from an infected library. The following steps perform error estimation of the approximated variant and update design space statistics. (iv) As postprocessing, the framework generates a synthesizable Verilog file of a design, and finally, (v) the framework outputs either the approximate design or Trojan inserted approximate design based on the settings described in the configuration file.

design could not be checked in this step since typically approximate components (adders and multipliers) have various implementations each having a significant difference in terms of hardware area. This is exactly the reason what we advocate in this paper i.e., that malicious logic could be hidden behind the approximation step.

In order to demonstrate the hardware Trojan insertion in an approximate accelerator circuits synthesis flow, we first generate Trojan-infected versions of the components provided in the EvoApproxLib. We select the components that lie on the Pareto-frontier line for the mean arithmetic error vs. power consumption. For each of the components, we modify the circuit description by including malicious logic which triggers when a specific input pattern is encountered. For instance, for the adder circuit that accepts two 16-bit inputs a and b, the adder logic would produce a highly invalid output when both a and b are zero. We applied this trigger logic ($a = b = 0$) to all the benchmarks having different adders and multipliers. The payload on activation turns the output bits to "1" thus resulting in an erroneous output. For instance, in case of an image processing filter, e.g., *Gaussian Blur Filter*, this could produce catastrophic results on the activation of the payload for multiple instances of the infected components. As a result, the image pixels could get highly corrupted, consequently compromising the underlying algorithms that typically perform critical decision-making in applications such as autonomous driving, security surveillance, and medical imaging technology, just to name a few. In all other cases, the adder circuit would produce a legitimate output. This is a trivial example that shows how malicious logic can turn an approximate component into a hardware Trojan.

The infected version of the components is then included in the library and provided to the framework as input. The framework is then configured to choose the infected components as replacements to generate the approximate instances of the accelerator circuit. During the run of the approximation flow, the framework chooses an available candidate from the circuit and replaces it with an approximate component from the library. Since the library is now containing infected components, the framework seamlessly can integrate the infected component into the circuit as the approximation process progresses. It is important to note that this malicious activity will remain unnoticeable to the testing/verification step of the framework since the test vectors which are used to evaluate the accuracy of the circuit during the verification step are also provided to the framework and do not contain the trigger combination that invokes the Trojan. Hence, the verification step will behave exactly the same as it was with the non-infected versions of the components. The attacker depending upon the threat model can activate the Trojan when the approximate circuits are integrated into a system-on-chip (SoC), e.g., an attacker can activate the trigger logic by giving an input pattern ($a = b = 0$) to activate the payload thereby resulting in illegitimate output.

Once the framework completes the assigned number of iterations, the next step is to identify the best available version of the approximate or approximate+infected circuit that provides the maximum improvement in the target metric (in this case the hardware area). This step is accomplished in the post-processing step with a heuristic function that evaluates all the designs explored during the process and chooses the best design. The design is then realized as a Verilog circuit and synthesized to estimate the actual area and power values.

IV. Experimental Setup

Table I shows the details of the accelerator circuits that are manipulated by *MAAS* to demonstrate the hardware Trojan insertion. We have selected four commonly used benchmark accelerator circuits, e.g., *Ternary Sum* and *FIR Filter*, having 16 inputs/outputs (I/O) each, and *RGB2GRAY* and *Gaussian Blur Filter*, each having 8 inputs/outputs. All accelerator circuits are coded in SystemC whereas the area and power consumption information of each of the circuits is obtained by synthesizing the equivalent Verilog circuits using Synopsys Design Compiler. The circuits range from small to medium-

TABLE I: Benchmark accelerator circuits

Circuit	I/O	QoR [§]	Area (μ^2)[*]	Power (mW)[*]
Ternary Sum	16/16	MRE(%)	454.00	1.04
FIR Filter	16/16	MRE(%)	7485.26	6.06
RGB2GRAY	8/8	MRE(%)	2427.50	0.10
Gaussian Blur Filter	8/8	PSNR	7729.23	0.75

[§] Error metric used for quality of results.
[*] The area and power of the accelerator circuits are measured using Synopsys Design compiler using a 22nm technology library

sized accelerators and are selected from different domains such as arithmetic, signal and image processing. Mean relative error percentage (MREP) is used as the error metric for all the circuits except the *Gaussian Blur Filter* which is evaluated with peak signal-to-noise ratio (PSNR), a commonly used error metric for image processing applications.

For hardware Trojan insertion, SystemC description of the adders and multipliers of EvoApproxLib [17] is altered by adding the malicious logic that triggers the payload when the specific input patterns occur. The infected components are then added to the library of the approximate components and the resultant component library is referred to as *approx.+inf.* component library. This allows for the substitution of approximate components with infected components during the approximation process (see Figure 2).

The framework is run for 1000 iteration for all the benchmarks for an error bounds of [15 dB, 25 dB, 35 dB, 45 dB] for PSNR and $[0.5\%, 1.0\%, 2.5\%, 5\%]$ of MREP for the *Gaussian Blur Filter* and other benchmarks, respectively. The experiments are performed on a system running a scientific Linux 7.2 (Nitrogen), comprising 16 nodes with an Intel® Xeon E5-2670 @ 2.6GHz and 256 Gigabytes of main memory.

V. Results

Figure 3 shows the normalized area results of the original, approximate, and approximate+infected of the benchmarks from Table I. The normalized area for both the *approximate* and *approximate+infected* versions obtained by the *MAAS* framework on each benchmark is reported and compared. The former represents a circuit in an approximated form where one or more components have been substituted with their approximate versions from the library but without any malicious logic. The latter represents the same configuration as the approximate version but with all approximate components containing malicious logic. The results are shown for different error bounds for each benchmark.

For *Ternary Sum* benchmark, the difference between the normalized area of *approximate* and *approximate+infected* is as small as 1.4% in case of 5% error bound and as large as 3.2% for 1% error bound. In the case of the *FIR Filter*, the minimum area difference remained 2.6% and the maximum area difference was 2.8%. For the benchmark *RGB2GRAY*, the difference could reach up to 3.9% while the minimum difference remains 3.2%. Finally, for the *Gaussian Blur Filter*, the *approximate+infected* version was 3.9% bigger than the *approximate* version. Overall, the difference between the two versions in terms of the hardware area varies from 1.4% to 3.9%. Since the approximation frameworks are typically based on stochastic search methods such as simulated annealing [2] and MCTS [1], they do not guarantee the same solution with every run of the framework, it is quite challenging to differentiate whether the area difference is due to the different search path of the design space resulting in a different approximate instance, or due to the malicious hardware hidden in the approximate instance. We, therefore, claim that the automated search-based approximation frameworks are prone to potential hardware Trojans insertion during the approximation process. Consequently, with such a small margin of area, they are quite hard to be identified during the post-processing of the flow.

Similarly, we report the power consumption results of the same benchmarks in Figure 4, in normalized form and again for various error bounds. Although the power consumption of a circuit mainly depends on the switching activity and the workload given, however, for a better demonstration of our attack, we also measure and compare the power consumption of approximate and approximate+infected designs. Moreover, since the underlying search algorithm is driven via stochastic sampling (i.e., MCTS), which might steer the design space exploration differently in each run, a slight variation of estimated area and power values is possible. To minimize this effect, we run the framework five times and then average the values.

Here, again we see a general trend of marginal difference between *approximate* and *approximate+infected* designs. For *Ternary Sum*, we have the smallest difference in terms of power consumption i.e., 0.1%, and even in case of 5% error bound, the *approximate+infected* design is 0.2% smaller than the *approximate*. For *FIR Filter*, the difference of power consumption ranges from a minimum of 1.8% up to a maximum of 2.0% and a similar trend can also be seen for *Gaussian Blur*, where the difference could reach up to 3.1% and the minimum difference remains 1.4%. However, in the case of *RGB2GRAY*, the difference in power consumption reaches up to 14% and even the minimum difference remains 5.2%.

While the overall trend of power consumption for both *approximate* and *approximate+infected* hints that the hardware Trojan insertion will go unnoticeable and is challenging to identify via power tracing tools. One of the benchmarks i.e., *RGB2GRAY* provides an interesting case where the power consumption of *approximate+infected* is considerably higher. One possible reason for this could be the hardware configuration of the benchmark which allows aggressive approximation of components, more specifically, multipliers. Since *RGB2GRAY* contains a relatively smaller number of components of which the majority are multipliers, the approximation of these multipliers with Trojan logic might cause higher switching activity and thus more power consumption. Nevertheless, we also see that the power consumption difference decreases with the increase in error bound. This might be caused by the harsh approximations due to larger error levy. However, this is out of scope of this paper and might be an interesting future work to investigate the effect of hardware Trojan infection in such circuits.

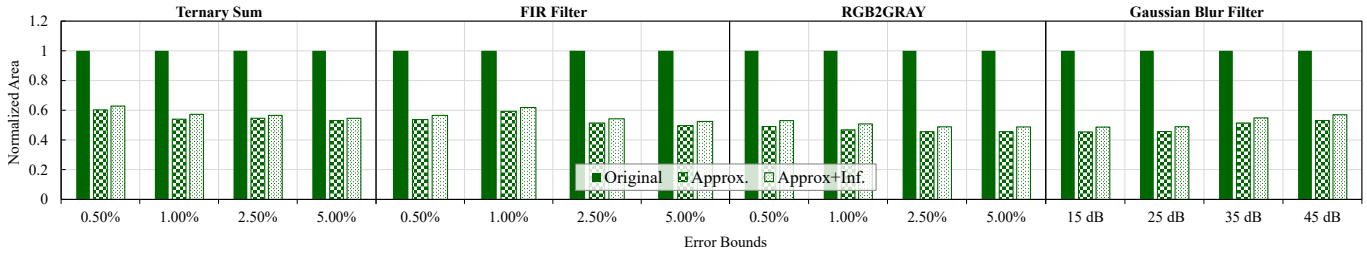

Fig. 3: Area results for original, approximate and approximate+infected of four most commonly used benchmarks

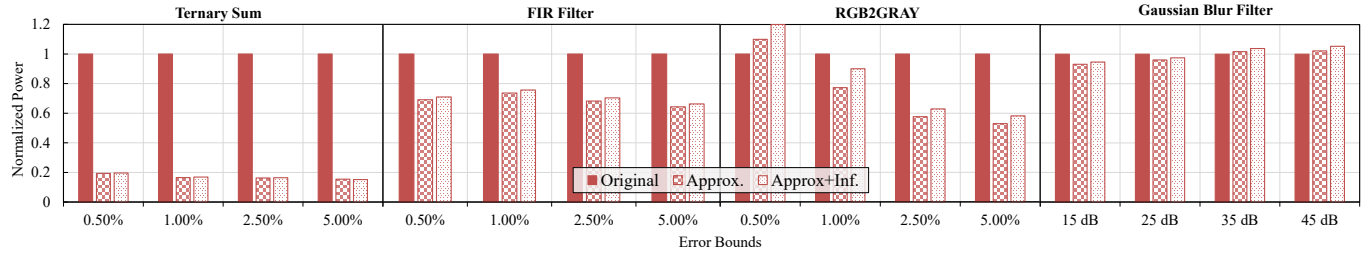

Fig. 4: Power results for original, approximate and approximate+infected of four most commonly used benchmarks

VI. Conclusion

This paper presents and demonstrates the threat model of hardware Trojan insertion in approximate accelerators through automated search-based synthesis framework. In particular, it presents *MAAS*, a framework that utilizes a search-based approximate accelerator synthesis flow and employs malicious library component substitution to generate Trojan inserted approximate versions of the accelerators. Essentially, *MAAS* exploits the approximation phase of the framework to substitute approximate+infected components thereby creating approximate versions that have infected components. The resultant approximate version could circumvents the testing/verification because the Trojan only triggers on a particular input sequence. Additionally, *MAAS* demonstrate the potential of hiding the hardware Trojan in the context of approximate computing/circuits and attempts to uncover new challenges for approximate computing. The vulnerability of approximate circuit synthesis in the wake of hardware Trojans puts them exploitable and additional steps could be required in the automated synthesis to identify malicious logic insertion during the process.

In the future, we will investigate our framework for larger-scale experiments, including newer and larger benchmark circuits with more hardware Trojan instances with multiple trigger conditions and varying payloads, such as modifying functionality and information leakage, for example, using side channels. Furthermore, we would like to investigate the accuracy versus efficiency trade-offs for original, approximate, and approximate+infected circuits. In addition to that, we would also like to explore the possible defense mechanism for the presented attack.

References

[1] M. Awais, H. G. Mohammadi and M. Platzner, "An MCTS-based Framework for Synthesis of Approximate Circuits," In IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC), 2018, pp. 219-224, doi: 10.1109/VLSI-SoC.2018.8645026.

[2] Linus Witschen, Muhammad Awais, Hassan Ghasemzadeh Mohammadi, Tobias Wiersema, Marco Platzner, "CIRCA: Towards a modular and extensible framework for approximate circuit generation," In Microelectronics Reliability, Volume 99, 2019, Pages 277-290, ISSN 0026-2714.

[3] Kumud Nepal et al., "ABACUS: A technique for automated behavioral synthesis of approximate computing circuits." 2014 Design, Automation & Test in Europe Conference & Exhibition (DATE). IEEE, 2014.

[4] Y. Jin, "EDA tools trust evaluation through security property proofs," 2014 Design, Automation & Test in Europe Conference & Exhibition (DATE), 2014, pp. 1-4, doi: 10.7873/DATE.2014.260.

[5] C. Krieg et al., "Malicious LUT: A stealthy FPGA Trojan injected and triggered by the design flow," in *ICCAD*, 2016, pp. 1–8. doi:https://doi.org/10.1145/2966986.2967054

[6] Q. A. Ahmed et al., "Malicious Routing: Circumventing Bitstream-level Verification for FPGAs", in *DATE 2021* (Virtual Conference, Feb. 1–5, 2021). IEEE, Feb. 2021, pp. 1490–1495.

[7] Q. A. Ahmed et al., "Proof-Carrying Hardware Versus the Stealthy Malicious LUT Hardware Trojan," in *(ARC) 2019*. Lecture Notes in Computer Science, vol 11444. Springer, Cham. [Online]. Available: https://doi.org/10.1007/978-3-030-17227-5.

[8] F. Wolff et al., "Towards Trojan-free trusted ics: Problem analysis and detection scheme," in *2008 Design Automation and Test in Europe*, March 2008, pp. 1362–1365.

[9] R. S. Chakraborty et al., "Hardware Trojan: Threats and emerging solutions," in *2009 IEEE International High Level Design Validation and Test Workshop*, Nov 2009, pp. 166–171.

[10] S. Bhunia et al., "Hardware Trojan Attacks: Threat analysis and countermeasures," in *Proceedings of the IEEE* 102(8), 1229-1247 (Aug 2014).

[11] F. Regazzoni, C. Alippi and I. Polian, "Security: The Dark Side of Approximate Computing?,' In IEEE/ACM International Conference on Computer-Aided Design (ICCAD), 2018, pp. 1-6.

[12] P. Yellu, N. Boskov, M. A. Kinsy, and Q Yu., "Security Threats in Approximate Computing Systems," In Proceedings of the 2019 on Great Lakes Symposium on VLSI (GLSVLSI '19), 2019, Association for Computing Machinery, New York, NY, USA, 387–392.

[13] Ariful Islam, S., "On the (In)security of Approximate Computing Synthesis," arXiv e-prints, 2019.

[14] Ye Wang, Jian Dong, Qian Xu, Zhaojun Lu, and Gang Qu., "Is It Approximate Computing or Malicious Computing?," In Proceedings of the 2020 on Great Lakes Symposium on VLSI (GLSVLSI '20), 2020, Association for Computing Machinery, New York, NY, USA, 333–338.

[15] Yuqin Dou, Shichao Yu, Chongyan Gu, Maire O'Neill, Chenghua Wang, and Weiqiang Liu., "Security Analysis of Hardware Trojans on Approximate Circuits," In Proceedings of the 2020 on Great Lakes Symposium on VLSI (GLSVLSI '20), 2020, Association for Computing Machinery, New York, NY, USA, 315–320.

[16] Muhammad Awais, Hassan Ghasemzadeh Mohammadi, and Marco Platzner. "LDAX: A Learning-based Fast Design Space Exploration Framework for Approximate Circuit Synthesis." In proceedings of the ACM Great Lakes Symposium on VLSI. 2021.

[17] V. Mrazek et al., "Evoapprox8b: Library of Approximate Adders and Multipliers for Circuit Design and Benchmarking of Approximation Methods," in IEEE DATE, 2017.