2023 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT) | 979-8-3503-1500-4/23/$31.00 ©2023 IEEE | DOI: 10.1109/DFT59622.2023.10313548

EnSAFe: Enabling Sustainable SoC Security Auditing using eFPGA-based Accelerators

Mridha Md Mashahedur Rahman, Shams Tarek, Kimia Zamiri Azar, and Farimah Farahmandi
University of Florida {mrahman1, shams.tarek, k.zamiriazar}@ufl.edu, farimah@ece.ufl.edu

***Abstract*—The utilization of reconfigurable and flexible acceleration for compute-intensive kernels, e.g., neural networks, crypto-engines, and arithmetics, have been on the rise in recent years, where embedded FPGA (eFPGA) architecture, as a *de facto* solution, is becoming an integral component of the system-on-chip (SoC) for the acceleration purposes. In the meantime, modern SoCs are getting larger and more complex in size and functionality, and with the inclusion of more IPs along with the eFPGA accelerator, they are getting more exposed to a wide variety of security-critical and sensitive information, hence innovative security infrastructure is needed to ensure the information security, i.e., *integrity*, *confidentiality*, and *availability*. Also, in an SoC equipped with the eFPGA-based accelerator, with the potential reconfigurability over time, sustainable and upgradable security infrastructure becomes a necessity. To address such concerns, in this paper, we introduce *EnSAFe*, a framework for enabling security policy auditing with upgradability within the designs enabled by eFPGA-based accelerators. The *EnSAFe* framework enables signal monitoring in a plug-and-play fashion, and the monitoring core logic will be mapped onto the eFPGA accelerator component with minimal overhead (almost zero). Our experiments demonstrate that the *EnSAFe* framework can detect runtime threats/vulnerabilities with a low area overhead.**

***Index Terms*—SoC Security Monitoring, Upgradability, eFPGA, Security Policy**

I. Introduction

With the ever-increasing shrinkage and integration density of modern semiconductor technologies, modern system-on-chips (SoCs) are getting larger and more complex with tens to hundreds of intellectual properties (IPs). Additionally, to bring versatility, flexibility, and efficiency into the SoCs with denser and shrunk technology nodes, recent studies show interest in investigating the possibility of integrating reasonably sized FPGA arrays, so-called embedded FPGA (eFPGA), for a wide variety of applications. In [1], Arnold SoC is presented, an SoC with an eFPGA, for flexible power-constraints energy-efficient IoT devices. It shows how the eFPGA can be used to extend the SoC peripheral subsystem, and accelerate the computations by offloading the CPU. Similarly, in [2], a reconfigurable SoC for smart power applications has been implemented, which shows that the eFPGA, despite its non-negligible area overhead, proves to be an excellent solution in terms of both energy efficiency and latency.

The utilization of eFPGA architectures is not only limited to acceleration and energy efficiency applications. Few recent studies have also discovered getting the benefit of reconfigurability of eFPGA for IC supply chain security threats, e.g., reverse engineering, IP piracy, and IC overproduction. In [3], hardware redaction using eFPGA has been introduced for IP protection, in which the bitstream of the eFPGA serves as the secret to being against IP piracy, IC overproduction, and reverse engineering. Fig. 1 is an example, where the eFPGA fabric is used for different purposes. Fig. 1(a) shows the overall usage of eFPGA in Arnold SoC [1], in which the eFPGA is mainly for acceleration with higher energy efficiency. On the contrary, Fig. 1(b) is when the eFPGA is replaced with a security-critical IP (or module) to be against reverse engineering, in which since performance improvement is not a concern, the eFPGA fabric is constructed only from logic cells (no DSP/memory).

In the meantime, as the SoC market continues to grow, rigid time-to-market (TTM) is becoming increasingly common, and third parties IPs (3PIPs) are becoming more and more reused, resulting in the introduction of multiple untrustworthy entities into the supply chain. Consequently, there is an increasing risk of introducing more vulnerabilities in the SoC architecture that may cause irreparable damage [4]. Hence, a security monitoring and verification engine are a must for today's SoCs [5], which is becoming more and more challenging as SoC complexity continues to grow and time-to-market shrinks. This security-related issue gets worse with the integration of eFPGAs. Although the eFPGA fabric integrated into the SoC can significantly improve the versatility and efficiency of the system, its reconfigurability may open new backdoors for attackers. Additionally, the upgradability may affect the whole system (from hardware to software), which may result in emerging new threats applicable to the whole system. None of the existing eFPGA solutions has tackled this issue to see how the eFPGA can be used to enhance the security of modern SoCs, especially when the system is in the field (against zero-day attack[1]).

In recent years, numerous studies focused on the potential solutions for SoC security monitoring, verification, and validation against known attacks [6]–[9]. However, none of these solutions involve the dynamicity and reconfigurability of the SoC as part of the problem description, and hence, they are all static, and yet inefficient against the zero-day and particularly new emerging attacks. As a result, we introduced *EnSAFe*,

[1]For a newly discovered vulnerability, developing a solution and installing it often takes some time. In the meantime, an increasing number of attacks are performed to exploit this vulnerability, which is known as a zero-day attack.

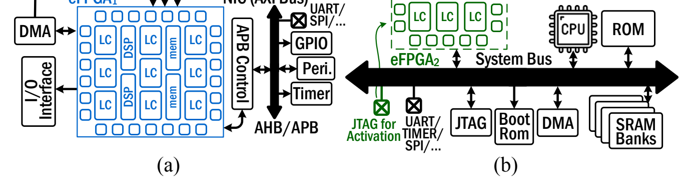


Fig. 1: The Engagement of eFPGA Fabric for (a) Acceleration and Energy Efficiency, (b) IP redaction against Reverse Engineering.

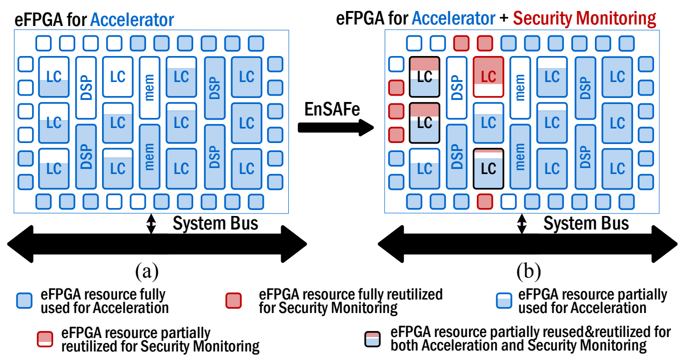

eFPGA for Accelerator
eFPGA for Accelerator + Security Monitoring
EnSAFe
LC
DSP
mem
System Bus
(a)
(b)
eFPGA resource fully used for Acceleration
eFPGA resource fully reutilized for Security Monitoring
eFPGA resource partially used for Acceleration
eFPGA resource partially reutilized for Security Monitoring
eFPGA resource partially reused&reutilized for both Acceleration and Security Monitoring

Fig. 2: Re-use of Unutilized Resource for (Decoupled) Security Monitoring.

a reconfigurable security policy checking engine[2], which is dynamically upgradable in the field so that the SoC designs can be (sustainably) protected from emerging and unforeseen threats. The *EnSAFe* framework has been built based on the fact that the utilization of eFPGA fabrics is not even close to 100%, meaning that a non-negligible part of the eFPGA fabric is always unutilized. Table I shows some of the reported numbers by the existing eFPGA automation tools, e.g., OpenFPGA [10], in terms of utilization, which is surprisingly low (in terms of unused LUTs). Hence, as shown in Fig. 2, the *EnSAFe* framework enables this automation tools to add an independent and separated security monitoring engine by reusing the unutilized resources of the existing fabric at a negligible (almost zero) overhead. The main contributions of this paper are as follows:
(1) The proposed *EnSAFe* is an extended eFPGA automation flow, in which two fully independent applications (one for acceleration/efficiency and one for security monitoring) will be mapped to the existing eFPGA fabric.
(2) We propose a plug-and-play observation deck that is IP-specific but SoC-level, known as the security status monitor (SSM), for collecting (status of) security-critical implications that utilize minimal I/O pins. It allows them to connect to the eFPGA fabric via the unutilized I/O pins.
(3) With SSMs, we present a reconfigurable security policy checking engine that will be mapped onto the unutilized eFPGA resources next to the main application (acceleration/efficiency).
(4) We investigate the usability and adaptability of the *EnSAFe* framework on a RISC-V-based SoC tailored with few vulnerabilities. We examine the security provided by the proposed framework as well as the efficiency (overhead).

II. Background and Prior Art

A. Requirements of SoC-level Security Verification

In the modern SoCs following the globalized supply chain, with the inclusion of numerous IPs from different vendors, each IP may have a set of information/data, called *security assets*, whose leakage leads to financial/reputation loss. Hence, recent studies and implementations try to define a set of requirements, known as *security policies* [11], [12], which must be met to ensure the information security, i.e., integrity, confidentiality, and availability [13]. These security policies can be categorized into some major breeds, including (i) Policies for Access restriction that define how components can access security assets of other ones; (ii) Policies for data/control flow restriction that regulate components behavior in response to an event/inquiry, and how the response will propagate; (iii) Policies for HALT/Denial-of-Service that indicate the liveness of components runtime; and (iv) Policies for insecure sequence of execution that check correct sequence of security-oriented actions (e.g., correct authorization). As SoCs get more complex and bigger, the definition of these policies is becoming more challenging. Additionally, the inclusion of eFPGA, which enables the reconfigurability and support of a variety of applications at runtime (Fig. 1(a)), has increased the complexity of security monitoring and verification.

B. Prior Art: SoC-level Security Monitoring

Over the last two decades, hardware-oriented security solutions and countermeasures have advanced tremendously. However, a significant portion of them heavily focuses on the IP-level security countermeasures, e.g., hardware obfuscation, watermarking, metering, camouflaging, side-channel analysis, etc. [14]–[17], which are not applicable for SoC-level vulnerabilities, as many of the SoC-level vulnerabilities are inter-IP or inter-component issues. Recent studies, however, show a shift to presenting SoC-level solutions, and in the case of SoC-level monitoring[3], following are the notable ones:

Some studies, e.g., [6], [7], attempt to implement a centralized control engine for implementing SoC security policies. In this case, each IP must be augmented with a wrapper customized for security policies for security-relevant events, communications, and data of the IP and coordinate with the centralized controller to ensure policy adherence. However, this custom wrapper is a fixed/static architecture and fails to offer in-field upgradability of policy implementations after deployment. Additionally, the types of policies covered in this breed require heavy intra-IP information with expert knowledge, which is rarely the case.

Similarly, few studies, e.g., [8], [9] follow design-for-debug (DfD) infrastructure to monitor intra-IP signals and forward captured events to a centralized policy engine to detect SoC violations. However, relying on the debug infrastructure to observe the internal signals requires addressing many challenges (e.g., Re-purposing the DfD should not interfere with debug usage of the system). Further, to adapt to this DfD-security architecture, the IP provider needs to map security-critical events to the DfD instrumentation for the IP, which is a very hard assumption in modern SoCs with numerous 3PIPs.

The work in [18] identifies common SoC security vulnerabilities by analyzing the design. Then to monitor these vulnerabilities, the authors define several classes of assertions and inserted them into the SoC design to enable runtime checking of security vulnerabilities. The assertions, however, are also for static systems and cannot be updated in-field, leaving the SoC susceptible to emerging security threats.

[2]*EnSAFe* is designed and dedicated for only eFPGA-equipped SoCs, whose eFPGA is customized for either acceleration or energy efficiency.

[3]We only focus on SoC-level monitoring prior works as the main focus of this paper is to introduce an SoC-level security monitoring engine.

TABLE I: eFPGA Utilization Ratio in OpenFPGA Framework [10]

Benchmark	# of Input Pins	# of Output Pins	LUTs Needed	eFPGA Size	Utilization Ratio (%)	Unutilized LUT
(4-64)-bit Counter	2	4-64	114	4×4	89%	14
Apex2	24-48	4	613	11×11	84%	163
SHA256	16-32	8-24	2845	20×20	88%	355
Elliptic Core	96-192	96-128	4064	28×28	64%	2208

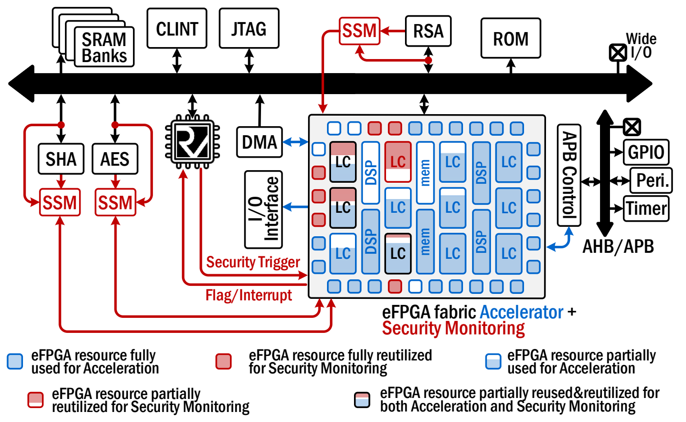

SRAM Banks
CLINT
JTAG
SSM
RSA
ROM
Wide I/O
SHA
AES
SSM
SSM
DMA
I/O Interface
Security Trigger
Flag/Interrupt
LC
DSP
mem
APB Control
GPIO
Peri.
Timer
AHB/APB
eFPGA fabric Accelerator + Security Monitoring
eFPGA resource fully used for Acceleration
eFPGA resource fully reutilized for Security Monitoring
eFPGA resource partially used for Acceleration
eFPGA resource partially reutilized for Security Monitoring
eFPGA resource partially reused&reutilized for both Acceleration and Security Monitoring

Fig. 3: Security Monitoring via the eFPGA and Security Status Monitor (SSM).

III. Proposed Scheme: EnSAFe

As shown in Table I, when the circuit size targeted to be mapped into an eFPGA fabric is getting larger, the fabric size will explode, and the utilization ratio decreases significantly. Further, as the utilization of larger eFPGA fabrics decreases, the number of resources available (unutilized) increases. For instance, in a 28×28 eFPGA fabric, there are more than 2K LUTs unutilized, while only 14 LUTs are unutilized in a 4×4 eFPGA fabric. Since eFPGAs will be mostly used for acceleration purposes (e.g., to model a complex compute-intensive kernel migrated from the CPU to the eFPGA), which requires a large fabric, it opens the possibility of reutilizing unused resources for another (non-computation-intensive) application. So, *EnSAFe* enables the security monitoring of the entire system in a dynamic manner on these unutilized resources (Fig. 2), which is typically not a resource-intensive application and can be fit on the unutilized part. Fig. 3 demonstrates the outcome of *EnSAFe* in a RISC-V-based SoC architecture. As shown, it consists of two main components: (1) eFPGA for acceleration/security, which realizes both acceleration and dynamic monitoring together, and (2) security status monitor (SSM), which is responsible for collecting required security-critical implications in a compressed manner. Enabling the security monitoring on eFPGA fabric allows us to upgrade the security policies in-field at runtime by reprogramming the eFPGA, which helps address new vulnerabilities or to prevent zero-day attacks.

A. *eFPGA for Acceleration+Security*

In this model, the eFPGA is assumed to be connected to the system bus, as it will perform hardware acceleration. So, the eFPGA can directly monitor the system bus transactions. It also enables the CPU to configure and communicate with the eFPGA when necessary (triggering). In this case, based on the policy, the eFPGA will be triggered to monitor the system bus for a specific time of operation. The CPU sends configuration bits to the eFPGA, which will be used as the trigger to start/stop monitoring. In *EnSAFe*, we use an enforced memory mapping in eFPGA so that no unauthorized IP can access the eFPGA at any time during the SoC lifecycle[4].

B. *Security Status Monitor (SSM)*

To complete the monitoring procedure, we propose a new monitoring module, called SSM. The SSMs, as shown in Fig. 3, are added per IP (security-critical ones), and they follow a very generic definition to minimize the customization required per IP[5]. The SSM concentrates mostly on inter-IP transactions. The SSMs will also take care of a limited amount of intra-IP signaling, which is difficult to achieve by only observing the system bus transactions. This, in turn, simplifies the policy implementation and reduces resource utilization of the eFPGA. Based on the generic specification of the IP (main behavior, timing diagram, handshaking protocol, etc.), they filter out and store critical data of interest. Then, this data will be sent in a compressed format to the eFPGA (if triggered for monitoring). The compression will be accomplished using a pre-defined protocol per IP (e.g., compressed AXI (AXI4-Lite) signaling, function, and timing description). SSMs are a static part of the SoC, and all the security policies in EnSAFe are written based on the information available through the SSMs (and main bus). Although it seems that having SSMs static may limit the reconfigurability, our case studies show these signals can cover a wide range of vulnerabilities.

Fig. 4 shows an overall view of the SSM architecture. SSMs have an identical connection to IPs (As IPs use the same handshaking, e.g., AXI). Each SSM is connected to the eFPGA fabric directly[6] through dedicated (bidirectional) wiring (to unutilized I/O pins). SSMs consist of a limited number of buffers to temporarily store IP-related information/data based on the event detection logic output. The events are defined based on the I/O values of IPs and their generic specifications. The event detection logic looks for these events or event sequences in the IO signal values (address, data, valid, ready, etc.). The event detection logic consists of a set of configurable registers, comparator(s), and tiny FSM(s). If the event to be detected occurs only in one clock cycle and does not depend on any sub-events, we do not need an FSM in the event detection logic. In this case, a simple comparator will suffice. On the other hand, if the event to be detected has precedent sub-events that occur over multiple clock cycles, we use a tiny FSM to detect such events. The tiny FSM(s) use configurable register contents to determine which event/value to look for in the IO signals. Thus, the event detection logic is implemented in a commandable manner so that the eFPGA can send various commands for detecting multiple events. A free up-counter is used in SSMs for building the timestamp for recording the events. The SSMs will monitor and collect data with timestamps so that the eFPGA will be able to tell at which time the vulnerability occurred. The security policies implemented within the eFPGA use the event information from the SSMs to detect security vulnerabilities.

C. *Overall Flow of the* EnSAFe *Framework*

Fig. 5 shows the overall flow of the *EnSAFe* framework. As shown, after generating the eFPGA fabric for the accelerator (based on the targeted eFPGA architecture), resource evaluation has been done to check the availability of unutilized resources. Afterward, based on the availability of unutilized resources, security-critical IP specifications, and potential threat models, we create security policies and SSM sub-circuits. The SSMs'

[4]Since multiple components may be authorized to engage eFPGA for applications, *EnSAFe* regulates them by defining specific security policies.

[5]SSMs resemble performance monitors used in today's modern SoCs [19].

[6]As there will be multiple SSMs in the SoC, each for a security-critical IP, the communication between the eFPGA and the SSMs through a shared private bus may create a bottleneck. Since eFPGA for acceleration occupies a significant portion of die size, our experiments shows slight delay for routing dedicated wires between IPs' SSMs and eFPGA.

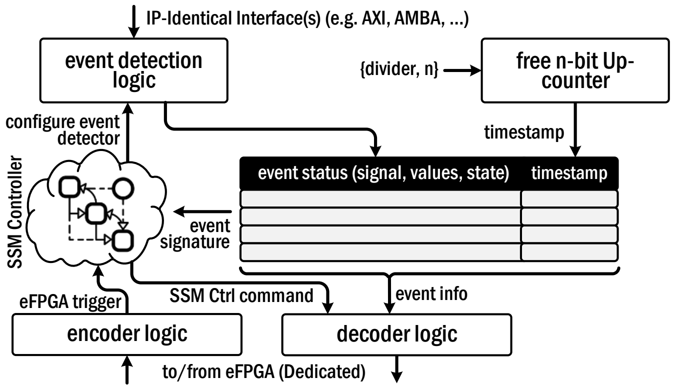


Fig. 4: Architecture of a Security Status Monitor (SSM).

sub-circuits will be integrated back into the SoC, and the policies' sub-circuit will be integrated with the accelerator for one more time mapping on eFPGA fabric.

If the routing and placement algorithms used in the eFPGA automation tool (VTR/VPR) fail to remap the accelerator and security policies together, the *EnSAFe* framework should reduce the logic size. There exist two valid options for the designer in this case: (1) back to re-designing the accelerator part (if the security is the priority), and (2) back to a relaxed selection of policies (if the performance/throughput is the priority). Although possible, our experiments show that for the case studies investigated so far via the *EnSAFe* framework, no failure happened for remapping.

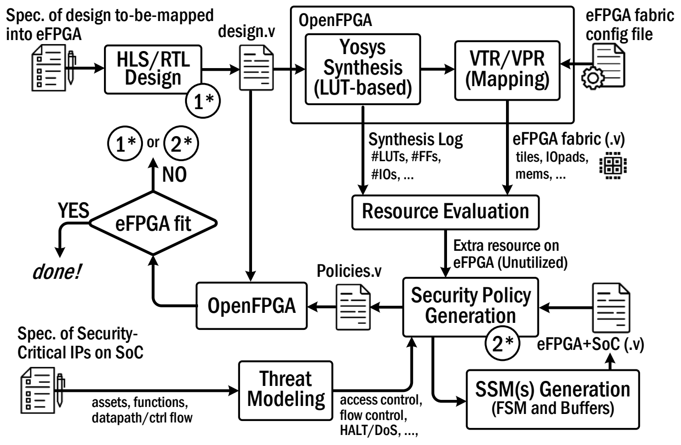
Furthermore, as shown in Fig. 5, the security policies (to monitor security-critical transactions of various IPs inside the SoC) are based on the threat models and assets associated with security-critical IPs. Threat models are carefully selected based on the nature of vulnerabilities (per IP) and their impact on the SoC. Security policies use the data collected from the SSMs and system bus monitoring to detect security vulnerabilities. We implemented the security policies in synthesizable HDL, primarily as tiny FSMs, which can be mapped into the eFPGA fabric and updated as needed (in the field).

Once the security policies are mapped into the eFPGA and the SSMs are added and connected for collecting data, in case of any security policy violation, either the eFPGA or the CPU can take care of the policy enforcement. However, to keep resource utilization for the security portion at a minimum, we have considered the CPU as the enforcement entity. So, as shown in Fig. 3, depending on the nature of the policy, the eFPGA raises a flag or sends interrupts to the CPU. The CPU can take any necessary actions from there, depending on the severity of the violation. These enforcement actions include but are not limited to stopping execution of the current process, blocking transactions from/to suspect IP, and sending 'disable' configuration bits to an IP to isolate it from the SoC.

IV. Case Studies and Experimental Results

A. Experimental Setup

To assess the performance and efficiency of the proposed *EnSAFe* framework, we targeted Ariane SoC [20] to be enabled with both acceleration and security monitoring. Ariane SoC consists of a 64-bit, 6-stage, in-order RISC-V processor and several (peripheral) IPs like UART, SPI, Ethernet, GPIO, etc., all connected using AXI crossbar. We augmented the SoC by adding a 128-bit Advanced Encryption Standard (AES) crypto



Spec. of design to-be-mapped into eFPGA
design.v
OpenFPGA
eFPGA fabric config file
HLS/RTL Design
Yosys Synthesis (LUT-based)
VTR/VPR (Mapping)
1*
1* or 2*
NO
Synthesis Log #LUTs, #FFs, #IOs, ...
eFPGA fabric (.v) tiles, IOpads, mems, ...
YES
eFPGA fit
Resource Evaluation
done!
Extra resource on eFPGA (Unutilized)
Policies.v
OpenFPGA
Security Policy Generation
2*
eFPGA+SoC (.v)
Spec. of Security-Critical IPs on SoC
Threat Modeling
SSM(s) Generation (FSM and Buffers)
assets, functions, datapath/ctrl flow
access control, flow control, HALT/DoS, ...,

Fig. 5: Overall Flow of the EnSAFe Framework.

module and a 64-bit Direct Memory Access (DMA) module. Additionally, to have a fair comparison, for the acceleration, we inserted an eFPGA with approximately equal size to the eFPGA used in Arnold SoC [1]. For building the eFPGA for the integration, we engaged OpenFPGA framework [10], which provides a complete Verilog to bitstream generation flow with customizable eFPGA architectures (suited for the design to be mapped). Using the framework summarily shown in Fig. 6, the OpenFPGA framework creates a tightly organized set of programmable tiles, block RAMs, DSP blocks (if needed), and I/O banks. We integrated the generated eFPGA fabric into the Ariane SoC through the system AXI4 crossbar. Additionally, for SSMs, direct connections to (unutilized) I/O banks are added. For overhead comparison, all experiments went through Synopsys Design Compiler using open Skywater 130nm process [21]. Also, the test of the security monitoring scenarios has been done using Vivado (for synthesis and integration) and OpenOCD (JTAG-enabled debugging). Further, the test are accomplished (with functional verification) using an emulation model on Genesys 2 Kintex-7 FPGA Development Board [22].

B. Use Case Scenario

We have considered three use case scenarios for eFPGA-based security monitoring proof of concept. They involve three different vulnerabilities and their detection approach using the proposed eFPGA-based monitoring architecture. Table II lists and describes these targeted vulnerabilities.
(Vulnerability 1) Rowhammer Attack: Rowhammer attack [23] is performed by repeated reading/writing and recharging of a DRAM row. This attack targets the security assets that have been stored in RAM banks during the execution of different applications on SoC. The repeat of read/write will redirect

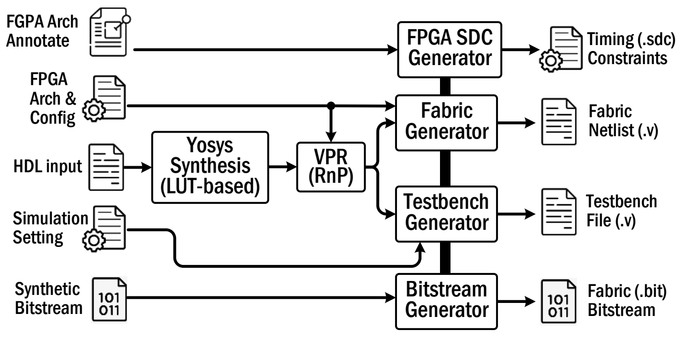


Fig. 6: Custom eFPGA design flow based on OpenFPGA [10].

TABLE II: Vulnerabilities Targeted for Monitoring in EnSAFe.

Vulnerability	Description	Category	Security Objective	Security Policy Monitor	SSM Event	FPGA-based Resource Needed for the Policy
Rowhammer Attack	Repeated reading/writing and recharging of a DRAM row for illegal access to security asset(s) stored in RAM	CWE-1260	Confidentiality	Repeated rd/wr req should not exceed a threshold within a limited time	DDR3 controller interface watch	61 LUTs + 68 FFs
AES Information Leakage	A Trojan leaks the AES secret key through the common bus in the SoC (Triggering with a specific plaintext)	AES-T1300, CVE-2018-8933, CVE-2014-0881	Confidentiality	No cipher key (partial or full) should be visible through the primary output	Cipher key and Ciphertext Observation	46 LUTs + 2 FFs
AES Denial-of-Service	A Trojan tracks incoming patterns (plaintext) and HALT will be occurred once a specific pattern is observed.	AES-T500, BASICRSA-T200, BASICRSA-T400	Availability	the valid_out signal must become high after n number of clock-cycle (once plaintext is refreshing).	AES FSM watch (counter-based)	9 LUTs + 10 FFs
DRAM Access Control Violation	A malicious IP tries to illegally access the secure DRAM region	CWE-1257, CWE-1190, CVE-2022-37302	Confidentiality	Identify illegal access to the secure memory region by monitoring read/write requests	DDR3 controller interface watch	N/A

access to different rows of the RAM on the same bank, and allowing the adversary to catch values not requested for (illegal access). To model this attack, we defined the security policy as: **repeated read/write accesses should not exceed a certain threshold within a limited time**. In this case, SSM dedicated to RAM (DDR3) will monitor DRAM bus continuously to find high-frequency read/write (count of read/write + timestamp). The event detection logic is parameterized for the number of memory banks, threshold value for counting, and time window (clock cycles) and can look for the rowhammer attack simultaneously in multiple banks (up to 8 banks). The threshold value corresponds to a physical attribute of the DRAM which defines the minimum frequency of row buffer recharge that may result in rowhammer effect. We experimentally chose this value for the DRAM during policy implementation. However, the threshold value does not have any cost impacts on the SSM. In the following snippet, the security policy for rowhammer detection is formulated at a high level.

```
if time_window_start
    count = 0;
else if read/write_addr_within_bank
    count = count + 1;
if count > threshold
    interrupt = 1;
```

(Vulnerability 2) AES Information Leakage: This vulnerability is Trojan-enabled, and when triggered, the cipher key or parts of the cipher key will be leaked through the primary outputs (within the ciphertext). In this specific Trojan, the trigger for this Trojan is the specific pattern in the plain text, and the payload is the cipher key leakage. The SSM dedicated to the AES module will check the AES data, including plaintext, cipher key, and the ciphertext for catching this vulnerability[7]. By sending the data collected and encoded in the SSM, the policy added into the eFPGA will detect whether the Trojan is triggered or not. In this case, the policy states that **no cipher key (or its parts) should be visible through the primary output** (shown in the following snippet).

```
for all ciphertext_bytes do
    if ciphertext_byte = cipherkey_byte
        interrupt = 1;
```

(Vulnerability 3) AES Denial of Service: Similar to vulnerability 2, this vulnerability is also Trojan-enabled, in which the Trojan is triggered by a sequence of specific patterns in the plaintext. If the trigger matches, the whole process goes into a denial of service (DoS). AES module cannot generate a *valid_out* signal while in a DoS situation (HALT in operation). In this case, the policy is that **the valid_out signal must become high after n number of clock-cycle**, where n is the number of clock-cycle needed to complete the encryption (high-level showin in the following snippet).

[7]As the cipher key is a security-critical entity, comparing the output with each byte of the key is performed inside the SSM.

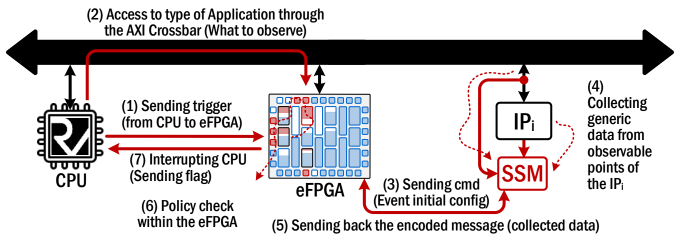

(2) Access to type of Application through the AXI Crossbar (What to observe)
(1) Sending trigger (from CPU to eFPGA)
(7) Interrupting CPU (Sending flag)
CPU
eFPGA
(6) Policy check within the eFPGA
(3) Sending cmd (Event initial config)
(4) Collecting generic data from observable points of the IPi
IPi
SSM
(5) Sending back the encoded message (collected data)

Fig. 7: Steps to detect vulnerability in the SoC via *EnSAFe*.

```
for all clock_cycles do
    if valid_input
        count = 0;
    else if !valid_out
        count = count+1;
    if count > n
        interrupt = 1;
```

In all use case scenarios, the *EnSAFe* framework follows identical procedure. Fig. 7 shows the overall steps of detecting the vulnerabilities/threats. Per each use case scenario, the CPU has the right to send a trigger signal to the eFPGA to start monitoring. A set of commands will be sent to the SSM from eFPGA that determines which event to be monitored. The SSM gathers all the signals into the event tracker buffers and sends the encoded data back to the eFPGA. Then, the eFPGA, with minimal logic resources, will complete the monitoring for detecting potential vulnerabilities.

C. Experimental Evaluation

As discussed previously (Table I), for larger eFPGAs, more unutilized resources are available. Table II shows what resources are required per each vulnerability to implement the policy into the eFPGA. As shown (in the last column), only tens (10-70) LUTs/FFs are required per policy. Our security policies (which behave like synthesizable assertions) can easily fit into the unutilized resources. With this low overhead, we can develop a more comprehensive security policy engine that monitors a wide range of policies within the eFPGA. Table III provides a comparative illustration between the resources required for modules mapped to the eFPGA in Arnold SoC vs. that of security policies. The resources needed for acceleration are more than tenfold. This is why the eFPGA fabric for accelerators is huge with a low utilization rate, and it allows us to reuse them with no overhead for security purposes. Additionally, as listed in Table III, the size of fabric needed for each application (separately) reveals how security portion could be small compared to the acceleration part(s). For instance, for Arnold (with more than 6K LUTs, 4K FFs, and DSPs), the

TABLE III: Resource Utilization (post-Synthesis) for Implementing Different Use Cases (Acceleration vs. Security) on the eFPGA Fabric

Resources	Acceleration Use Cases					Security Policies		
	Custom I/O	Wide I/O	BNN	CRC	Elliptic	V1	V2	V3
GPIO	72	192	384	64	192	18^{*1}		
LUT	355	565	1465	122	4064	61	46	9
FF	240	550	935	41	2122	68	2	10
eFPGA size	$32\times32^{*2}$				28×28	$2\times2^{*3}$		

^{*1}: 18 is the total IO needed for all Vulnerabilities (together). V_i: Vulnerability i
^{*2}: 32×32 is the eFPGA fabric for all I/Os, BNN, and CRC.
^{*3}: 2×2 is the eFPGA fabric for all security policies together.

TABLE IV: Area Distribution of Some Main Components of the SoC.

Module	Area [μm^2]	Cell Count	Normalized Ratio (w.r.t CPU)
CVA6 (CPU)	1,734,757	216,922	1
eFPGA for Sec. Policy	380,903	42,424	0.219
eFPGA for Acceleration	79,715,267	11,085,611	46.004
AES Core	902,325	110,375	0.520
SSM (3)	656,750	85,154	0.37
Direct Memory Access	167,780	13,572	0.096
SPI Peripherial	12,827	1,269	0.007

fabric size becomes 32×32, while for the security policies of Table II, the eFPGA fabric size is only 2×2, which shows a ratio of 256:1. Future work will expand the list of use cases to demonstrate that even the inclusion of more security policies does not impact the ratio significantly.

Table IV shows a simple breakdown of some RISC-V Ariane SoC modules' area overhead. In this experiment, a dedicated eFPGA fabric per application (one for security and one for accelerator) is implemented, integrated, and tested. All the ratios are normalized w.r.t. the size of the CPU to provide a big picture of eFPGAs' sizes. As shown, the eFPGA for acceleration is almost 46x bigger than the CPU alone. This is consistent with the output of Arnold SoC [1], where eFPGA occupies $\sim$80% of the die size. However, the eFPGA used for the security policies of targeted vulnerabilities is almost 5x smaller than the CPU. This, again, confirms that once the eFPGA fabric is in place, in which the utilization is also low, the unutilized part can be reused for other non-compute-intensive applications, like security monitoring, which has almost no impact on the overhead. Additionally, the area overhead of the SSM reported in Table IV shows that adding these monitoring modules per each IP (based on the inter-IP transactions and important intra-signaling) does not incur considerable overhead. The main part of the SSMs is the event buffer and tuning this part (compressing and decompressing) affects the overhead significantly. However, for the targeted vulnerabilities, based on the size and the depth of buffer used for detecting the vulnerabilities, the overhead is kept reasonable. In this experiment, we implemented security monitoring for the three vulnerabilities (rows 1-3) in table II. When the chip is fabricated and in-field, there could be unforeseen vulnerabilities. To address these unforeseen vulnerabilities, we have to reprogram the eFPGA with a revised security policy. Row 4 in the table II denotes such an unforeseen vulnerability where a malicious IP tries to access the secure memory region in the DRAM. The SSM used for rowhammer detection monitors the DRAM interface and the eFPGA is connected to the system bus (Fig. 7). Therefore, we can reprogram the eFPGA with a new security policy that uses the same SSM to address this access control violation.

V. Conclusion

Relying on the fact that large-size eFPGA fabrics used for acceleration/efficiency in modern SoCs suffer from low utilization, in this paper, we introduced *EnSAFe*, a framework for security monitoring of the SoCs' vulnerabilities with upgradability over time, which is the most suited solution for zero-day attacks. *EnSAFe* benefits from the unutilized resources of already-integrated eFPGA and enables the designers to support both acceleration and security monitoring (with upgradability) on the same fabric with almost no overhead.

References

[1] P. S. *et al.*, "Arnold: An eFPGA-augmented RISC-V SoC for flexible and low-power IoT end nodes."

[2] F. Renzini *et al.*, "A fully programmable eFPGA-augmented SoC for smart power applications," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 67, no. 2, pp. 489–501, 2019.

[3] P. Mohan *et al.*, "Hardware Redaction via Designer-Directed Fine-grained eFPGA Insertion," in *Design, Automation & Test in Europe Conf. & Exhibition (DATE)*, 2021, pp. 1186–1191.

[4] P. Kocher *et al.*, "Spectre attacks: Exploiting speculative execution," in *IEEE S&P*, 2019, pp. 1–19.

[5] K. Z. Azar *et al.*, "Fuzz, Penetration, and AI Testing for SoC Security Verification: Challenges and Solutions," *Cryptology ePrint Archive*, 2022.

[6] X. Wang *et al.*, "IIPS: Infrastructure IP for secure SoC design," *IEEE Transactions on Computers*, vol. 64, no. 8, pp. 2226–2238, 2014.

[7] A. Basak *et al.*, "A flexible architecture for systematic implementation of SoC security policies," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2015, pp. 536–543.

[8] A. Basak *et al.*, "Exploiting design-for-debug for flexible SoC security architecture," in *ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2016, pp. 1–6.

[9] A. Nath *et al.*, "System-on-chip security architecture and CAD framework for hardware patch," in *Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2018, pp. 733–738.

[10] X. Tang *et al.*, "OpenFPGA: An open-source framework for agile prototyping customizable FPGAs," *IEEE Micro*, vol. 40, no. 4, pp. 41–48, 2020.

[11] S. Mohammad, M. M. M. Rahman, and F. Farahmandi, "Required policies and properties of the security engine of an soc," in *2021 IEEE International Symposium on Smart Electronic Systems (iSES)*, 2021, pp. 414–420.

[12] N. Farzana *et al.*, "SoC Security Properties and Rules," *Cryptology ePrint Archive*, 2021.

[13] W. Stallings *et al.*, *Computer Security: Principles and Practice*. Pearson Upper Saddle River, 2012, vol. 2.

[14] N. N. Anandakumar *et al.*, "Rethinking watermark: Providing proof of IP ownership in modern socs," *IACR Cryptol. ePrint Arch.*, p. 92, 2022.

[15] J. Rajendran *et al.*, "Security analysis of integrated circuit camouflaging," in *ACM SIGSAC conference on Computer & communications security*, 2013, pp. 709–720.

[16] K. Z. Azar *et al.*, "From cryptography to logic locking: A survey on the architecture evolution of secure scan chains," *IEEE Access*, vol. 9, pp. 73 133–73 151, 2021.

[17] H. M. Kamali *et al.*, "Advances in Logic Locking: Past, Present, and Prospects," *Cryptology ePrint Archive*, 2022.

[18] Y. Lyu *et al.*, "System-on-chip security assertions," *arXiv preprint arXiv:2001.06719*, 2020.

[19] H. Kyung *et al.*, "Performance monitor unit design for an axi-based multi-core soc platform," in *ACM Symposium on Applied Computing*, 2007, pp. 1565–1572.

[20] F. Zaruba *et al.*, "The cost of application-class processing: Energy and performance analysis of a Linux-ready 1.7-GHz 64-bit RISC-V core in 22-nm FDSOI technology," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 11, pp. 2629–2640, 2019.

[21] Skywater Technology, "SkyWater Open Source PDK," https://github.com/google/skywater-pdk, 2020.

[22] Digilent Genesys 2, "Genesys 2 FGPGA Development Board," https://digilent.com/reference/programmable-logic/genesys-2/start, 2016.

[23] O. Mutlu and J. S. Kim, "Rowhammer: A retrospective," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 8, pp. 1555–1571, 2020.