Hacking the Fabric: Targeting Partial Reconfiguration for Fault Injection in FPGA Fabrics

Jayeeta Chaudhuri[†], Hassan Nassar[‡], Dennis R.E. Gnad[‡], Jörg Henkel[‡],
Mehdi B. Tahoori[‡], and Krishnendu Chakrabarty[†]
[†]School of Electrical, Computer, and Energy Engineering, Arizona State University, Tempe, AZ, USA
[‡]Institute of Computer Engineering, Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany

2024 IEEE 33rd Asian Test Symposium (ATS) | 979-8-3315-2916-1/24/$31.00 ©2024 IEEE | DOI: 10.1109/ATS64447.2024.10915413

***Abstract*—FPGAs are now ubiquitous in cloud computing infrastructures and reconfigurable system-on-chip, particularly for AI acceleration. Major cloud service providers such as Amazon and Microsoft are increasingly incorporating FPGAs for specialized compute-intensive tasks within their data centers. The availability of FPGAs in cloud data centers has opened up new opportunities for users to improve application performance by implementing customizable hardware accelerators directly on the FPGA fabric. However, the virtualization and sharing of FPGA resources among multiple users open up new security risks and threats. We present a novel fault attack methodology capable of causing persistent fault injections in partial bitstreams during the process of FPGA reconfiguration. This attack leverages power-wasters and is timed to inject faults into bitstreams as they are being loaded onto the FPGA through the reconfiguration manager, without needing to remain active throughout the entire reconfiguration process. Our experiments, conducted on a Pynq FPGA setup, demonstrate the feasibility of this attack on various partial application bitstreams, such as a neural network accelerator unit and a signal processing accelerator unit.**

I. Introduction

Field-Programmable Gate-Arrays (FPGAs) are increasingly being used for customized accelerator platforms. Today, FPGAs are widely incorporated by cloud service providers such as Amazon Web Services (AWS) and Microsoft Azure to accelerate user modules in the cloud [1] [2]. Due to the flexibility and versatility of FPGAs across a wide range of high-performance computing applications, several industry and academic projects have suggested that FPGAs should operate in multi-tenancy [3] [4], similar to other shared resources in the cloud (CPUs, GPUs, memory). Specifically, in a multi-tenant scenario, multiple users can simultaneously reconfigure their applications on different modules of the same FPGA. Currently, FPGAs are capable of partial reconfiguration to enable multi-tenancy as it increases utilization and overall efficiency. Logical isolation is maintained between modules of different users to ensure the trust in their operation on multi-tenant reconfigurable fabrics [5].

However, FPGAs, being dynamically reconfigurable, are prone to security vulnerabilities [6] [7] [8] [9]. In a multi-tenant operation, users may gain access to the partial reconfigurable regions (PRRs) of the FPGA fabric to execute fault attacks by deploying malicious power-wasting circuits; these attacks can lead to severe voltage drops in the FPGA, resulting in computational faults or denial-of-service (DoS) of the FPGA and other co-tenants using it [6] [10] [11]. In [6], ring oscillators (ROs) are utilized to create sudden voltage fluctuations, which are then used to carry out fault attacks on the Advanced Encryption Standard (AES) module. [7] describes remote side-channel attacks in a multi-tenant scenario using voltage-based sensors to retrieve the secret key from an AES module. [12] shows that remote side-channel attacks can occur by exploiting the crosstalk between long wires, potentially leaking secret information from AES.

The fault attacks demonstrated in [6] [10] [11] require that the power-wasting circuits be activated for the entire duration while the victim module is configured on the FPGA. These attacks have temporary fault effects on the victim module and typically result in detectable faults that can be identified by prior detection schemes [13] [14].

Despite the demonstrated success of previously proposed countermeasures, there is a need to explore new security solutions; attackers can exploit the vulnerabilities in new techniques for managing dynamic reconfiguration in FPGAs. For example, to efficiently manage bitstream reconfiguration in the PRR of an FPGA, the authors in [15] propose a runtime reconfiguration manager (RM), namely Command-based Reconfiguration Queue (CoRQ). The RM can store multiple partial bitstreams that are uploaded by different users, and subsequently perform partial reconfiguration of these bitstreams on the FPGA [14].

In this paper, we introduce a novel type of FPGA-internal fault attack that can be activated and deactivated remotely. This attack exploits the process through which bitstreams are loaded to the FPGA, specifically targeting the partial reconfiguration process. The attack is precisely timed, e.g., based on sensing side-channel leakages [16], to inject faults in a bitstream **while** it is being reconfigured on the RM, and cease once the bitstream is fully loaded onto the RM. As highlighted in [13], voltage sensors typically take a longer time to detect most attacks. The proposed attack is designed to be active only during the reconfiguration process, which lasts only milliseconds; hence, it evades detection by [13] [14]. Additionally, the fault introduced during bitstream reconfiguration is not merely transient. Instead, the injected fault becomes embedded within the bitstream itself. Therefore, even after the bitstream is reconfigured to the FPGA, the faults persist in the FPGA. We

The work of Jayeeta Chaudhuri and Krishnendu Chakrabarty was supported in part by the National Science Foundation under grant no. CNS-2011561. The work of Hassan Nassar was supported in part by the German Federal Ministry of Education and Research (BMBF) through grant 01IS23066 as part of the Software Campus Project "HE-Trust" and in part by the "Helmholtz Pilot Program for Core Informatics (kikit)" at KIT.

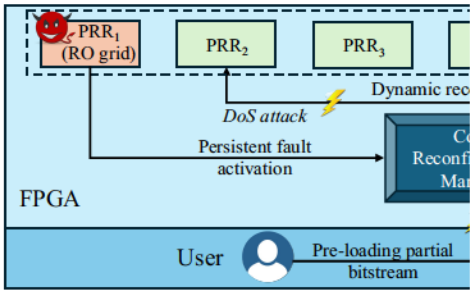

Fig. 1: Proposed persistent reconfiguration f

refer to this type of new attacks as *persistent reconfiguration fault attacks*. Fig. 1 illustrates the fault-attack scenario.

The key contributions of this paper are as follows.

- We show the first persistent reconfiguration fault attack where we leverage pre-configured power-wasters to corrupt user bitstreams while they are being loaded through the RM in a multi-tenant FPGA environment.
- We introduce persistent faults into partial bitstreams that configure the neural network accelerator unit and the digital signal processing unit, resulting in erroneous computations after their reconfiguration on the FPGA.
- We highlight the effectiveness of our attack through experimental demonstration using a Xilinx FPGA board.

The remainder of the paper is organized as follows. Section II describes prior attacks and related work on multi-tenant FPGAs. Section III presents the proposed attack methodology and the threat model. In Section IV, we demonstrate successful persistent reconfiguration fault attacks on several bitstreams. Section V concludes the paper.

II. Background and Related Prior Work

A. Security Threats for Multi-Tenant FPGAs

In a multi-tenant environment, multiple users share the same chip, but run multiple contexts, which often means that they also share the same power distribution network (PDN). The PDN is responsible for supplying power to all the user modules located on the FPGA. It is composed of a network of resistive, capacitive, and inductive elements. While higher resistance leads to a greater voltage drop, rapid current fluctuations can cause significant voltage spikes due to inductance. In [6], authors show fault-injection attacks via power-wasting RO circuits on an AES module deployed on a multi-tenant FPGA. The voltage fluctuations induced in the PDN are impacted by the following parameters of the RO grid: (a) toggling frequency, (b) number of active cycles when the RO grid is enabled, and (c) number of RO instances. Due to voltage fluctuations from the ROs, timing fault injection is made possible, which makes the AES module generate faulty ciphertexts. These ciphertexts are then analyzed using Differential Fault Analysis (DFA) to retrieve the secret key.

In [17], the authors demonstrate that ROs occupying less than 12% of the LUTs on the FPGA fabric are sufficient to cause excessive voltage fluctuations of the FPGA. Activating a grid of ROs to switch on and off at a specific frequency, namely f_{toggle} may lead to high power consumption, potentially causing voltage fluctuations. These voltage fluctuations can cause voltage-based faults or crashes. In [18], voltage-based fault attacks can be observed when the percentage of LUTs configured by ROs is as low as 25%.

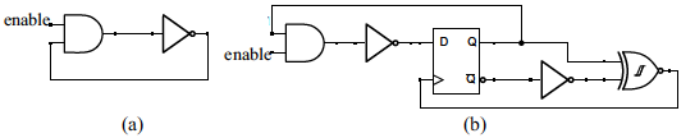

Fig. 2: (a) Single instance of an RO, (b) Self-clocked RO.

In [19], authors present a fault-injection attack on deep neural network (DNN) accelerators running on cloud FPGAs. An on-chip voltage sensor based on time-to-digital converters (TDC) is used to monitor the voltage fluctuations [20], while a malicious power-wasting circuit incurs glitching at specific time instances. High power consumption results in voltage glitches, impacting computation in the DNN layers and resulting in timing faults. The glitch-inducing circuits contain self-oscillating loops, which evade design rule check (DRC). Malicious power-wasting circuits based on AES and shift registers are shown in [21], while [22] explore non-combinational ROs based on latches and flip-flops. When activated at specific frequencies, even non-combinational ROs cause significantly high voltage drops leading to voltage and power-based attacks on the FPGA. Since these circuits do not contain combinational loops, they evade DRC warnings, and can be maliciously deployed on cloud FPGAs to launch power-based attacks. The work in [23] demonstrates successful bitstream generation corresponding to non-combinational ROs on the AWS platform. Examples of non-combinational circuits that are capable of fault-injection attacks are shown in Fig. 2.

In [11], an attack model is demonstrated that does not require specific types of power-wasting circuits such as combinatorial loops, which might leave detectable signatures in the FPGA bitstream. Instead, the attack leverages the capability of causing short circuits in the FPGA by writing opposite logic values concurrently to the same memory address from both ports of a dual-port RAM in the FPGA. The resulting memory collisions can create severe voltage drop and excessive heating of the FPGA, leading to DoS.

B. Detecting FPGA-Based Attacks

Several countermeasures have been explored to prevent fault attacks based on power-wasting circuits, where some methods check the designs before they are getting loaded, and others try to detect the attack during runtime.

To prevent deployment of malicious designs, reverse engineering (RE)-based techniques are used to convert an FPGA bitstream into its technology-mapped netlist. The netlist is then analyzed for combinational loops and other malicious circuits before deploying the corresponding bitstream on the cloud FPGA [24]. AWS DRC also has basic capabilities for detecting some of the possible combinational loops within user designs that are deployed on their cloud infrastructure [22] [25]. [26] and [27] explore bitstream checking mechanisms that are applicable to large-scale cloud deployments. The checks

for malicious bitstream detection include combinational loops, presence of large fanout, short circuits, and glitch detection. Machine learning (ML)-based detection methods have also been proposed recently to extract malicious signatures directly from bitstreams. In [28], bitstreams are converted into image files and used to train a convolutional neural network (CNN) model to identify RO-based signatures. [23] additionally diagnoses the type of malicious circuit implemented by a specific bitstream. These methods can effectively detect non-combinational ROs as well as several power-wasting circuits.

To detect fault attacks at runtime, in [13], the authors develop an on-chip voltage sensor to monitor voltage fluctuations periodically across several regions (or tenants) of an FPGA. In another work, a RO-based power monitor is used to detect the insertion of power-wasting circuits in the FPGA [29]. In [20], voltage-based attacks are detected using TDCs, where in [14], that information is used to disable specific interconnect points of attacker modules, reducing the impact of these attacks.

III. Persistent Attacks on FPGA Tenants During Reconfiguration

A. Proposed Persistent Attack Methodology

We leverage the multi-tenancy capability of an FPGA to induce persistent reconfiguration fault attacks on partial bitstreams. Specifically, we use power-wasters that already reside in our malicious PRR (cf. Fig. 1) to integrate persistent faults into user designs while they are being dynamically reconfigured on the PRRs of the FPGA through a RM. Once activated, the power-wasters induce persistent fault attacks that corrupt the partial bitstreams being loaded to the FPGA. Contrary to prior fault attacks demonstrated in [6] [10] [11], the persistent reconfiguration fault attack is precisely timed to corrupt the partial bitstream only when it is being loaded to the FPGA. Once the faults are injected, they continuously corrupt the bitstream configured to the PRR of the FPGA, and persist throughout the operation of the FPGA until the device is reconfigured. Consequently, even if the power-wasters are deactivated, the FPGA continues to execute based on the corrupted configuration, producing faulty computations. In summary, the proposed attack is particularly unique from prior fault attacks presented in [6] [10] [11] because it:

- Reduces the duration for which the power-wasters need to remain activated, referred to as *the exposure window*, thus minimizing the risk of detection by voltage sensors [13].
- Extends the attack impact, as the fault-injected bitstream remains corrupted until the next reconfiguration.

Although the persistent fault attack scenario targets a specific RM in our experiments, it is important to note that this attack is generic and can be applied to a range of different reconfiguration platforms used in FPGAs. The proposed fault attack results in erroneous computations when user designs are deployed on the FPGA. We show the feasibility and severity of the proposed attack on two classes of typical use-cases for FPGA, specifically neural network acceleration and signal processing. Table I presents a qualitative comparison of our attack with prior attacks on multi-tenant FPGAs.

TABLE I: Comparison of proposed attack framework with prior FPGA-based attacks.

Attribute	[6]	[17]	[10]	[11]	**Proposed method**
Target module	AES	FPGA	FPGA bitstream	RAM	Partial bitstream
Attack mechanism	RO grid	RO grid	Modifying LUT values	Memory collisions	RO grid
FPGA	Cyclone V	Virtex-6, Zynq	Virtex-5	Spartan-6	Pynq
Attack objective	Key recovery	DoS	Key recovery	Timing faults, bit-flips	Fault-injection, DoS
Caused DoS?	Yes	Yes	No	Yes	Yes
Self-clocked RO evaluated?	No	No	No	No	Yes

B. Threat Model

We assume that the FPGA is spatially shared by different user modules. In this multi-tenant scenario, an adversary can be a third-party user who gains access to the FPGA to deploy a malicious circuit. A RM is configured on the same FPGA. The RM is responsible for (1) pre-loading the partial bitstreams for FPGA reconfiguration, and (2) configuring the bitstreams in the PRRs of the FPGA. The RM is considered semi-trusted, meaning that another tenant on the same FPGA could exploit a vulnerability of the RM, e.g., side-channel leakage. In this scenario, the goal of the attacker is to configure a malicious power-waster on one of the PRRs of the FPGA such that it induces faults while other user bitstreams are getting configured on a PRR of the same FPGA through the RM. The power-waster can be a voltage sensor such as RO, which detects side-channel leakages, such as voltage fluctuations, during the loading of a partial bitstream on the RM [16] [30]. For instance, by monitoring the voltage levels, an attacker can time the activation of the power-wasters to inject faults into the partial bitstream. Although the PRRs of the FPGA support logical isolation, the PDN of the FPGA is shared among these modules. When the power-wasters are activated at a particular toggling frequency f_{toggle}, it causes a sudden voltage drop in the PDN. Due to the abrupt voltage fluctuations, faults can be injected into the bitstream while it is being reconfigured on the FPGA through the RM.

The RM enables the loading of both encrypted and decrypted bitstreams. However, bitstreams are usually decrypted before they are configured on the FPGA [31]. Therefore, we evaluate our proposed attack specifically on decrypted and unencrypted bitstreams that are loaded to the FPGA.

IV. Experimental Platform and Results

A. Fundamentals of CoRQ framework

CoRQ [15] was implemented as a runtime RM framework for placing partial bitstreams onto the FPGA. If a bitstream is encrypted, CoRQ uses specific commands to decrypt it before FPGA reconfiguration. Most of the FPGA vendors such as Xilinx and Intel decrypt a bitstream during the configuration process [31]. Therefore, in all our experiments on CoRQ, we evaluate the fault attack using unencrypted partial bitstreams. The RTL implementation of CoRQ is written in VHDL. The block diagram of CoRQ is shown in Fig. 3. The modules illustrated in Fig. 3 are explained below.

- CoRQ_top: It denotes the CoRQ framework. CoRQ includes

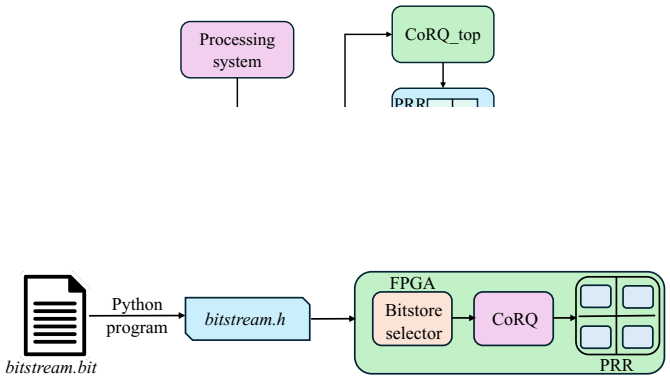

Fig. 4: Procedure of reconfiguring a partial bitstream on the FPGA through CoRQ.

a **Bitstore** memory and an internal configuration access port (ICAP). The bitstreams are first stored into the memory, before being reconfigured on the FPGA via the ICAP.

- AXI interconnect: The AXI is the standard interface used for facilitating high-bandwidth communication between the processing system and the programmable logic of the FPGA. The use of AXI interfacing in CoRQ facilitates reading the bitstreams from the Bitstore memory and dynamically reconfiguring them on the FPGA.
- Processing system: It denotes the processor core.

The application code involving the software driver for using CoRQ and the bitstream decryption units have been implemented in C. The partial bitstreams (.bin format) are converted into header files using a Python program '*encrypt.py*' before uploading it to the software driver of CoRQ. Embedding the bitstream as a header file enables the bitstream to be included within the application code, thus facilitating dynamic partial reconfiguration. If the evaluated partial bitstreams are unencrypted, the process bypasses any need for decryption.

After implementation of the CoRQ and the grid of ROs on the Pynq FPGA, we export the hardware from Vivado to Vitis. This generates the .XSA (Xilinx Support Archive) file, which helps to run the software application associated with CoRQ. Once uploaded, CoRQ reconfigures the header files corresponding to each partial bitstream on the static PRRs of the FPGA. The procedure of configuring an user bitstream via CoRQ is shown in Fig. 4.

While a bitstream is being uploaded to the FPGA through CoRQ, we monitor the $corq\_status$ flag to detect a persistent fault-injection attack during the reconfiguration process.

B. Fault-Attack Evaluation

We present three case studies to evaluate the persistent reconfiguration fault-injection attack. In the first case study, we monitor the blinking of the LEDs of the FPGA to detect initiation of fault injection in a partial design. For this experiment, we evaluate the following partial bitstreams – blinkall, blinkline, and blinkcount. The above bitstreams are implemented specifically to test their effect on the blinking of the FPGA LEDs. The blinkall bitstream enables blinking of all the LEDs simultaneously. The blinkline bitstream causes the LEDs to blink in a sequence. Finally, the blinkcount bitstream involves blinking LEDs in a pattern that represents a count sequence. In the second case study, we study persistent reconfiguration fault attacks on a Multiply-Accumulate unit, which is a crucial component in neural network accelerators that are deployed by cloud providers such as AWS F1 instance. In the third case study, we further demonstrate the fault attack on a Fast Fourier Transform unit. The Fast Fourier Transform unit is crucial for many applications such as speech and audio recognition and analysis, and for polynomial multiplication of homomorphic encryption which run on cloud [32]. Several commercially available FPGAs such as Intel Altera and Xilinx Zynq support Multiply-Accumulate and Fast Fourier Transform operations for performing high-performance computing applications such as deep learning and signal processing.

TABLE II: Decryption time of encrypted partial bitstreams.

Type of encryption circuit	Size (in bytes)	Time (in clock cycles)
aes_medium	65532	533820
aes_big	65540	534006
aes_full	131068	1066358

TABLE III: The size (in bytes) of the evaluated partial bitstreams.

Bitstream	blinkline	blinkall	blinkcount
Size (in bytes)	51956	40524	40820

In the following case studies, we present fault-attack results where the attacker uses power-wasters based on ROs to disrupt the partial configuration of designs on the multi-tenant FPGA, such that persistent errors are there in the configured design. We choose the number of RO instances as 50,000 to be consistent with experimental setups in prior work [18]. To initiate the fault-injection process during the loading of user designs to the RM, we activate the *ro_ena* signal in the power-waster. Enabling the *'ro_ena'* bit activates the power-wasters to perform fault-injection during loading of the partial bitstreams to CoRQ. A demo of the fault-attack experiments is available in [33], showing errors in the computation of a neural network accelerator unit and a digital signal processing unit when power-wasters are activated. The fault attack is evaluated on a Pynq-Z1 FPGA.

1) Case Study 1: Fault Attack Observed via Blinking LED

We evaluate the persistent reconfiguration fault attack on the blinkall, blinkline, and blinkcount partial bitstreams. We convert these bitstreams to their corresponding header files before preloading them to CoRQ. Table III lists the size of the header files corresponding to these bitstreams.

In common practice, a cyclic redundancy check (CRC) is enabled by default in the partial bitstreams (in this specific case, blinkline, blinkall, and blinkcount) before they are configured on the FPGA [34]. When enabled, an initial CRC value is generated for the original partial bitstream. When the partial bitstream is loaded to CoRQ, the CRC value is recalculated before it is configured on the FPGA. If the recalculated CRC value does not match the original value, it is flagged as a fault-injection attack. Therefore, if a fault occurs during dynamic partial reconfiguration of an user design, CoRQ will raise a flag, indicating that one or multiple bits in the partial bitstream is corrupted. As a result, CoRQ will stop the partial

reconfiguration process for that specific design. In cases wh the partial bitstream is encrypted, the CRC is disabled. T CRC can be disabled during the bitstream generation proc using the constraint *BITSTREAM.GENERAL.CRC Disable* Xilinx Vivado. For this particular case study, we report observation by 1) enabling and 2) disabling CRC.

Enabling CRC We keep CRC enabled during the bitstre generation process of the blinkall, blinkline, and blinkco designs. After bitstream generation, we obtain the header fi corresponding to the bitstreams as explained in Section IV Next, we launch Vitis IDE through Vivado to access software controller application of CoRQ, preload the part header files to CoRQ, and then dynamically reconfigure the on the FPGA. While uploading the blinkall bitstream to CoRQ, an error is observed in $corq\_status$ as follows:

Error while loading blinkall_plain. corq_return: 2

Thus, when the CRC is enabled, the persistent reconfiguration fault attack is immediately detected at the preloading stage, which prevents the design from being configured to the FPGA; this attack is demonstrated in [33]. From [33], we observe that only if all the bitstreams are successfully preloaded to CoRQ, they are reconfigured to the FPGA. Therefore, if the fault attack occurs in one of the partial bitstreams, the remaining bitstreams are also prevented from FPGA configuration. This leads to a DoS of CoRQ.

Disabling CRC Next, we generate the partial bitstreams by disabling the CRC during Vivado bitstream generation. By disabling the CRC, we ensure that no verification occurs when preloading the bitstreams to CoRQ, allowing the fault-injected bitstreams to be loaded onto the FPGA via CoRQ. A demo of the glitching of LEDs induced by the persistent reconfiguration fault attack is shown in [33]. We perform 200 iterations of the preloading of the blinkline, blinkcount, and blinkall bitstreams to statistically assess the impact of RO-induced persistent faults on the FPGA LED behavior. We observe that the blinkline and blinkcount bitstreams are not impacted by the persistent fault-injection attack using ROs. Therefore, when these specific bitstreams are configured to the FPGA via CoRQ, the LEDs blink according to their corresponding functionalities. For example, when the blinkline is uploaded to the FPGA, the LEDs blink in a sequence, indicating that it has been successfully configured. However, the blinkall bitstream is injected by the faults. Therefore, when the faulty blinkall bitstream is uploaded to the FPGA, we observe glitching in the rhythm of blinking of the LEDs on the FPGA.

2) *Case Study 2: Fault Attack on a Neural Network Accelerator Unit*

Next, we demonstrate the persistent reconfiguration fault attack on a neural network accelerator component, specifically the Multiply-Accumulate unit. The partial bitstream that implements the Multiply-Accumulate unit is converted to its corresponding header file, known as *mac.h*. This file, *mac.h* is then loaded onto the FPGA using CoRQ for evaluation. We provide a pair of test inputs (v_1, v_2) to the Multiply-Accumulate unit configured on the FPGA, and collect the output from the Multiply-Accumulate operation. We specify the register addresses, namely *IN_A* and *IN_B* to store the test input values v_1 and v_2, respectively. After the Multiply-Accumulate computation, we read the data from the output port of the Pynq FPGA. The computed value received from the FPGA, is given by N_{comp}. The expected value N_{mac} is given by: $N_{mac} = N'_{mac} + v_1 \times v_2$, where N'_{mac} represents the previously accumulated value.

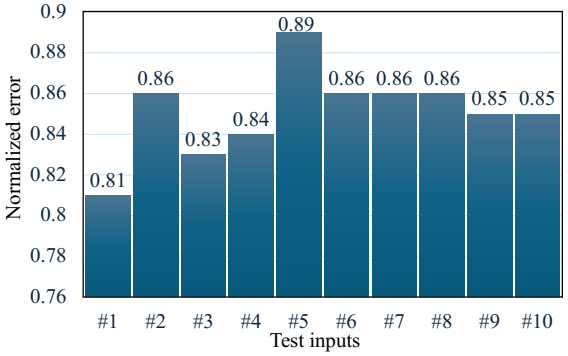

Fig. 5: Impact of fault injection (in terms of the normalized error) on a partial user design implementing the Multiply-Accumulate unit.

We use the metric e_{mac} to quantify the error between the computed output and the expected outcome. The normalized error e_{mac} (absolute value) is given by $e_{mac} = |\frac{N_{mac} - N_{comp}}{N_{mac}}|$, where N_{mac} is the expected output of the Multiply-Accumulate operation and N_{comp} is the computed value in the presence of a persistent reconfiguration fault attack. We choose the test inputs randomly from the range [1, 10], and perform the Multiply-Accumulate computation over 10 iterations i.e., for 10 different test input pairs (v_1, v_2). The partial accumulated sum from $(i-1)^{th}$ iteration is added to the product of test input pair (v_1, v_2) in the i^{th} iteration.

Fig. 5 illustrates the normalized error due to fault-injection attack on the Multiply-Accumulate unit. Table IV lists the N_{comp} and e_{mac} values over 1000 iterations; the faulty N_{comp} outputs highlight the severity of RO-based fault-injection.

3) *Case Study 3: Fault Attack on a Digital Signal Processing Unit*

We show evaluation results of the persistent reconfiguration fault attack for a digital signal processing component, specifically the Fast Fourier Transform unit. The partial bitstream implementing Fast Fourier Transform is converted to the respective header file, *fft.h*. As explained in the previous subsection, we use a similar technique to provide a pair of test inputs (v_1, v_2) to the Fast Fourier Transform unit that is configured on the FPGA, and read the computed data from the FPGA. The actual (expected) outcome of the Fast Fourier Transform operation, N_{fft} is given by $N_{fft} = v_1 \times v_2$. The normalized error e_{fft} (absolute value) is formulated as $e_{fft} = |\frac{N_{fft} - N_{comp}}{N_{fft}}|$, where N_{fft} is the expected output of the Fast Fourier Transform computation and N_{comp} is the computed value in the presence of the persistent reconfiguration fault attack. For the i^{th} iteration, test inputs are chosen

TABLE IV: Normalized error for fault-injected Multiply-Accumulate and Fast Fourier Transform units.

Iteration	Multiply-Accumulate		Fast Fourier Transform	
	N_{comp}	e_{mac}	N_{comp}	e_{fft}
10	39	0.84	5	0.66
50	343	0.76	9	0.87
100	875	0.66	13	0.62
250	1967	0.73	10	0.66
500	3648	0.76	0	1

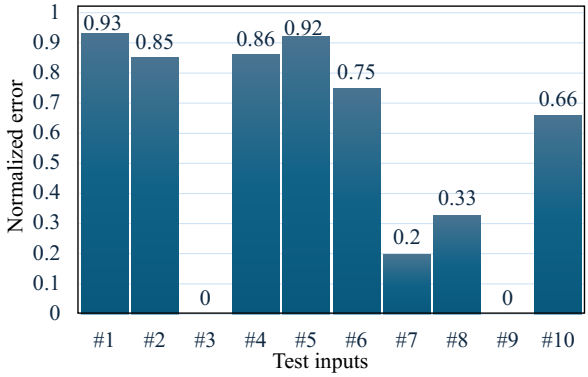

Fig. 6: Impact of fault injection (in terms of the normalized error) on a partial bitstream implementing the Fast Fourier Transform unit.

randomly from the range [0, 10]; the computation in each iteration is independent of the previous iterations. Fig. 6 shows the normalized error over 10 different test input pairs. We report the e_{fft} values over 1000 iterations in Table IV.

C. Comparison to Runtime Fault-Injection Attacks

Several previous works have discussed fault injection at runtime. [6] used ISCAS-based circuits to inject faults on AES. [18] explored the right combinations of fault-injection parameters for causing fault attacks on specific types of FPGAs. [35] targets injecting faults to DNNs. All these methods require the attack to persist for the whole runtime of the victim accelerator to be effective. For example, if the victim accelerator runs for two hours, the attack will have to run for two hours. This duration puts the attack at risk as simple monitoring of the power activity can give it away. In contrast, our attack induces persistent faults and only needs to be active during the upload of the bitstream, typically in the range of milliseconds. Once, the bitstream is configured to the PRR of the FPGA, the attack can cease, yet the faults remain embedded in the configured bitstream. Therefore, even if our attack is detected, it will already have affected the accelerator and the fault will persist.

V. Conclusion

We have demonstrated persistent fault attacks on several partial bitstreams that are uploaded to the FPGA via the RM, and have shown how these attacks can successfully lead to faulty computations in a multi-tenant environment. We have shown that for as low as 15% of FPGA LUTs used for implementing the power-wasters, a DoS condition can be triggered in the RM. Additionally, we have demonstrated successful fault attacks on several partial bitstreams while they are being loaded to the RM, thus highlighting a critical vulnerability of the FPGA reconfiguration process.

References

[1] Amazon, "Amazon EC2 F1 Instance," https://go.aws/3ENtUj9, 2021.
[2] D. Firestone *et al.*, "Azure accelerated networking: SmartNICs in the public cloud," in *15th USENIX (NSDI)*, 2018, pp. 51–66.
[3] J. M. Mbongue *et al.*, "Architecture support for FPGA multi-tenancy in the cloud," in *ASAP*. IEEE, 2020, pp. 125–132.
[4] M. Yamakura *et al.*, "A multi-tenant resource management system for multi-FPGA systems," *IEICE TRANSACTIONS on Information and Systems*, vol. 104, no. 12, pp. 2078–2088, 2021.
[5] T. Huffmire *et al.*, "Moats and drawbridges: An isolation primitive for reconfigurable hardware based systems," in *SP '07*, 2007, pp. 281–295.
[6] J. Krautter *et al.*, "FPGAhammer: Remote voltage fault attacks on shared FPGAs, suitable for DFA on AES," *IACR TCHES*, 2018.
[7] F. Schellenberg *et al.*, "An inside job: Remote power analysis attacks on FPGAs," in *DATE*, 2018.
[8] M. Zhao *et al.*, "FPGA-based remote power side-channel attacks," *Proc. IEEE S&P*, 2018.
[9] R. Elnaggar and K. Chakrabarty, "Machine learning for hardware security: Opportunities and risks," *JETTA*, vol. 34, no. 2, 2018.
[10] P. Swierczynski *et al.*, "Bitstream fault injections (BiFI)–automated fault attacks against SRAM-based FPGAs," *IEEE TC*, 2018.
[11] M. M. Alam *et al.*, "RAM-Jam: Remote temperature and voltage fault attack on FPGAs using memory collisions," in *FDTC*, 2019, pp. 48–55.
[12] C. Ramesh *et al.*, "FPGA side channel attacks without physical access," in *FCCM*, 2018, pp. 45–52.
[13] G. Provelengios *et al.*, "Mitigating voltage attacks in multi-tenant FPGAs," *ACM TRETS*, vol. 14, no. 2, 2021.
[14] H. Nassar *et al.*, "LoopBreaker: Disabling interconnects to mitigate voltage-based attacks in multi-tenant FPGAs," in *ICCAD*, 2021.
[15] M. Damschen *et al.*, "CoRQ: Enabling runtime reconfiguration under WCET guarantees for real-time systems," *IEEE Embedded Systems Letters*, vol. 9, no. 3, pp. 77–80, 2017.
[16] F. Schellenberg *et al.*, "An inside job: Remote power analysis attacks on FPGAs," *IEEE Design & Test*, vol. 38, no. 3, pp. 58–66, 2021.
[17] D. Gnad *et al.*, "Voltage drop-based fault attacks on FPGAs using valid bitstreams," in *FPL*, 2017, pp. 1–7.
[18] D. R. Gnad *et al.*, "Remote electrical-level security threats to multi-tenant FPGAs," *IEEE Design & Test*, vol. 37, no. 2, 2020.
[19] Y. Luo *et al.*, "Deepstrike: Remotely-guided fault injection attacks on DNN accelerator in cloud-FPGA," in *DAC*, 2021, pp. 295–300.
[20] K. M. Zick *et al.*, "Sensing nanosecond-scale voltage attacks and natural transients in FPGAs," ser. FPGA. ACM, 2013.
[21] G. Provelengios *et al.*, "Power wasting circuits for cloud FPGA attacks," in *FPL*, 2020.
[22] T. Sugawara *et al.*, "Oscillator without a combinatorial loop and its threat to FPGA in data centre," *Electronics Letters*, 2019.
[23] J. Chaudhuri and K. Chakrabarty, "Diagnosis of malicious bitstreams in cloud computing FPGAs," *IEEE TCAD*, vol. 42, no. 11, 2023.
[24] J. Krautter *et al.*, "Mitigating Electrical-level Attacks towards Secure Multi-Tenant FPGAs in the Cloud," *ACM TRETS*, vol. 12, no. 3, 2019.
[25] Amazon AWS, "Amazon AWS FPGA usage," https://bit.ly/2XcGnXy.
[26] T. M. La *et al.*, "FPGADefender: Malicious self-oscillator scanning for Xilinx UltraScale + FPGAs," *ACM TRETS*, 2020.
[27] T. La *et al.*, "Denial-of-service on FPGA-based cloud infrastructures—attack and defense," *IACR TCHES*, 2021.
[28] J. Chaudhuri and K. Chakrabarty, "Detection of malicious FPGA bitstreams using CNN-based learning," in *ETS*, 2022.
[29] A. L. Masle and W. Luk, "Detecting power attacks on reconfigurable hardware," in *FPL*, 2012, pp. 14–19.
[30] V. Meyers *et al.*, "Active and passive physical attacks on neural network accelerators," *IEEE Design & Test*, vol. 40, no. 5, pp. 70–85, 2023.
[31] Xilinx, "Ultrascale architecture configuration," https://bit.ly/3yyxvQ9 .
[32] R. Kiesel *et al.*, "Potential of homomorphic encryption for cloud computing use cases in manufacturing," *Journal of Cybersecurity and Privacy*, vol. 3, no. 1, pp. 44–60, 2023.
[33] "Demo files for fault attacks on FPGA," https://rb.gy/ce4v1x [Downloading is necessary for the videos to be played].
[34] AMD, "Error testing using POST_CRC in 7 Series FPGAs," https://rb.gy/3mbsag.
[35] A. Boutros *et al.*, "Neighbors from hell: Voltage attacks against deep learning accelerators on multi-tenant FPGAs," in *ICFPT*, 2020.