Criticality Analysis of Ring Oscillators in FPGA Bitstreams*

Jayeeta Chaudhuri† and Krishnendu Chakrabarty‡
†Department of Electrical and Computer Engineering, Duke University, Durham, NC, USA
‡School of Electrical, Computer, and Energy Engineering, Arizona State University, Tempe, AZ, USA

2023 IEEE European Test Symposium (ETS) | 979-8-3503-3634-4/23/$31.00 ©2023 IEEE | DOI: 10.1109/ETS56758.2023.10173861

***Abstract*—The popularity of cloud computing has led to increasing demand for efficient and scalable hardware. Multi-tenant FPGAs are becoming popular because of their ability to provide high performance and flexibility, yet being cost-effective. While multiple tenants have the ability to configure the same FPGA with customized modules, several security vulnerabilities can be exploited by adversaries. Attackers can use an FPGA to perform malicious actions, such as injecting malicious bitstreams and launching denial-of-service attacks. We propose a two-tier machine learning framework that first detects malicious features from an FPGA bitstream and then performs criticality analysis to evaluate the severity of potentially malicious ring oscillators (ROs) configured by that bitstream. The latter step is crucial as it ensures the security of FPGAs from voltage and power-based attacks and also reduces the risk of inappropriately blocking benign RO-based circuits from FPGA configuration. The proposed framework is evaluated using a diverse set of real-world bitstreams. We achieve an accuracy of 100% in detecting malicious bitstreams and an accuracy of 96.55% in detecting malicious bitstreams that are critical.**

I. Introduction

Field-programmable gate-arrays (FPGAs) are often used in applications that require high performance with customization, such as video processing, cryptography, and deep learning. FPGAs are now commonplace in cloud computing environments and are also being incorporated in data centers hosted by Amazon and Microsoft [1], [2]. FPGAs offer flexibility and enhanced performance in the implementation of complex machine learning (ML) applications and FPGA-accelerated services. For example, users upload their own customized applications to cloud FPGAs. As FPGAs offer multi-tenancy, users share a common platform where they implement their own applications while being logically isolated from others.

While multi-tenant FPGAs offer a number of benefits, there are several associated security risks that are of concern. Some key security threats include fault attacks, power and voltage-based attacks, and denial-of-service (DoS) attacks. An attacker may activate several ROs simultaneously, causing the FPGA to overheat, and subsequently malfunction [3].

Prior work on malicious bitstream detection has focused on reverse-engineering (RE) of the FPGA configuration bitstream by looking for specific patterns that indicate malicious behavior [4]. However, the development of these RE tools is arduous and complex. A promising approach is to use ML-based techniques which detect RO-like signatures more effectively than traditional methods. The work in [5] represents bitstreams as data-series images and uses a convolutional neural network (CNN) to detect RO-like patterns from the images.

Although prior methods can efficiently detect and prevent RO-based circuits from FPGA configuration, not much attention has been devoted to determining the criticality of a particular RO variant that is implemented by a user bitstream. A digital circuit that performs genuine, real-life applications can be blocked from FPGA configuration due to the presence of RO-based circuits in its design. For example, ROs are used in true random number generators (TRNGs) to generate secret keys. Therefore, it is crucial to perform criticality analysis of the incoming FPGA bitstream and determine the security impact of the ROs implemented by that bitstream. We propose an ML-based criticality analysis framework to assess the input bitstreams before they are configured on an FPGA. The key contributions of this paper are as follows:

- We present a method that can automatically detect malicious RO-based Trojans implemented by FPGA bitstreams.
- We develop a two-stage ML-based framework that incorporates frequency domain-based feature extraction from the FPGA bitstreams for malware criticality assessment.
- We devise a metric that provides a score for each bitstream indicating its criticality, and further use this metric to compute the criticality of RO-based Trojans that are capable of launching power- and voltage-based attacks.
- We evaluate the effectiveness of the proposed method on a variety of real-world bitstreams.

The remainder of the paper is organized as follows. Section II discusses attacks on FPGA-based systems and prior countermeasures. Section III presents the threat model. Section IV describes the procedure of identifying RO-based signatures from FPGA bitstreams. Section V presents the two-tier ML-driven framework for criticality analysis of ROs. Section VI presents experimental results. Section VII concludes the paper.

II. Background

A. Attacks on FPGA-based Systems

One of the most important components of an FPGA-based system is its power distribution network (PDN). The PDN supplies power to all the components of the FPGA and is typically modeled using RLC circuits. The high switching activity of ROs can cause a large amount of current to be drawn from the PDN, resulting in a voltage drop. The increased current draw can lead to significant power consumption of the FPGA.

Prior work has demonstrated successful voltage-based attacks on FPGAs, thus disrupting their functionality and leading

*This work was supported in part by the National Science Foundation under grant no. CNS-2011561.

TABLE I: Prior malicious bitstream detection methods.

Characteristics	[4]	[9]	[12]	**Proposed method**
Performed criticality analysis	✗	✗	✗	✓

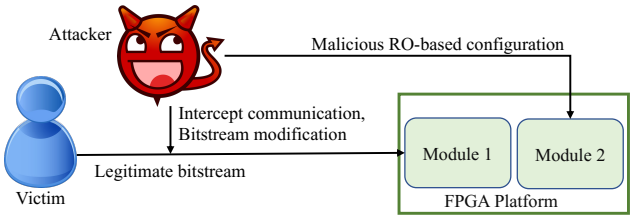

Fig. 1: Schematic of the threat model.

to DoS [6]. In [3], a fault attack using ROs is presented. By carefully controlling the frequency of the RO and utilizing only 25% of the available FPGA logic, an attacker can create voltage fluctuations that cause the FPGA to crash. In another approach [7], malicious non-combinational ROs based on flip-flops and latches evade the design rule check (DRC) and are carried out for Amazon Web Services (AWS) instances.

B. Countermeasures

In [4], the Yosys tool is extended to detect combinational cycles in technology-mapped netlists obtained through reverse-engineering of FPGA bitstreams. However, the reverse-engineering of bitstreams is a time-consuming procedure. For example, a simple bitstream performing only partial reconfiguration may take a few hours to reverse engineer whereas a full bitstream performing more complex tasks (e.g., data analysis) could take days or even weeks to reverse engineer [8]. The work in [9] analyzes netlist graphs obtained from bitstreams for malicious patterns such as combinational cycles, hidden ROs, and self-oscillating circuits. However, the procedure of generating a netlist graph from a bitstream can take a considerable amount of time depending on the complexity of the design. Table I summarizes the limitations of existing methods for malicious bitstream detection and also highlights our contributions. ML-based approaches have also been used to detect hardware Trojans. The work in [10] [11] learns features that are extracted from Trojan-inserted and Trojan-free netlists using an ML classifier.

III. Threat Model

The threat model is illustrated in Fig. 1. We assume that an attacker can configure an FPGA with malicious RO-based circuits. The FPGA supports time-based multi-tenancy i.e., a single user has access to the FPGA at a time. The circuits have the potential to cause overheating and DoS in the FPGA. An attacker may also intercept the communication between a legitimate user and the deployed FPGA, and modify the bitstream that configures the FPGA. This can be done by either physically altering the bitstream, or by using a software tool to modify the bitstream before it is downloaded to the FPGA. Our proposed ML-based bitstream detection framework is executed off-chip and is assumed to be physically accessible only to authorized users. Hence, an adversary will not be able to tamper the detection framework. As explained in [13], the FPGA first decrypts the incoming FPGA bitstream before it is configured. Therefore, in this work, we evaluate our proposed classification framework specifically on decrypted bitstreams.

IV. Learning Malicious Signatures using ML

A. Data Pre-processing

We assume without loss of generality that the FPGA bitstreams are generated as .bin files. While full bitstreams can be used to train ML models, they have a few limitations. First, due to their large size, full bitstreams can lead to higher training times for the ML classifier. Second, in a full bitstream, many bytes are either unused or have no effect on the output functionality, making it difficult for the ML model to learn specific patterns [14]. Alternatively, training the ML models on window-based partitioned bitstreams to identify malicious signatures significantly reduces the training time. Since we aim to identify specific windows from a given bitstream that correspond to possible RO-based patterns, we propose dividing a bitstream into a set of ψ windows. Since the size of a Xilinx Ultrascale FPGA bitstream is 128966372 bytes, the size of each window $n = \lceil \frac{128966372}{\psi} \rceil$. For a given number of windows ψ, we train ψ equivalent ML classifiers. For training each classifier, we first partition the bitstreams in the dataset into ψ windows. Next, we store the partitioned bitstreams as Numpy (.npy) arrays. Let us denote the i^{th} window by $\phi^i_\psi, 1 \leq i \leq \psi$. We train the i^{th} ML classifier on the window ϕ^i_ψ extracted from the dataset of benign and malicious bitstream arrays. Evaluation results for the choice of ψ are provided in Section VI.

B. Importance of Criticality Assessment of RO-based Designs

The *criticality* of a bitstream is determined by the amount of malicious behavior that can be caused by the ROs implemented by that bitstream. The number of ROs and the frequency of the ROs implemented by the bitstream affect how critical the bitstream is. The criticality analysis of a bitstream can help in reducing the risk of inappropriately blocking benign RO-based circuits from FPGA configuration. A metric for evaluating the criticality of a given bitstream is presented in Section VI-B. This metric can help us to identify which bitstreams are more likely to cause harm to the FPGA.

The proposed approach is based on CNNs to learn features extracted from the frequency-domain representation of the FPGA bitstreams. One of the most popular feature extraction techniques is Fast Fourier transform (FFT) [15]. FFT is a useful tool for exploratory analysis and can be used for detecting malicious patterns in FPGA bitstreams.

V. Proposed Critical Assessment Framework

Before the FPGA is configured, the bitstream is fed to our classification framework to determine the criticality of the RO-based circuit implemented by that bitstream. Fig. 2 illustrates the proposed two-tier framework.

A. Data Generation

We evaluate the proposed method on bitstreams implementing a diverse set of real-world benign and malicious designs. For our experiments, we target the Xilinx Virtex Ultrascale (VU440) and Kintex Ultrascale (KU085) FPGA boards. These

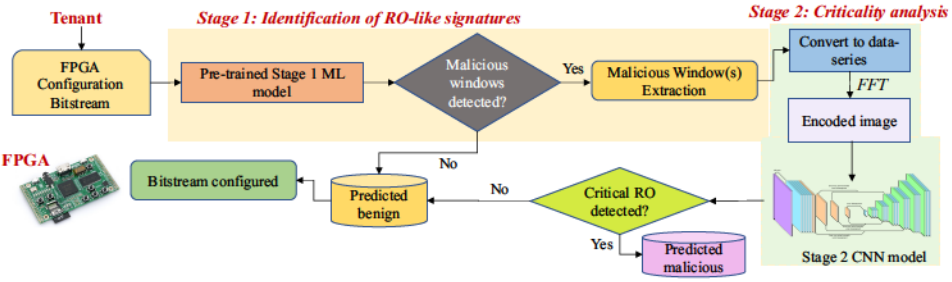

Fig. 2: Methodology used for criticality analysis

FPGAs are used in many real-world applications, such as high-performance computing, networking, and data storage [16].

Benign bitstreams: These bitstreams implement AES cores, microprocessors, VGA Opencores, LCD controllers, and IS-CAS '85, ITC '99, and EPFL benchmark circuits.

Malicious bitstreams: In this work, we define a malicious bitstream as a bitstream that contains RO-based signatures. However, as discussed in Section IV, an RO-based design may not be critical to the FPGA it is configured upon. In other words, all critical bitstreams are malicious but all malicious bitstreams might not be critical.

- *Critical bitstreams*: We implemented simple ROs, loop-free ROs (including latched and self-clocked oscillators), staged ROs, and conditional ROs. We ensure that the number of stages of these ROs are consistent with the number of RO circuits that are used to launch voltage-based attacks in [3].
- *Non-critical bitstreams*: These bitstreams implement several variants of TRNGs and physically unclonable functions (PUFs). Although these bitstreams include ROs, they are not harmful when actually placed on the FPGA.

B. First Stage: Detection of Malicious Signatures

1) Selection of ML model

Next, we select a suitable classifier for training and evaluation purposes based on the F1-score. We evaluate the following supervised ML models [17]: 1) Random Forest (RF), 2) Support Vector Machine (SVM), 3) eXtreme Gradient Boosting (XGB), and 4) Multilayer Perceptron (MLP).

2) Training and Inferencing with Pre-Trained ML Model

After obtaining a suitable ML classifier, we train ψ such classifiers, namely ζ_i, where $1 \leq i \leq \psi$, according to the procedure explained in Section IV. For VU440 bitstreams, the number of input features to ζ_i is $\lceil \frac{128966372}{\psi} \rceil$. We employ hyper-parameter tuning to improve the trainability of our model in learning RO-based signatures from the dataset. To run ζ_i in the evaluation mode for inferencing, we split the bitstream into ψ windows of equal size and pass each of the windows through the ψ ML classifiers. Given a window ϕ_ψ^i, the classifier ζ_i returns a confidence score S_i^1, such that: $0 \leq S_i^1 \leq 1, 1 \leq i \leq \psi$. We specifically calculate the S_i^1 values as they are a measure of the accuracy of an ML model's predictions. A higher confidence score S_i^1 suggests that the i^{th} window more likely contains malicious RO-based signatures. We collect all such malicious windows and their corresponding S_i^1 values, and pass them to Stage 2 for criticality classification and evaluation.

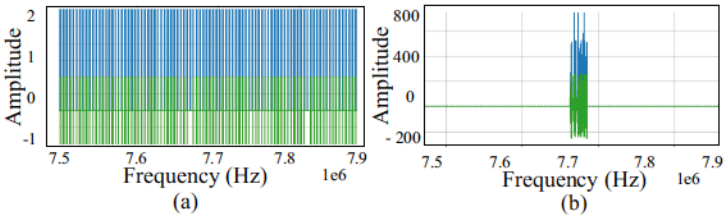

Fig. 3: FFT-encoded images of: (a) Critical RO; (b) TRNG.

C. Second Stage: RO Criticality Assessment

1) Data Collection

We store the malicious windows extracted from the first stage of the detection pipeline as comma-separated values (CSV) files. Next, we apply FFT to convert the CSV files into the frequency domain. The FFT-encoded images corresponding to examples of a critical and a non-critical circuit are illustrated in Fig. 3(a) and Fig. 3(b), respectively.

2) Neural Network Architecture

We apply a CNN-based detection approach to perform criticality assessment on FFT-encoded images. We reshape the encoded images to 224×224 arrays before model training. Note that we train the Stage 2 model on only the critical and non-critical malicious bitstreams. The CNN architecture used in our experiments has five convolutional layers and five max-pooling layers, followed by a fully-connected layer. We use the ReLU and Softmax activation functions in the fully-connected layer. While ReLU reduces the risk of overfitting, Softmax allows better generalization of the input data and robustness to outliers than other activation functions [18].

VI. Experimental Results

A. Experimental Setup

We implement the benign and malicious circuits in Verilog and generate bitstreams corresponding to these circuits using Xilinx Vivado 2018.2. We use Python 3.4 to build the ML and CNN models. The training and inferencing of the ML and CNN models are run on a 2.4 GHz Intel Xeon Gold 5115 CPU with 768 GB of RAM. The selected hyperparameters for the two-stage framework are listed in Table II.

We generate 150 benign bitstreams and 156 malicious bitstreams for the VU440 board. To ensure that these bitstreams are diverse and sufficient, we implement a variety of malicious circuits as well as benign samples that are deployed in real-life applications [3] [4] [6]. We randomly split the benign and malicious bitstreams into training and test datasets in the ratio 70:30. Therefore, the training dataset consists of 214

TABLE II: Best hyperparameters for the selected classifiers.

Classifier	Selected hyper-parameters
SVM	$C = 100$, kernel: RBF
CNN	Loss: Categorical crossentropy, Optimizer: RMSProp, Learning rate = 0.001, Number of epochs = 200

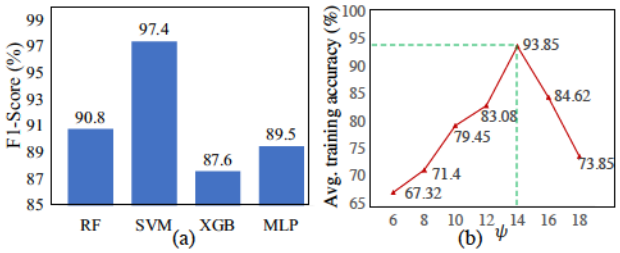

Fig. 4: (a) F1-score of the ML classifiers after k-fold cross-validation (b) Determining the best value of ψ using SVM.

bitstreams and the test dataset comprises of 92 bitstreams. The bitstreams in the evaluation dataset include 34 benign bitstreams, 29 malicious and critical bitstreams, and 29 non-critical malicious bitstreams.

B. Evaluation Metrics

- TPR_{mal} is the percentage of malicious bitstreams that are correctly classified as malicious. TNR_{ben} is the percentage of malicious non-critical bitstreams that are correctly identified as non-critical.
- Classification accuracy (A_c) is the ratio of the number of correct predictions to the total number of predictions.
- For an input bitstream, the criticality metric γ is formulated as: $\gamma = S_k^1 \times S^2, 1 \leq k \leq \psi$, $0 \leq \gamma \leq 1$, where S_k^1 is the highest confidence score (corresponding to the window ϕ_ψ^k) returned by the Stage 1 ML classifiers and S^2 is the confidence score associated with the label (i.e., critical or non-critical) predicted by the Stage 2 CNN model. After running simulations on a variety of test bitstreams (benign and malicious), we conclude that a bitstream with $\gamma > 0.7$ is critical. In general, γ can be determined on the basis of the data used for training the CNN model.
- R_c is the percentage of critical bitstreams that are correctly classified as critical.

C. Overall Training and Evaluation Results

1) Choice of ML Classifier and Selection of ψ

We perform k-fold cross-validation and evaluate the F1-scores to determine a suitable ML classifier for training and the inferencing of malicious RO-based signatures in Stage 1 of our detection framework. We select $k = 5$ as it is commonly used in practice [17]. We present the performance results of the ML classifiers in Fig. 4(a). We observe that SVM provides the highest F1-score. In Fig. 4(b), we observe that $\psi = 14$ is the best choice for window-based partitioning of the bitstreams.

2) Analysis of RO Criticality

As shown in Table III, the best results are obtained when we select time-domain (TD)-based malicious bitstream detection exclusively in Stage 1 and frequency-domain (FD)-based criticality analysis particularly in Stage 2 of the pipeline. We observe that selecting other combination of feature extraction techniques results in poor performance metrics for the two-stage framework. Additionally, in Table III, we highlight the

TABLE III: Evaluation results using different frameworks.

Framework	Stage 1		Stage 2	
	A_c (%)	TPR_{mal} (%)	R_c (%)	TNR_{ben} (%)
TD (1) + TD (2)	96.73	100	82.7	85.18
TD (1) + FD (2)	**96.73**	**100**	**96.55**	**100**
FD (1) + TD (2)	40.2	55.17	48.27	62.06
FD (1) + FD (2)	40.2	55.17	65.5	44.82
Prior work [5]	93.47	94.82	76.92	79.31

TABLE IV: Evaluation of γ for test bitstreams.

Type	Count	Avg. value of γ
Benign	34	0.008
Malicious but non-critical	29	0.115
Malicious and critical	29	0.863

strength of the proposed method over prior work on detection of malicious RO signatures [5]. The average γ values for each type of bitstream are listed in Table IV. The time overhead during the inferencing phase is 1.1 minutes.

D. Evaluation for Other FPGA Families

For a KU085 FPGA bitstream (size: 48251520 bytes), we obtain $\psi = 9$ by hyperparameter tuning. Note that the training and test dataset include the circuits described in Section V-A. The evaluation (in %) of the two-stage framework on a test dataset of 30 benign and 52 malicious bitstreams yields: $A_c = 91.46, TPR_{mal} = 93.33, R_c = 95.45$, and $TNR_{ben} = 91.3$. The time overhead associated with the criticality assessment of a test bitstream is less than 1 minute.

VII. Conclusion

We have presented a method for efficiently detecting RO-like signatures and determining the criticality of different RO variants implemented via FPGA configuration bitstreams. Our method can be easily extended to other FPGA families, with minimal modifications. The proposed two-tier ML/CNN-based framework achieves a significantly higher classification accuracy compared to baseline detection frameworks.

References

[1] Amazon, "Amazon EC2 F1 Instance," https://go.aws/3ENtUj9, 2021.
[2] K. Eguro *et al.*, "FPGAs for trusted cloud computing," in *FPL*, 2012.
[3] D. Gnad *et al.*, "Voltage drop-based fault attacks on FPGAs using valid bitstreams," in *FPL*, 2017, pp. 1–7.
[4] J. Krautter *et al.*, "Mitigating Electrical-level Attacks towards Secure Multi-Tenant FPGAs in the Cloud," *ACM TRETS*, vol. 12, no. 3, 2019.
[5] J. Chaudhuri *et al.*, "Detection of malicious FPGA bitstreams using CNN-based learning," in *ETS*, 2022.
[6] M. Zhao *et al.*, "FPGA-based remote power side-channel attacks," *Proc. IEEE S&P*, 2018.
[7] T. Sugawara *et al.*, "Oscillator without a combinatorial loop and its threat to FPGA in data centre," *Electronics Letters*, 2019.
[8] T. Wollinger *et al.*, "Security on FPGAs: State-of-the-art implementations and attacks," *ACM TECS*, p. 534–574, 2004.
[9] T. M. La *et al.*, "FPGADefender: Malicious self-oscillator scanning for Xilinx UltraScale + FPGAs," *ACM TRETS*, 2020.
[10] T. Inoue *et al.*, "Designing hardware trojans and their detection based on a SVM-based approach," in *ASICON*, 2017, pp. 811–814.
[11] J. Yang *et al.*, "Hardware trojans detection through RTL features extraction and machine learning," in *AsianHOST*, 2021, pp. 1–4.
[12] H. Nassar *et al.*, "Loopbreaker: Disabling interconnects to mitigate voltage-based attacks in multi-tenant FPGAs," in *ICCAD*, 2021, pp. 1–9.
[13] Xilinx, "Ultrascale architecture configuration," https://bit.ly/3yyxvQ9 .
[14] E. Alpaydin, *Introduction to Machine Learning*. The MIT Press, 2010.
[15] SKlearn, "Fourier transforms," bit.ly/3hco841.
[16] Xilinx, "AWS cloud," https://bit.ly/3Vgx1aT.
[17] Scikit-learn, "Machine learning in Python," https://bit.ly/3OzdLBZ .
[18] A. Krizhevsky *et al.*, "Imagenet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, no. 6, p. 84–90, 2017.