FPGA Security Techniques with Applications to Cloud and Multi-Tenant Use Cases

FPGA SECURITY TECHNIQUES WITH APPLICATIONS TO CLOUD AND MULTI-TENANT USE CASES

A Dissertation Presented

by

XIANG LI

Submitted to the Graduate School of the
University of Massachusetts Amherst in partial fulfillment
of the requirements for the degree of

DOCTOR OF PHILOSOPHY

February 2024

Electrical and Computer Engineering

FPGA SECURITY TECHNIQUES WITH APPLICATIONS TO CLOUD AND MULTI-TENANT USE CASES

A Dissertation Presented

by

XIANG LI

Approved as to style and content by:

Daniel Holcomb, Chair

Russell Tessier, Member

Adam O'Neill, Member

Wayne Burleson, Member

Christopher V. Hollot, Department Head
Electrical and Computer Engineering

ACKNOWLEDGMENTS

Firstly, I wish to thank my advisor Prof. Daniel Holcomb. He has given me numerous and consistent guidance and helped me to be successfully in each stage of my Ph.D. studies. Also, I want to thank Prof. Russell Tessier, Prof. Burleson and Prof. Adam O'Neill for their valuable and rigorous feedback to my works. I am grateful to my colleagues including Shayan Moini, George Provelengios, Siva Nishok Dhanuskodi who collaborated and contributed to my works. Besides, I appreciate help from other lab members during these years from Arjun Suresh, Shahriar Hadayeghparast, Shahrzad Keshavarz, Bharadwaj Madabhushi, Tiankai Su and Jiteshri Dasari. I also feel grateful to have friends including Taioh Kutoba, Wei Huang. In the end, I wish to thank the support from my parents.

ABSTRACT

FPGA SECURITY TECHNIQUES WITH APPLICATIONS TO CLOUD AND MULTI-TENANT USE CASES

FEBRUARY 2024

XIANG LI

B.Sc., NORTHWESTERN POLYTECHNICAL UNIVERSITY

M.Sc., XI'AN JIAOTONG UNIVERSITY

Ph.D., UNIVERSITY OF MASSACHUSETTS AMHERST

Directed by: Professor Daniel Holcomb

Field programmable gate arrays (FPGAs) are integrated circuits that consist of programmable logic that a user can configure and deploy for applications such as hardware emulation and accelerating high performance computing. In recent years, the emergence of FPGAs in the cloud has led to research on multi-tenant FPGAs. In a multi-tenant scenario, the same FPGA fabric is shared among multiple users, or among multiple untrusting IP cores. Multi-tenancy has economic benefits, largely due to improvements in resource utilization, but also brings new security concerns since the tenants could behave maliciously. Although the tenants sharing an FPGA are logically isolated from each other, they may still have unintended interactions through side channel attacks and fault attacks. In this dissertation, we aim to evaluate security threats and defenses in the multi-tenant FPGA scenario.

Firstly, the work in this dissertation studies a true random number generator (TRNG) on cloud FPGAs that is robust against voltage manipulation from co-tenants. The TRNG design is based on harvesting clock jitter using a tunable time-to-digital converter circuit. In accordance with best practices, a stochastic model is built to evaluate the min-entropy of the design, and further validated by NIST entropy assessment test suite and NIST statistical tests. The basic version of the TRNG is extended with a linkable sampling module to increase min-entropy per sample and throughput at a modest resource cost.

Then the dissertation analyzes a type of fault attack that can be conducted by one tenant against another in a multi-tenant setting. Specifically, the fault attack is differential fault intensity analysis (DFIA), which is a biased-fault based attack on Advanced Encryption Standard (AES) circuits. Ring oscillators (ROs) are deployed as effective power wasters to cause a supply voltage drop through the shared power distribution network (PDN) of tenants. The attack is highly relevant to multi-tenant scenarios because the attacking tenant can create the voltage drop without physical access, and can precisely control the shape of the voltage drop by adjusting both the number of activated ROs and their duration as required for the attack. The voltage drop will in turn increase the delay in the logic and eventually cause specific timing faults which are analyzed to successfully recover the AES keys.

In the last part, we use on-chip voltage sensors to detect the location of a target circuits. The sensing scheme leverages time-to-digital converters (TDCs) as voltage sensors, and a novel differential analysis is applied to the sensor data. In a multi-tenant setting, this method can be used either as part of a defensive scheme to monitor against attacks, or it can be used to probe a system and determine how to effectively target an attack to a particular co-tenant victim.

TABLE OF CONTENTS

LIST OF TABLES

Table **Page**

LIST OF FIGURES

Figure **Page**

CHAPTER 1

INTRODUCTION

1.1 Current State of FPGAs

A field-programmable gate array (FPGA) is an integrated circuit (IC) that contains many configurable logic blocks. FPGAs have a wide range of applications for datacenter acceleration, hardware emulation, and distributed memory systems. Modern FPGAs possess abundant resources such as lookup tables (LUTs), flip-flops, carry chains, block RAMs, and digital signal processing (DSP) elements. Each building block of an FPGA can be configured to implement combinational and sequential logic and these building blocks are connected by a programmable routing network of wires and switches.

FPGAs are an important and growing segment of the overall semiconductor market. The global FPGA market was valued at $6.9 B in 2022, and is expected to reach $12.44 B by 2028 [1]. The largest FPGA manufacturers at the time of this thesis are Xilinx and Intel. Xilinx holds approximately 50% - 55% of FPGA market while Intel comprises 32% - 35% as of 2020 [2].

1.2 FPGA Tools and Design Flow

The most common FPGA design software tools are Intel Quartus Prime and Xilinx Vivado. Both tools support a wide range of FPGAs and provide a library of intellectual property (IP) blocks that a user can deploy in their designs. The tools are essentially each providing a complete self-contained computer aided design (CAD) flow from register transfer level (RTL) to bitstream as explained in the following paragraphs.

Additionally, users can perform behavioral and functional simulation, static timing analysis, and generate utilization and power reports with the tools.

The software flow for implementing a design on an FPGA begins with a hardware description language (HDL) such as VHDL, Verilog and SystemVerilog. After users write the HDL for their designs, the FPGA software synthesizes it into a network of LUTs, flip-flops, and block RAMs. Among these resources on FPGAs, the most common resources are LUTs and flip-flops; large FPGAs may have more than a million instances of each. A flip-flop is a sequential element to store a bit, used in aggregate to create state machines or memory blocks. A LUT functions as combinational logic; an n-input LUT can be configured to implement any n-input logic function because it is essentially a user-defined 2^n-bit table in which each input combination causes a different bit of the table to propagate to the LUT output. An example of AND function implemented by a LUT6 on an FPGA is shown in Fig. 1.1. In a synthesis-based design flow, the user typically works from behavioral Verilog, but when desired Vivado also allows the user to explicitly instantiate LUTs and flip-flops in their HDL as structural Verilog primitives.

After synthesis, the software tool will place each logical instance of a primitive to a corresponding physical instance of the primitive, subject to placement constraints, while also deciding how to connect them with routing resources so that the routed design meets timing requirements. Users can also supply custom constraint files to impose specific requirements. These requirements may include defining where a LUT or flip-flop is placed on the FPGA, specifying a clock source for a design, or setting maximum or minimum delay of a path. In this dissertation, constraint files are widely used to optimize timing and fix placement and routing.

Finally, a bitstream which contains the information of logic, routing and initial values of registers is created. The bitstream can be loaded on the FPGA to configure

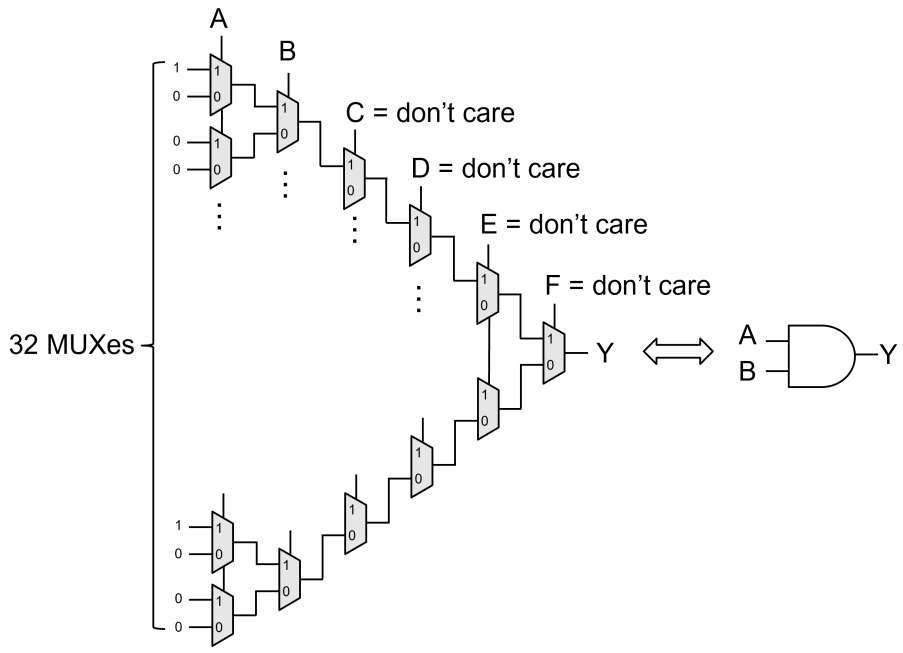


Figure 1.1: 2-input AND function implemented by a LUT6 on an FPGA. Inputs C, D, E, and F of the LUT6 are don't-care logic as their values do not affect the output Y.

it to implement the design. In this dissertation, Vivado is the primary development tool, and Verilog and SystemVerilog are the HDLs being used.

1.3 FPGAs vs. ASICs

It is instructive to contrast FPGAs against application-specific integrated circuits (ASICs) to better understand their respective places in the market. ASICs are ICs typically designed for a sole application such as in medical devices, telecommunication, or digital cameras. Compared with ASICs, FPGAs are programmable after fabrication, providing users a platform to implement versatile applications. Because each design implemented on an FPGA does not require a mask set, the non-recurrent engineering (NRE) cost and implementation time are both much lower when a design is implemented on an FPGA instead of as an ASIC. The lower NRE makes FPGA attractive for low-volume parts. Additionally, one important application of FPGAs is their use in emulating large ASICs so that the logic of the ASIC can be extensively tested before

the corresponding ASIC is produced. In this use case, FPGA emulation can be viewed as higher-performance alternative to a software simulation of the ASIC design, and ensures that the design is correct when the ASIC gets subsequently fabricated.

Compared to purpose-built ASICs, there are costs to the generality and reconfigurability of FPGAs. FPGAs have lower performance than ASICs, because the LUTs and programmable routes in FPGAs have higher propagation delays than the analogous standard cells and dedicated wires of ASICs. For the same reason, FPGAs are also less power-efficient than ASICs. Moreover, the unit cost of FPGAs is higher than that of ASICs because cost is proportional to die area, and the LUTs and programmable routes of FPGAs are larger than the analogous standard cells and custom routes of ASICs.

1.4 Cloud FPGAs and Multi-tenant FPGAs

As size, resources, and complexity of modern FPGAs evolve, some companies, such as Amazon Web Services (AWS) [3] and Microsoft [4], have been developing platforms and protocols for cloud FPGA instances. Microsoft initiated the Catapult project to accelerate large-scale datacenter services in 2014 [5]. AWS introduced EC2 F1 FPGA instances in 2016, allowing customers to create designs to accelerate applications such as natural language processing, artificial intelligence, and deep learning. Instead of purchasing expensive FPGAs costing tens of thousands of dollars for these applications, users can rent high-end cloud FPGAs for around one dollar per hour. In this dissertation, the cloud FPGA we use is AWS EC2 F1 FPGA, which is a 16 nm Xilinx UltraScale+ FPGA. After a design is completed, it can be registered as an Amazon FPGA Image (AFI), enabling reuse across multiple instances.

To reduce costs and maximize the resource utilization of cloud FPGAs, researchers have proposed the notion of multi-tenant FPGAs. One type of multi-tenant FPGAs is a single FPGA fabric shared spatially by several users, each running individual

applications in a logically isolated region, supported by partial reconfiguration such that each region can be changed at runtime. Although the notion has been proposed, this type of multi-tenancy is not deployed on cloud FPGAs yet. A second type of multi-tenancy on FPGAs arises from aggregating multiple third-party IPs in one design, and this is already common practice today. In this case, one IP can be viewed as a co-tenant by another IP on the same FPGA. While third-party IPs can be embedded to accelerate the design process, a malicious IP could monitor or tamper with user data, or create covert channels that leak user information. This type of threat is akin to a hardware Trojan [6].

1.5 Security Threats on Cloud and Multi-tenant FPGAs

While the utilization efficiency of cloud FPGAs increases with multi-tenancy, new security threats have arisen, presenting severe challenges for cloud FPGAs or multi-tenant FPGAs. Cloud FPGA vendors have the full control over their platforms. They have physical access to FPGAs and are able to control whether users can even access their own data. Besides, once a user's design is completed, the bitstream is entrusted to the cloud provider. The cloud provider can potentially reverse-engineer the bitstream and gain knowledge of the IPs like how data is processed or transmitted in a system, posing a risk of IP theft or data exfiltration [7].

In addition to the concerns about trusting FPGA vendors, cloud providers, and toolchains, security threats on multi-tenant FPGAs can also be introduced by malicious co-tenants, which is the focus of this dissertation. FPGA co-tenants can be either designs from other users that are dynamically assigned to a shared FPGA by the cloud provider, or can be third-party IPs deployed within a single design under the control of the user. In either case, circuits of a victim and attacker can be placed together on the same FPGA fabric. The shared hardware platform provides malicious co-tenants with access to perform side channel attacks, fault attacks, or establish

covert channels without control over the platform. The next chapter discusses these threats from co-tenants in details.

1.6 Main Takeaway of This Thesis

Multi-tenancy on FPGAs can be formed by running several designs on the same FPGA or integrating various third-party IPs in one design. While the adoption of multi-tenancy on FPGAs brings cost and design efficiency to users, it introduces security threat. Whether deliberate or inadvertent, the co-tenants on multi-tenant FPGAs can potentially cause malfunctions or lead to the leakage of private data from others. This thesis focuses on developing applications aimed at defending co-tenants against malicious ones and exploring novel remote attack methods initiated by malicious co-tenants on multi-tenant FPGAs, as discussed in Chapter 3, Chapter 4 and Chapter 5. Our research not only enhances the security of applications on cloud FPGAs but also serves as a reminder about the potential security challenges posed by co-tenants on multi-tenant FPGAs.

CHAPTER 2

RELATED WORK

This chapter provides background on the existing work about security threats in multi-tenant FPGAs, as well as overall information about this dissertation. Sections 2.1 to 2.3 introduce the previous research on side channel attacks, fault attacks, and associated countermeasure on multi-tenant FPGAs. Section 2.4 offers an overview of this dissertation. Section 2.5 shows my publications to date and introduces the publications that are not covered in this dissertation.

2.1 Fault Attacks

Fault attacks are a class of attacks that aim to cause errors while a computational process is running, resulting in incorrect outputs or intermediate values. On FPGA boards, fault attacks are implemented traditionally by manipulating the clock or power, which requires physical access to the board. In the multi-tenancy context where physical access may be restricted, the attacker can instead inject faults using on-chip power wasters to cause a drop in supply voltage.

2.1.1 Power Wasters

Malicious FPGA tenants can cause the supply voltage drop and induce timing faults in other tenants. Despite the absence of physical connections between different tenants, they share a common power distribution network (PDN) on an FPGA. Consequently, the attacker can induce voltage drops by placing power waster circuits that consume significant power to overwhelm the PDN. Voltage drops can lead to increased delays

in logic operations. Thus, when the voltage drop lasts for one cycle or more, and has large enough magnitude, it induces timing faults in other nearby circuits or even a denial-of-service (DoS) [8] by crashing the FPGA. In most cases, voltage-based attacks lack stealth since the voltage reduction could be sensed by the victims.

Ring oscillators (ROs), circuits consisting of an odd number of inverters connected in a loop, are commonly deployed as power wasters. The logic level between each inverter keeps switching at a high frequency, consuming a significant amount of power. Some cloud FPGA providers like AWS use design rule checking (DRC) to prohibit ROs [9] and other combinational loops. However, there are circuits like flip-flop based ROs [10, 11], garbled XOR [12] and AES-based power wasters [13, 14] that can pass DRC and still consume enough power to inject faults.

In addition, researchers have also found more methods to inject faults. Alam et al. [15] show that simultaneously writing opposite values to the same address of the dual-port RAM on FPGAs can cause transient short circuits that create excessive heat and voltage drop. Such attacks occur at runtime and can bypass bitstream analysis tools with RO checkers.

2.1.2 Remote Voltage-based Fault Injections

Gnad et al. [16] use flip-flop based ROs as power wasters to explore how their activation patterns affect transient voltage drop in FPGAs. Their experiment shows that switching activities with a longer period and lower duty cycle can lead to higher transient voltage drop. In their later work [17], they perform DoS attack by sudden activation of ROs on Xilinx FPGAs with only about 12% of the available LUT resources. Mahmoud et al. [18] perform a voltage attack to induce timing faults in true random number generators (TRNGs) while changing the placement of victim and attacker as well as attack duration. In their design, 24.5% of the LUTs on a Xilinx Virtex-7 FPGA are occupied by ROs, which is sufficient to inject faults that

make outputs fail statistical tests. Luo et al. [19] present a stealthy power attack. The victim is attacked at the moment when higher power consumption is detected by ROs. Because of this opportunistic attack timing, their power wasters need a smaller incremental power consumption to shut down the FPGA, making it more stealthy. In a fault attack, attackers can also extract private information from the victim. Krautter et al. [20] toggle ROs to inject faults into an Intel Cyclone V FPGA on a Terasic DE1-SoC board. They calibrate and adjust the intensity to inject fault into specific bytes before ninth round of AES and successfully recover secret keys by performing differential fault analysis (DFA).

Besides targeting cryptographic algorithms, fault attacks have also been applied to machine learning model accelerators on multi-tenant FPGAs. Boutros et al. [12] evaluate the resilience of a deep learning accelerator running ImageNet classification against fault attacks on an Intel Stratix 10 FPGA. They show the conservative timing margin added by Quartus and resilience of the deep learning model make the prediction accuracy unaffected when they overclock the accelerator to a 27%-31% higher clock frequency. Besides, the overall prediction accuracy also remains unaffected with their strongest attack when the accelerator runs at its safe operating frequency. Luo et al. [21] put and activate latch-based ROs as power wasters beside a deep neural network (DNN) accelerator on a Xilinx PYNQ-Z1 FPGA board. Their test shows that the inference accuracy drops as the number of power wasters increases.

2.2 Side Channel Attacks and Covert Channel

Side channel attacks typically involve extracting private information from the victim without direct observation of the information itself. Instead, attackers can measure power, electromagnetic, and timing information and then analyze the information to infer knowledge about the victim. Classic side channel power analysis, such as differential power analysis [22] and correlation power analysis [23, 24], has been

demonstrated by researchers. Most of these attacks require physical access to the device. However, multi-tenant FPGAs provide new attack vectors to implement side channel attacks. In multi-tenant FPGAs, several works have shown that side channel attacks can be performed by on-chip voltage sensors or long wires without physical access.

2.2.1 Voltage Sensors

Time-to-digital-converters (TDCs) and ROs commonly play roles as on-chip voltage sensors to record voltage fluctuations. RO frequency is sensitive to and varies with the supply voltage. A counter is usually connected to ROs and used to count the oscillations during each sampling period. TDCs are commonly implemented by carry chains in FPGA because of their fine granularity [25, 26, 18, 27]. The output of each stage in the carry chains is sampled by a flip-flop. A rising or falling edge propagates through the carry chain each clock cycle. When there is a voltage drop, the edge passes fewer carry stages during the clock cycle and this is revealed by the sampled values in the flip-flops having a higher or lower Hamming weight. In this way, voltage fluctuations are recorded. Then the attackers can perform power analysis to the sensor data to infer the knowledge of the victim.

2.2.2 Remote Power Analysis

Zhao et al. [28] demonstrate power analysis to attack an RSA module on an FPGA by RO sensors and show that it is possible to recover the full 1,024-bit private key. Schellenberg et al. [29] use TDCs as Trojan sensors to monitor supply voltage fluctuation and successfully perform correlation power analysis to extract keys from an AES module. While most works about remote side channel attacks are carried out in a controlled environment, Glamocanin et al. [30] show a side channel attack in the cloud on an AWS EC2 F1 FPGA. They use TDC sensors to measure power consumption and recover the entire 16-byte key of an AES-128 module. Their research

shows that the security threat raised by multi-tenancy on cloud FPGAs is valid and there is necessity to mitigate it.

Besides cryptographic cores, side channel attacks can also be applied to accelerators for machine learning on multi-tenant FPGAs. Tian et al. [31] leverage TDCs to attack the versatile tensor accelerator on a Xilinx Zynq ZC706 board. The TDC module captures voltage traces while the accelerator is running, and the voltage traces leak information about the machine learning model implemented by the accelerator. With their approach, the attacker can infer the composition of instruction groups, and can reverse-engineer the structure and layer of the machine learning model. Moini et al. present a power-based side channel attack on a binarized convolutional neural network accelerator on both local Xilinx FPGAs [32] and AWS EC2 F1 FPGAs [33]. By analyzing TDC data, they successfully recover recognizable images of 10 digits that are inputs to the accelerator. Both of their works do not require physical access to the victim accelerators.

2.2.3 Long Wire Crosstalk Attacks

Resources other than voltage sensors have also been exploited to implement remote side channel attack. The logic value on long wires influences the delay of neighboring long wires, and this can cause data leakage or create covert channels. Giechaskiel et al. [34] firstly found the "long wire" coupling phenomenon. They observe that a long routing wire carrying logic 1 can reduce the propagation delay of the adjacent but unconnected long wires. The pairing long wires can be called transmitters and receivers. When the receivers are wires of ROs, the frequency of ROs will be influenced by the logic value on the transmitters. This crosstalk leakage between long wires opens a door to covert communication on multi-tenant FPGAs. In [34], such a covert channel achieves a bandwidth of 6 KBps and a 99.9% accuracy on Xilinx FPGAs. In their later work [35], they use RO variants to overcome combinational loop constraints

and measure delay changes of long wires on the cloud FPGA. Besides, in [36], they demonstrate even medium wires and logic within a configurable logic block (CLB) can cause information leakage. Ramesh et al. [37] find that long wire leakage also exists on Intel FPGAs and perform a side channel attack by placing neighboring long wires beside the victim. They successfully extract keys from an 8-bit datapath implementation [38] of an AES-128 engine, running at 10 MHz. The metric they use to characterize the delay change of the RO receiver depends on the period of the RO instead of just the long wire itself. To quantify the delay effects caused by the neighboring long wires more precisely, Provelengios et al. [39] demonstrate a new metric, which is able to eliminate RO-based variability, to characterize the coupling effect between long wires at the femtosecond scale. By exhaustively characterizing the information leakage between all possible wire pairs, they discover which pairs of wires are routed closely in different FPGA families without information from vendors.

2.2.4 Other Covert Channels

Alternatively, there are other ways to create covert channels. Gnad et al. [40] build a voltage-based covert channel that achieves a bandwidth of 8 MBps. By activating all ROs gradually and disabling all of them at once, the transmitter sends a positive voltage spike representing logic 1. By activating all ROs at once and disabling all of them gradually, a negative voltage spike representing logic 0 is sent. The receivers are TDCs that sense the voltage spike. Other than voltage-based covert channels, Tian et al. [41] explore a thermal covert channel on Microsoft Catapult cloud FPGAs, which are Intel Stratix V FPGAs with no restrictions enforced against combinational loops. In their work, the transmitter can activate many 3-stage ROs to heat the chip and then exit the FPGA and leave the FPGA idle. The receiver can subsequently load their bitstream to the same cloud FPGA and measure the frequency of their own ROs. Because RO frequency depends on temperature, they can infer whether the chip is

heated and thus receive the transmitted bit. Their approach could likely be adopted on other cloud FPGA platforms with restrictions on combinational loops, like AWS, but would then need to use sensors and heaters that are not based on ROs.

2.3 Countermeasures

Along with the works showing attacks on multi-tenant FPGAs, there have also been a number of works that propose countermeasures to these attacks, which can help to secure multi-tenant FPGAs against malicious co-tenants.

2.3.1 Countermeasures to Fault Attacks

One way to mitigate or detect fault attacks is to validate the DRC and ensure that malicious circuits fail predefined rules. For instance, AWS checks the bitstreams and prohibits the existence of combinational loops before a final bitstream is loaded onto the cloud FPGA. However, some variants of ROs, such as latch-based ROs and flip-flop based ROs, can still pass the checker [10]. La et al. [42] present an FPGA virus scanner against short-circuits, self-oscillating circuits, and circuits with high fanouts. Krautter et al. [43] propose a bitstream checking methodology to analyze the characteristics of malicious circuits such as combinational loops and delay line based sensors. New features have to be updated in bitstream checkers as malicious circuits with new structures emerge.

Another approach to mitigate fault attacks is to use runtime monitoring. Provelengios et al. [44, 45] use ROs to monitor the voltage drop and locate the malicious circuits by characterizing the magnitude of voltage drop. They use 46 ROs that only occupy 5% of LUTs on an Intel Cyclone V FPGA. Once a suspicious attacker is found, the process of the victim can be stopped, and the suspicious attacker can be disabled. Mirzargar et al. [46] also use ROs to detect power wasters. Compared with [44], a major improvement is that instead of placing RO sensors manually, they

automate the design process of searching for free and optimal resources in each clock region to insert RO sensors. Shen et al. [26] disable the clock when fast voltage transients are detected. The switching activities are disabled until the fast voltage transients pass. Luo et al. [47] propose a framework of dynamic frequency scaling by adjusting the phase-locked loop (PLL) or mixed-model clock manager (MMCM). The victim application can still run under a lower and safe frequency when voltage drop is detected.

2.3.2 Countermeasures to Side Channel Leakage

To mitigate remote side channel attacks, Krautter et al. [48] deploy active fences of ROs beside the design to flatten voltage fluctuation. ROs work as noise generators to reduce the signal-to-noise ratio. The number of activated ROs is based on the feedback of TDC as voltage sensors. They show that the required number of power traces to break the keys of an AES-128 module increases from 1.8K without mitigation to 300K with the mitigation. This countermeasure is simple to deploy and independent of the design being protected.

The current countermeasure against long wire attacks primarily focuses on avoiding wires with sensitive information being placed in proximity to malicious entities. Seifoori et al. [49] modify routing algorithms to route sensitive wires away from untrusted signals. This is achieved at cost of a small increase in routing resources and critical path delay. Luo et al. [50] provide a hardware isolation platform called HILL. They validate the long-wire obfuscation technique on AES and demonstrate that it effectively reduces the correlation between side channel leakage and the secret key. Other standard defenses such as time randomization can similarly be used to diminish the leakage [51].

As for thermal channel leakage, Tian et al. [41] propose to enforce a minimum idle period between users. Alternatively, one may consider cooling or heating the FPGA to a known temperature before the FPGA is accessed by the next user.

2.4 Dissertation Overview

In this dissertation, we delve into the security challenges on multi-tenant FPGAs. Our exploration begins with an in-depth study of a TRNG on AWS cloud FPGAs. Subsequently, we study a type of fault attack and a method for locating target circuits on multi-tenant FPGAs.

2.4.1 True Random Number Generator on Cloud FPGAs

In Chapter 3, we tailor a TRNG, which is an important cryptographic primitive, to AWS EC2 F1 cloud FPGAs. By harvesting randomness from clock jitter, the TRNG can obviate restrictions on combinational loops imposed by cloud FPGAs. We build a stochastic model to evaluate its min-entropy and also apply the model to 60 AWS EC2 F1 instances. Furthermore, our stochastic model is validated by the NIST entropy assessment suite. We also show that the samples from our design can pass the NIST statistical tests. In addition, we extend this baseline design to a linkable sampling module to improve throughput and min-entropy, as well as compare our work with prior TRNG designs.

Research Contribution: A detailed TRNG for cloud FPGAs is designed, the randomness of which is extracted from clock jitter. To the best of our knowledge, this is the first TRNG work on cloud FPGAs, avoiding the restriction on combinational loops imposed by cloud FPGA providers. Our design can be utilized to enhance cryptography systems on cloud FPGAs.

2.4.2 Differential Fault Intensity Analysis on AES

In Chapter 4, we explore a type of fault attack known as differential fault intensity analysis (DFIA) on AES, imposed by a co-tenant on multi-tenant FPGAs. We assume the victim is running an AES application, while the malicious co-tenant places ROs to inject timing faults. We elaborate on how to control the fault injection at precise

timing with fine granularity, analyze different faulty ciphertexts under various fault intensities, and successfully extract keys from AES on a Xilinx Spartan-7 FPGA.

Research Contribution: We conduct the first DFIA attack on AES in remote FPGAs. Instead of externally controlling the clock period, we inject faults with ROs. We demonstrate that ROs are qualified candidates for performing DFIA on AES. This work introduces a potential security threat on multi-tenant FPGAs.

2.4.3 Differential Analysis of On-chip Sensing and Locating

In Chapter 5, we evaluate a method to detect the location of a target circuit using TDC sensors on multi-tenant FPGAs. In this work, we apply differential analysis to Hamming Weight (HW) from TDCs, use contours to characterize how the locations and power consumption of the target circuits affect HW and explain how we use contours to predict the location of the target. In the experimental section, we assess the accuracy of prediction and the uniqueness of the impact by the target circuit on TDCs on AWS EC2 F1 FPGAs.

Research Contribution: We propose a method to predict a target circuit on multi-tenant FPGAs through differential analysis of TDC sensors. By knowing the location of the target, voltage-based fault attacks can be detected or an attacker can be guided to effectively target a victim to implement side channel attack.

2.5 Research Publications and Timeline

The research conducted during my Ph.D. since September 2018 (see Table 2.1) has resulted in six publications, listed below with their venues. As outlined in Section 2.4, the text of the dissertation is specifically focused on works culminating in the first three publications [52, 53, 54]. The other three publications [55, 27, 56], and my role in each, are briefly described below, in Section 2.5.2.

Table 2.1: Ph.D progress timeline; (**) denotes works that are the primary focus of this dissertation.

Description of Research Task	2018	2019	2020	2021	2022	2023
Anti-counterfeit IC package authentication [55]	*	*				
TRNG on cloud FPGA [52, 53]		**	**	**		
Comparing on-chip voltage sensors [27, 56]		*	*			
DFIA on AES in remote FPGAs [54]				**	**	
Differential analysis of on-chip sensors					**	**

2.5.1 List of Publications

1. X. Li, P. Stanwicks, G. Provelengios, R. Tessier, and D. Holcomb, "Jitter-based adaptive true random number generation for FPGAs in the cloud," in *2020 International Conference on Field-Programmable Technology (ICFPT)*. IEEE, 2020, pp. 112–119. [52]

2. ——, "Jitter-based adaptive true random number generation circuits for FPGAs in the cloud," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 16, no. 1, pp. 1–20, 2023. [53]

3. X. Li, R. Tessier, and D. Holcomb, "Precise fault injection to enable DFIA for attacking AES in remote FPGAs," in *2022 IEEE 30th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 2022, pp. 1–5. [54]

4. S. N. Dhanuskodi, X. Li, and D. Holcomb, "COUNTERFOIL: Verifying provenance of integrated circuits using intrinsic package fingerprints and inexpensive cameras," in *29th USENIX Security Symposium (USENIX Security 20)*, 2020, pp. 1255–1272. [55]

5. S. Moini, X. Li, P. Stanwicks, G. Provelengios, W. Burleson, R. Tessier, and D. Holcomb, "Understanding and comparing the capabilities of on-chip voltage

sensors against remote power attacks on FPGAs," in *IEEE 63rd International Midwest Symposium on Circuits and Systems (MWSCAS)*. IEEE, 2020, pp. 941–944. [27]

6. S. Moini, A. Deric, X. Li, G. Provelengios, W. Burleson, R. Tessier, and D. Holcomb, "Voltage sensor implementations for remote power attacks on FPGAs," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 16, no. 1, pp. 1–21, 2022. [56]

2.5.2 Research not Otherwise Covered in Dissertation

COUNTERFOIL [55] is an authentication technique based on the unique features of molded IC package surface, that is similar to physical unclonable functions (PUFs). The chip manufacturer extracts fingerprints of package surface features from chips by an image processing method and logs the fingerprints into a database. Then the customer extracts the features and authenticates the fingerprints at the end of supply chain by comparing the similarity with the fingerprints in the database. Our experiments demonstrate success in authenticating packages across 10 different chip models, including packages that are formed from the same physical mold. Additionally, we evaluate the resiliency, runtime, and performance under variations in lighting and magnification. My work on this project was to collect image data with cameras, test the resiliency, add digital signatures, and compare four image processing algorithms implemented within OpenCV.

I also contributed to a study comparing ROs and TDCs as voltage sensors on FPGAs, resulting in a conference [27] and journal [56] publication. The sensitivity and stability of each sensor are evaluated in the presence of power wasters. An important finding is that RO sensors and TDC sensors demonstrate agreement with each other when detecting voltage drops. TDCs show a higher resolution than ROs and are more suitable for short transient detection, while RO sensors require more runs to

average out the noise. However, RO sensors do not require calibration and are easier to implement. My contribution to this study and its publications [27, 56] was the design of the TDC circuits and a calibration method that allows voltage drops to be more accurately measured by TDCs. Subsequently, I have extensively used ROs and TDCs for my experiments in Chapters 3, 4, and 5.

CHAPTER 3

TRUE RANDOM NUMBER GENERATOR ON CLOUD FPGAS

In this chapter, we present and evaluate a true random number generator (TRNG) design that is compatible with the restrictions imposed by cloud-based FPGA providers such as AWS EC2 F1 [3]. Random numbers are fundamental to cryptographic systems and widely used for generating keys, nonces, and initialization vectors. The quality of randomness required in these applications necessitates the use of TRNGs. TRNGs exploit the inherent physical properties of the system in which they are embedded to generate statistically random and unpredictable numbers. This characteristic makes the outputs of a TRNG unpredictable even to an adversary that knows the current state of the circuit. Physical sources of entropy commonly used in FPGAs by on-chip TRNGs include thermal noise and clock or oscillator jitter. The randomness of the numbers created by TRNGs is typically evaluated using stochastic models and statistical tests [57].

TRNGs are widely used in real-world applications with differing throughput requirements, ranging from one-time pads [58] and keys for web authentication which have low throughput requirements to initialization vector (IV) generation for block ciphers in storage drives or internet protocol packets which can have higher throughput requirements. For example, data passed from a host to a self-encrypting storage drive needs encryption/decryption before being read/written [59]. In this case, the throughput and quality of the TRNG affect system performance, because the latency will increase if the IV generation speed can not keep up with the read/write speed

of the drive. To support applications such as these which require random numbers, several works have studied TRNG on FPGAs [60, 61, 62, 63, 64] in recent years.

FPGAs are increasingly being used in cloud-based systems for prototyping and acceleration, and to support secure soft processors [65] which require a source of random numbers. Yet to protect their infrastructure from malicious voltage attacks [44], cloud providers such as AWS impose restrictions on the types of circuits that are allowed on their FPGAs. Circuits that deviate from standard digital design flows, including logic-driven clocks and combinational loops as found in ring oscillators (ROs), are detected during bitstream compilation and disallowed from being loaded onto the FPGA [35]. This restriction causes difficulty in creating and characterizing jitter-based TRNG circuits for cloud applications. Despite this unique restriction on cloud-FPGAs, TRNGs on FPGAs must nonetheless be designed to work correctly when deployed across multiple instances, must be supported by a stochastic model to validate the entropy claims, and should be robust against external perturbations.

In this chapter, we present a TRNG design and validation procedure that is tailored around the restrictions of cloud-based FPGAs. Our design is able to harvest jitter without creating oscillators, applicable to multiple cloud-FPGA instances, and adaptable to differences in clocks. The design adjusts to changing environmental conditions and can be characterized without requiring ground truth delay measurements that are commonly obtained by counting oscillations. We make several specific contributions in this work:

- A TRNG for Virtex UltraScale+ FPGAs used in the cloud is detailed, implemented, and analyzed across numerous AWS EC2 F1 instances. The design, which is based on tunable delay chains and a TDC that harvests entropy from clock jitter, avoids primitives such as combinational loops that are common in TRNGs but disallowed by AWS and other cloud providers.

- A novel procedure is proposed for computing the min-entropy per sample using a stochastic model. The model empirically relates component delays to clock jitter by least-squares fitting. The delays that are independently computed by the model during entropy evaluation strongly correlate to the delays from the FPGA's own timing report, which supports the validity of the delay values that are inferred by our approach.

- Beyond our basic TRNG design, we introduce a new approach based on linkable sampling modules, that can be instantiated and connected together to increase the entropy per sample without requiring additional tuning. The new variant of the TRNG improves throughput and min-entropy according to the stochastic model by 250% and 270% respectively at a modest 17% increase in slice count by sharing the control unit and tunable delay elements across the linkable sample modules.

The result of this work has been published in the proceedings of IEEE International Conference on Field Programmable Technology [52] and an extended version of this paper has been published in ACM Transactions on Reconfigurable Technology and Systems (TRETS) [53].

The remainder of this chapter is structured as follows. Section 3.1 provides background on previous FPGA TRNG approaches. Section 3.2 describes the structure of our TRNG and modeling is discussed in Section 3.3. Section 3.4 evaluates and discusses the TRNG entropy, resilience, and costs. Section 3.5 introduces and evaluates the TRNG with linkable sampling module. Section 3.6 concludes the chapter.

3.1 Related Work

Several works on cloud security have shown the demand for TRNGs on cloud-FPGAs. Zeitouni et al. [66] propose a scheme to protect clients' IPs while checking

rogue circuits for cloud-FPGA vendors. Their scheme requires that the trusted shell on an FPGA has a TRNG to generate a nonce that is sent to the client to compute a proof of authenticity. Wolfe et al. [67] perform secret sharing Multi-Party Computation (MPC) on FPGAs in the datacenter. In the implementation of their MPC protocol on AWS FPGAs, a random key for each party needs to be generated and shared with one other party. The implementation of TRNGs in FPGAs has been widely studied, although none of the prior approaches comprehensively address the unique challenge of cloud FPGAs as well as the general TRNG requirements listed in Section 3.1. A large majority of these previous implementations rely on ROs to generate high-frequency signals that exhibit significant jitter. For example, Kohlbrenner and Gaj [68] use two ROs and a sampling circuit to measure jitter and Maiti et al. [69] deploy up to 128 ROs to amplify uncertainty. Some TRNGs augment ROs with delay paths to increase timing sensitivity. Like our approach, Rozic et al. [70] and Yang et al. [60] use carry logic-based delay chains to assist with entropy extraction. These approaches do not include tunable delays to combat environmental factors and an RO is used to excite the delays.

Several non-RO based TRNGs have been built for FPGAs, but they also have limitations that make them inappropriate for cloud deployment. Majzoobi et al. [62] use programmable delay lines built from lookup tables (LUTs) that can be difficult to characterize on a per-FPGA basis. Deák et al. [71] use the jitter from an on-FPGA phase locked loop (PLL) to create a TRNG using clock settings that are a challenge to replicate in a cloud setting. Perhaps the most similar TRNG approach to ours [72] uses a standard clock input, tunable delay buffers, and a delay path. However, unlike our approach, the delay path is made from a chain of LUTs which have variable logic and routing delays across stages.

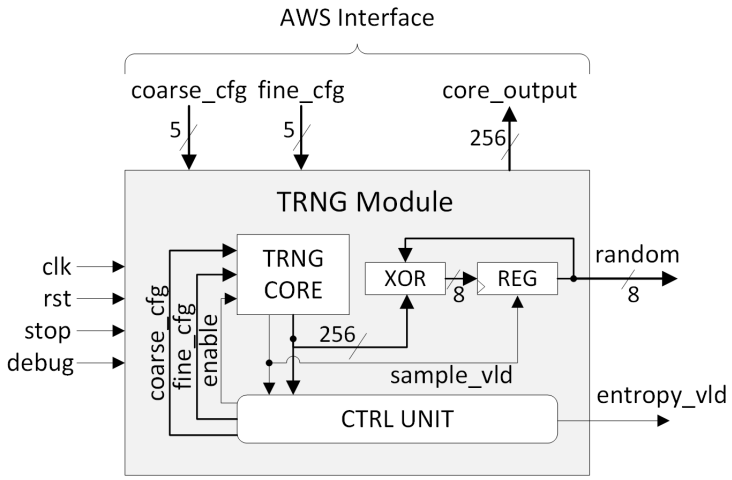
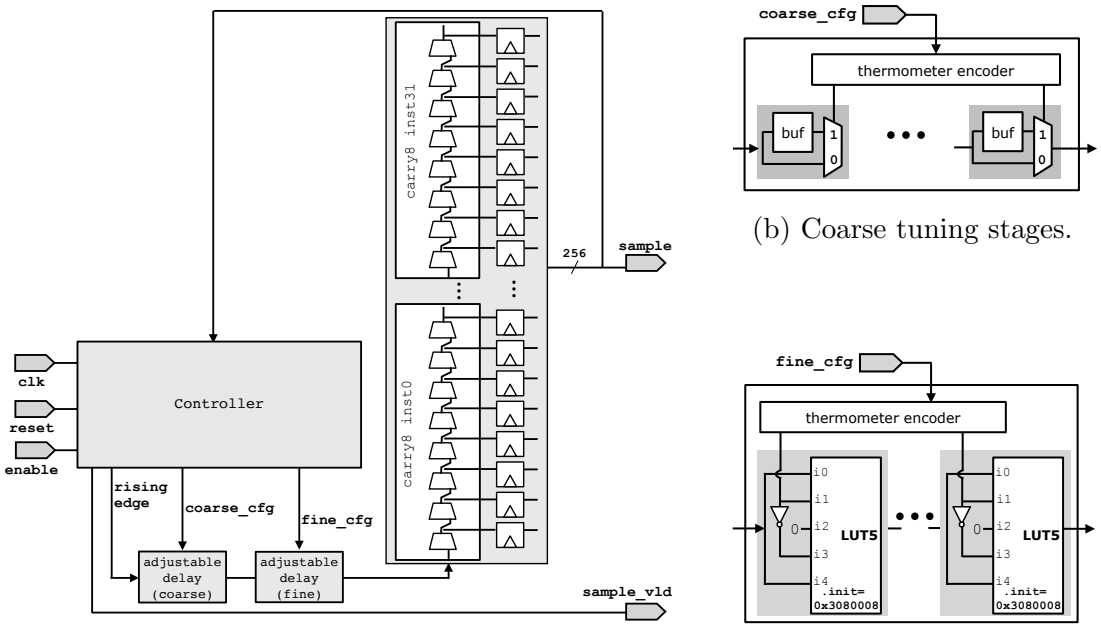


Figure 3.1: Structure of TRNG design and interface.

3.2 Structure of TRNG

Fig. 3.1 shows the top-level view of our TRNG design. It is a hardware module that serially generates 8-bit random numbers. We instantiate the TRNG module within a hardware testbench for analysis. The design is created for AWS EC2 F1 instances, which contain Xilinx Virtex UltraScale+ VU9P FPGAs. The bitstream is generated using Amazon's Hardware Development Kit (HDK), and then converted to an Amazon FPGA Image (AFI) that is reused for deployment across F1 instances. Amazon provides a runtime tool to interact with the deployed design by reading and writing 32-bit data to or from user-defined registers using AXI4 over PCIe. We make the signals at the top of Fig. 3.1 accessible to the runtime tool only when the TRNG hardware is set to debug mode. The debug mode allows us to control the TRNG and observe sample values from the TRNG core, which is useful for data collection and analysis in the cloud, but would be insecure if enabled in production.

Internally, the TRNG module gets entropy from the TRNG core, and hashes it into a local entropy pool by XOR operation. The control unit keeps a conservative estimate of the current entropy in the pool by counting the number of valid samples provided to it. Once enough entropy is collected, a signal from the control unit is



(a) TDC chain.

(b) Coarse tuning stages.

(c) Fine tuning stages.

Figure 3.2: Components of TRNG core and control unit.

asserted and the 8-bit random value in the registers can be read out, upon which the entropy count is reset to 0. We now describe in more detail the components of the TRNG core (Fig. 3.2).

3.2.1 Carry Chain Description

The TDC in our circuit (Fig. 3.2a) consists of 32 8-bit carry stages, and each output bit from the carry chain is the data input to a D flip-flop in the same slice. The controller repeatedly generates a single rising edge that propagates into the carry chain with an appropriate delay such that it will be propagating through the carry chain when the next rising clock edge occurs. The number of 1-values captured in the 256 flip-flops, i.e. the Hamming weight of the sample, is an indication of how far up the chain the rising edge has propagated by the time of the rising clock edge. The Hamming weight of samples will fluctuate slightly in each trial due to clock jitter, which is our source of randomness.

3.2.2 Tunable Delay Elements and Feedback Control

Our circuit uses tunable delay elements and feedback to ensure that the rising edge from the delay line is within the TDC chain when the clock arrives. The control unit measures Hamming weight by summing the sampled values captured in the flip-flops. Increasing the propagation delay will cause the rising edge to reach fewer TDC stages and thereby will reduce the Hamming weight of samples. Similarly, decreasing the delay will increase the Hamming weight. In this way, the tunable delay circuits are the knob used for adjusting the Hamming weight. The control unit uses the delay knob to position the rising edge in the TDC chain during clock arrival, as is required to generate randomness.

Ideally, the Hamming weight of the samples should be centered at around 128 in a 256-stage delay chain, which gives a maximum margin against delay changes in either direction that could detune the circuit. The control unit adjusts the coarse-tuning and fine-tuning settings for the next sample based on the Hamming weight of the current sample using the simple feedback scheme of Eq. 3.1, where (c, f) and (c', f') are the current and next values of the coarse and fine tuning settings, and HW is the Hamming weight of the current sample; note in Eq. 3.1 that f_{mid} represents the middle setting for fine tuning, which in our case is 15. Furthermore, with each sample, the control unit credits entropy to the entropy pool only when the Hamming weight is between 30 and 225 so that samples are not counted as random if the circuit becomes detuned and the rising edge is approaching either end of the carry chain where jitter may not be captured. Once enough samples are collected, the control unit asserts a signal to indicate the generation of an 8-bit random number is complete, which will be further explained in Section 3.2.3. During testing, the control unit is able to configure the TRNG tuning manually using values provided through AWS interface, and to return the resulting samples via the same interface.

$$(c', f') = \begin{cases} (c+1, f_{mid}) & \text{if } 208 \leq HW \\ (c, \quad f+1) & \text{if } 158 \leq HW < 208 \\ (c, \quad f-1) & \text{if } \quad 48 < HW \leq 98 \\ (c-1, f_{mid}) & \text{if } \quad\quad HW \leq 48 \end{cases} \tag{3.1}$$

The coarse and fine tuning stages are implemented as follows. Each coarse tuning stage adds or bypasses a LUT1 primitive that implements a logical buffer, as shown in Fig. 3.2b. Each fine-tuning stage selects between a shorter and longer pin-to-pin delay of a LUT5, where the enabled path through the LUT is set by the control input, as shown in Fig. 3.2c. The stages are controlled using thermometer encoding, so that incrementing or decrementing their configuration settings will change only one stage along the delay line, which helps ensure predictable control but has higher area cost than a binary-encoded tunable delay in which each stage has twice the delay of the next. Fig. 3.3 shows the Hamming weight of samples for all combinations of tuning; note that debug mode is used to generate this plot, as it overrides the feedback of the controller, and allows the samples from the TRNG core to be logged.

3.2.3 Post-processing Circuit

The 256-bit samples from the TRNG core are hashed into the 8-bit state of the entropy pool using a simple scheme as shown in Fig. 3.4. Each update includes a 1-bit circular shift of the 8-bit entropy pool, which ensures that randomness will get distributed through the 8 bits even if always coming from the same position in the 256-bit sample. We have used this particular scheme for simplicity, but it could be replaced with any number of other hash functions for the same effect. A counter tracks the number of valid samples produced by the TRNG core, and requires that 80 valid samples are hashed into the entropy pool before it is considered to be random, which assumes 0.1 bits of entropy per sample. The actual value of entropy per sample should exceed the assumed entropy per sample, but the specific value of 0.1 bits is

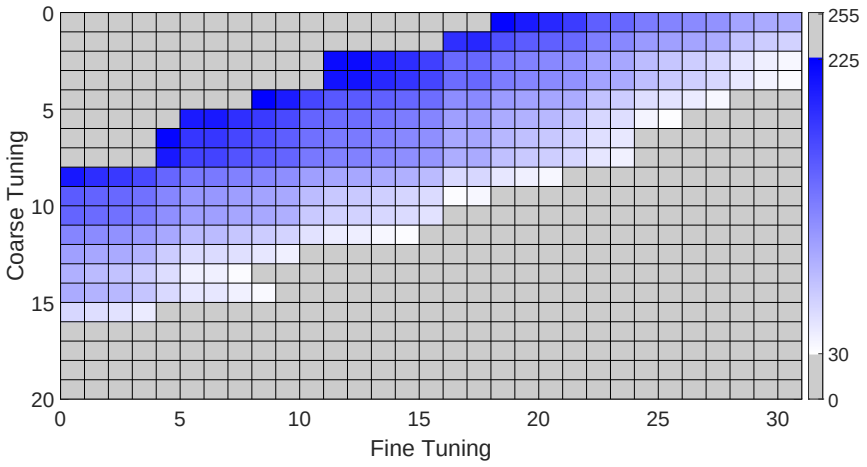


Figure 3.3: Heatmap showing the Hamming weight of samples on a single F1 instance for all possible tuning settings. Increasing the coarse or fine tuning reduces the Hamming weight. Sampled values indicative of a poorly-tuned TDC, colored gray, would not cause the entropy count to be incremented.

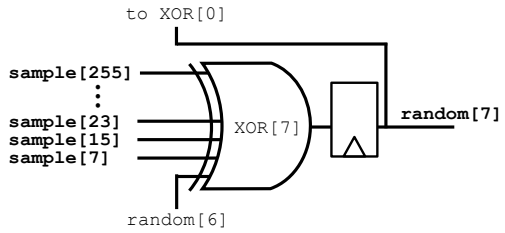


Figure 3.4: 256-bit samples are hashed into the 8-bit random signal which is the entropy pool. The hashing uses eight XOR gates, all configured in the same manner as the one that is shown.

chosen somewhat arbitrarily as a conservative assumption. If the entropy per sample is assumed to be higher, then the circuit will need fewer valid samples before deciding that the TRNG has accumulated enough entropy to produce an 8-bit random output. This assumption, therefore, has an impact on the throughout of the TRNG. Section 3.3 of the chapter will show that the actual entropy per sample exceeds this conservative estimate by a factor of more than 2 using both a stochastic model and NIST tests.

3.3 Stochastic Model

The randomness of the TRNG comes from the samples of the delay line in the TDC. Even if the propagation delay of the delay line does not change across trials, the TDC can produce different samples if it has fine enough time resolution and its sampling clock has sufficient jitter. The time resolution on the TDC is a consequence of the low propagation delay of the stages in the hard carry chains of Xilinx Ultrascale+ devices. A larger clock jitter or finer time resolution will both have the same effect of making the TDC samples more random because both changes will make the jitter relatively larger in comparison to the TDC time resolution. Accordingly, the relevant consideration for modeling a TRNG such as ours is to quantify how the amount of jitter compares to the time resolution of the TDC.

In our stochastic model of the TRNG, we relate jitter to TDC time resolution without relying on conservative timing reports or jitter estimates. We rely in modeling only on the simplifying assumption that jitter is normally distributed, which is in particular consistent with the jitter component caused by thermal noise [73, 74]. We also assume its standard deviation is invariant with respect to the tuning settings of the delay line. From this, we calculate the time resolution of each TDC stage in terms of the standard deviation of jitter, which we use as the unit delay in our model. After calculating the time resolution of each stage, we use the same model to calculate a lower bound on min-entropy per sample. The next subsection describes our modeling approach.

The bits produced by any TRNG design should be as random as the bits produced by any other, unless one of the designs is fundamentally flawed. The stochastic model assures the randomness by validating the claimed entropy of the source. If a source has lower entropy, its TRNG will still produce fully random outputs, but it will require more samples from the source to accumulate the required amount of entropy for each random output value.

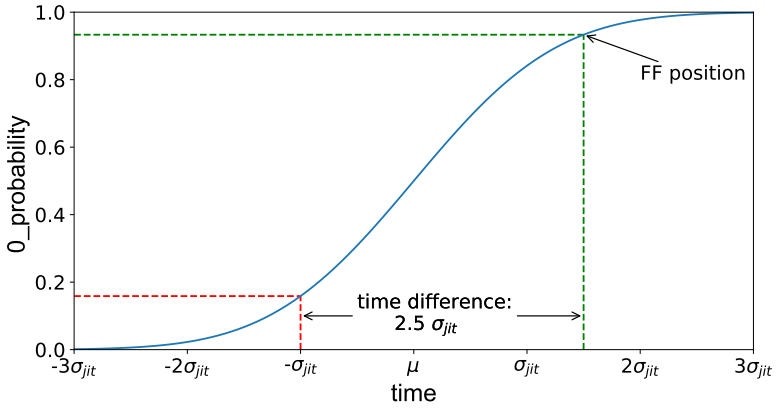


Figure 3.5: TDC position in normal distribution CDF for modeling.

3.3.1 Empirical Model Relating TDC Delay to Jitter

In a given trial, the flip-flop associated with each TDC stage will sample a 0 value if its clock arrives before its rising data input from the delay line, and will sample a 1 otherwise. If the delay tuning settings are held constant across samples, the 0-probability (1-probability) of the stage indicates the proportion of trials in which its clock arrives before (after) its rising data input from the delay line.

If the flip-flop of stage i samples 0 with probability 0.159, then 15.9 percent of clock edges arrive there before the rising data input, and 84.1 percent of clock edges arrive after. Under the assumption that jitter is normally distributed, the observation that 15.9 percent of clock edges arrive before the rising data input reveals that rising data input coincides with the clock being $-1.0 * \sigma_{jit}$ away from its mean value, because $\Phi^{-1}(0.159) = -1.0$, where Φ^{-1} is the inverse CDF of a normal distribution. This scenario is depicted graphically in Fig. 3.5.

In these same trials, if the flip-flop of stage j samples 0 with probability 0.933, then we similarly conclude that its rising data input coincides with its clock being $+1.5 * \sigma_{jit}$ away from its mean because $\Phi^{-1}(0.933) = 1.5$. If there is no clock skew between the flip-flops of i and j, then these two findings together indicate that the time difference between the rising data inputs on i and j is equal to $2.5 * \sigma_{jit}$. If

clock skew is allowed, then we generalize the claim slightly to more formally conclude only the difference in criticality (i.e. timing slack) between the two flops is equal to $2.5 * \sigma_{jit}$, although for our purposes it is actually the criticality that matters so we need not worry about skew. The delay or criticality difference between any two stages can therefore be estimated from their 0-probabilities in a set of trials. Because the estimate is noisy when the associated 0-probabilities are close to 0 or close to 1, we apply it only when the 0-probabilities indicate that both stages are within $\pm 2\sigma_{jit}$ of their means. Note that the delay difference is being calculated in units of σ_{jit} even though the value of σ_{jit} is not known in absolute terms. Using this approach, two flip-flops i and j have arrival times (denoted T_i and T_j) that are related as shown in Eq. 3.2, where $\widehat{P}_i$ and $\widehat{P}_j$ are their respective 0-probabilities for a particular tuning setting.

$$T_i - T_j = \left(\Phi^{-1}(\widehat{P}_i) - \Phi^{-1}(\widehat{P}_j)\right) \sigma_{jit} \tag{3.2}$$

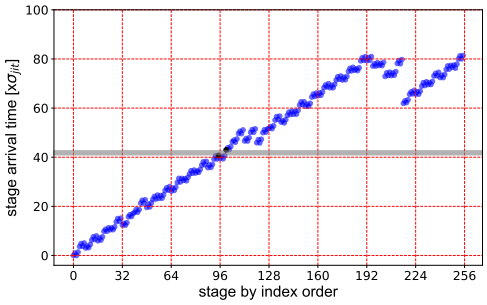
3.3.2 Calculating Stage Arrival Times

Given that 0-probabilities from each tuning setting will relate the arrival times of some of the TDC stages, and that the same stages can be related to each other by multiple different tunings, we can solve a set of equations to obtain the arrival time T_i for each stage i . The set of equations is as described in Eq. 3.3 where T is the n-element column vector of unknown arrival times, A is the m-by-n matrix of coefficients in which all entries are 0 except for a single +1 and single -1 in each row for the two stages that are related, and B is an m-by-1 column vector of the arrival time differences, calculated as shown on the right hand side of Eq. 3.2. We then find the least-squares solution to $AT = B$ (Eq. 3.3) which gives stage arrival times in terms of σ_{jit}. Because the formulation deals with differences in arrival times, we adopt the

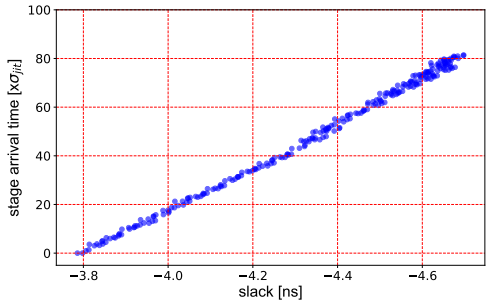
convention that T_0, the arrival time of the first stage, is 0; other T_i values therefore represent the arrival time of stage i relative to the first stage.

$$AT = B\sigma_{jit} \tag{3.3}$$

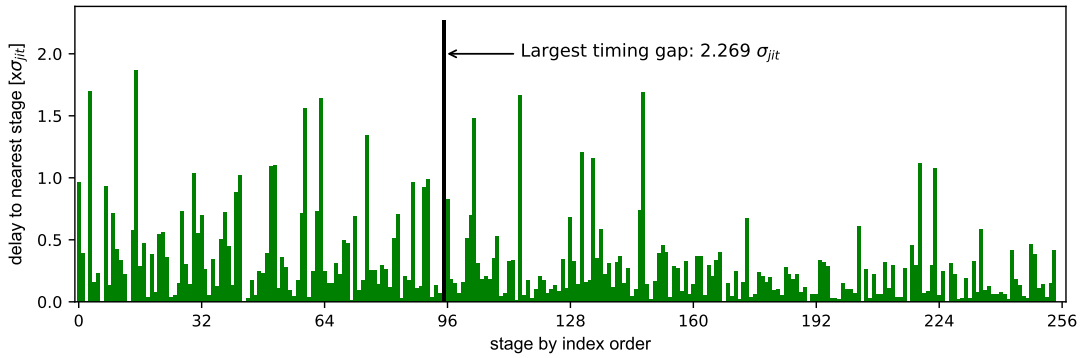
Fig. 3.6a shows, for one instance of the TRNG, the stage arrival times $(T_0, \ldots, T_{255})$ obtained by solving Eq. 3.3. The arrival times across the 256 stages cover a span of approximately 84 times σ_{jit}. While the arrival time generally increases while moving up the TDC chain, note that the trend is not smooth. Although the rising edge does propagate through the carry chains in sequential order, the anomalies in this trend imply that it does not reach the d input of the flip-flops in sequential order. This can occur because the delays between carry chain and flip-flop of each stage are not uniform, and because clock skew at the flip-flop aliases to delay on its data input; both of these artifacts are captured in the timing report so it is instructive to compare the modeled arrival times to the slack from the timing report. Fig 3.6b shows the same modeled arrival times from Fig. 3.6a, but now plotted against the reported timing slack for the corresponding flip-flop which accounts for all delays and clock skew. Because a span of 84 times σ_{jit} corresponds to slightly above 800ps of reported slack, we can estimate σ_{jit} to be around 10ps in absolute terms per the timing report, although the timing report itself uses a conservative delay with built-in safety margin, so the average-case delay may be somewhat less. There is a high correlation $(r = -0.997)$ between the arrival times from the model and the reported slack. Given that the model was fitted to data measured on the board, the high correlation between the two quantities supports the validity of the model for correctly resolving on-chip delays, and hence also for capturing the difference in criticality between stages. Now that the empirical timing model is validated, we use it as the basis for estimating the worst-case entropy of our TRNG.



(a) Modeled arrival time vs stage index.



(b) Modeled arrival time vs timing slack.



(c) Precision of each stage in TDC chain from one FPGA instance, obtained from characterization procedure.

Figure 3.6: TDC characterization. The largest timing gap between any two neighboring stage arrival times is highlighted and annotated in Fig. 3.6a and Fig. 3.6c

3.3.3 Stochastic Model for Entropy Estimation

Based on the precision of each stage in Fig. 3.6c, we obtain the largest timing gap ($2.268 * \sigma_{jit}$) between any two neighboring arrival times as highlighted in Fig. 3.6, which can be used to estimate a lower bound on min-entropy of the samples. The worst-case min-entropy corresponds to the sampled value that can be produced with the highest probability. This would occur when the mean arrival time of the clock coincides with the center of the largest timing gap of any stage, which we denote here as Δ_{max}. This is illustrated in Fig. 3.7, which depicts clock jitter assumed as normal distribution. In the worst-case min-entropy, the mean of jitter is in the middle of the two stages with $\Delta_{max} = 2.268 * \sigma_{jit}$. The shaded region represents all the clock arrival times that would result in the same sample being produced. The probability of producing this sample would then be the probability associated with the shaded region, which we denote as P_{max}, and calculate from the normal CDF as in Eq. 3.4. The min-entropy of an outcome with probability P_{max} is given by Eq. 3.5. For the specific instance used to generate these results, the largest interval is $2.268 * \sigma_{jit}$ shown in Fig. 3.6c, which corresponds to a shaded area in Fig. 3.7 with P_{max} of 0.743, and hence min-entropy of 0.429 bits.

$$P_{max} = \Phi\left(\frac{\Delta_{max}}{2}\right) - \Phi\left(\frac{-\Delta_{max}}{2}\right) \tag{3.4}$$

$$entropy_{min} = \log_2\left(\frac{1}{P_{max}}\right) \tag{3.5}$$

3.3.4 Impact of Routing and Clock Skew on Entropy

The previous subsection explains that worst-case min-entropy is limited by the largest timing gap among all the stages. It is therefore desirable to make all of the timing gaps uniform so that none are unusually large. We now present further results

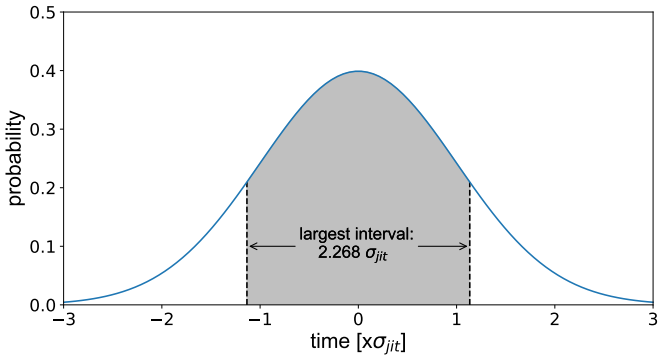
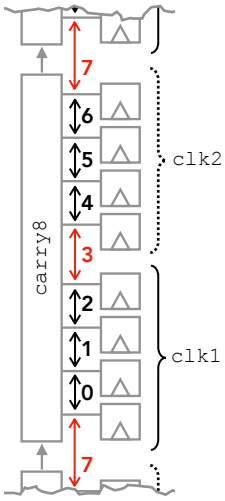


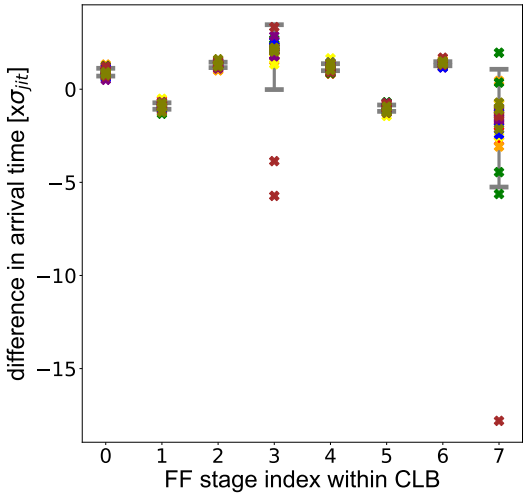
Figure 3.7: Largest share of clock arrival times that will cause TDC to sample the same value.

and discussion to explain why routing makes this objective difficult to accomplish in practice.

Fig. 3.8b shows the difference in arrival time between the FF of each stage and that of the next stage by index. Here, instead of using indices from 0 to 255 to represent the stages across all 32 CLBs as was done in Fig. 3.6a, we use indices 0-7 for each CLB as annotated in Fig. 3.8a, and have occurrences of each index from all 32 CLBs. Fig. 3.8b shows, for each stage index, the 32 differences in arrival time between that stage and the next. The differences in arrival time are predictable for stages 0, 1, 2 and for stages 4, 5, 6. In stages 3 and 7, the arrival time difference is inconsistent from CLB to CLB. This inconsistency occurs because Ultrascale+ uses different clock inputs for the upper and lower halves of the CLB, which causes stages 3 and 7 to span two different clock leaf nodes (clk1 and clk2 in Fig. 3.8a); the skew between the clock leaf nodes aliases to arrival time as discussed in Section 3.3.2. For the TRNG design, one must therefore be careful to avoid large positive clock skew at these points as it can reduce worst-case entropy by causing highly probable outcomes for certain unlucky conditions. Negative skew causes no such problem, as can be observed in Fig. 3.6a, so the one-sided restriction on skew is easy to satisfy in practice. In fact, note that the negative skew at stage 208 in Fig. 3.6a can improve the quality of the



(a) CLB stage indices.



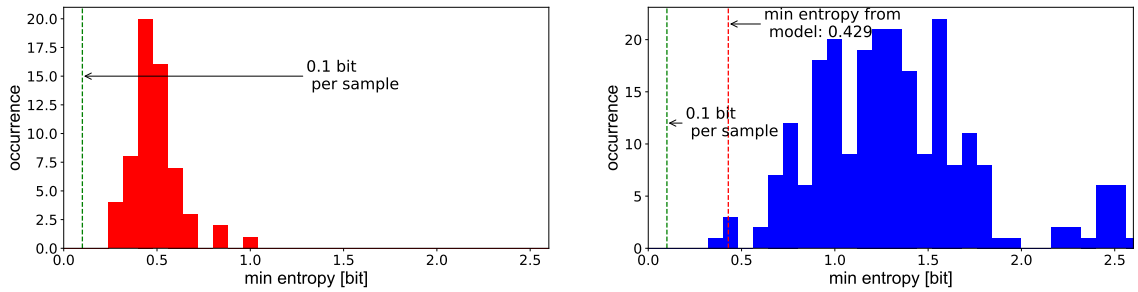
(b) Differences in arrival times by index.

Figure 3.8: Across the 32 CLBs in the TDC, the difference in arrival time between one index and the next is predictable for indices 0, 1, 2 and 4, 5, 6. Indices 3 and 7 are each followed by a stage that is on a different clock leaf, and there the difference in arrival time is inconsistent due to clock skew. Error bars extend one standard deviation from the mean.

TRNG, because it causes the rising edge to be captured twice in the same sample, at different positions in the chain. In Section 3.5 we build on this principle to design a TRNG that samples the rising edge multiple times.

3.4 TRNG Quality Evaluation

In this section, we use three different techniques to test the quality of the random numbers produced by our design. As described in the following three subsections, the results support 1. that our design exceeds the 0.1 bits of min-entropy per trial that was assumed as a security parameter; 2. that our stochastic model gives a reasonable estimate of min-entropy; and 3. the random numbers generated pass tests for statistical randomness.



(a) Entropy per stochastic model for 60 FPGA instances.

(b) Entropy per NIST SP800-90B for 234 tunings on one instance.

Figure 3.9: Min-entropy by both stochastic model and NIST SP800-90B suite exceeds 0.1 bits per sample.

3.4.1 Stochastic Model Applied Across EC2 F1 Instances

The stochastic model from Section 3.3.3 is our primary strategy for estimating the worst-case min-entropy for each single instance of the TRNG. To test across FPGAs, we load the same bitstream onto 60 different EC2 F1 instances, and on each machine apply our characterization procedure to evaluate the worst-case min-entropy. The distribution of calculated min-entropy values (Fig. 3.9a) ranges from 0.250 to 0.972. These values indicate that across all 60 instances our design exceeds, by at least a margin of 2.5×, the 0.1 bits of min-entropy per sample that was assumed.

3.4.2 Stochastic Model vs. NIST Entropy Assessment

Next, to check our stochastic model, we apply the NIST SP800-90B entropy assessment suite [75] to obtain an independently calculated estimate of min-entropy. To generate data for the NIST assessment, we apply on one instance all tuning settings and collect 1,000,000 samples from the TDC with each setting applied [76]. We keep the data from any settings in which the average Hamming weight of samples is in our allowed range of 30 to 225, which corresponds to a total of 234 tuning settings. The NIST assessment is applied separately to each of these 234 datasets, and the distribution of results is shown as a histogram in Fig. 3.9b. The NIST estimate of

min-entropy for most of the tuning settings fall above the estimate of 0.429 bits per sample from the stochastic model on this instance. Because the NIST entropy values tend to exceed our estimated worst-case, we gain some confidence that our stochastic model is not overestimating entropy.

3.4.3 End-to-End NIST Statistical Tests

Although the evaluation of the entropy source in the prior subsections is the primary validation for a TRNG, we also apply statistical tests to the post-processed 8-bit values produced by the TRNG as a further validation. The NIST Statistical Test Suite [77], which is widely used with random number generators, applies a collection of statistical tests and for each test reports whether the sequences of bits are consistent with being random. The report shows how often the P-values from each test fall within uniformly sized bins C1 through C10, and should tend toward being uniformly distributed when enough random data is tested. The test suite is applied to a dataset comprising 100 sequences of 1,000,000 bits from the TRNG and the results are displayed in Table 3.1. The final column of the table shows the proportion of sequences that pass the test, indicating that the sequences have statistical properties consistent with being random.

3.4.4 Empirical Min-entropy with Respect to Lag

Our stochastic model provides an upper bound on the ability to predict the response of the TRNG circuit, but the model assumes that all variation in the data is explained by temporally uncorrelated jitter. If variation in the characterized data actually has temporal correlation due to a physical cause like temperature or voltage, then the model may overestimate jitter. As an additional validation of entropy, we evaluate empirically whether an attacker that knows the value of a leading sample can predict the value of a lagging sample a few cycles later. We perform this evaluation for all valid tuning settings, and collect a dataset of 100,000 samples for each. From each

Statistical tests	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	P-value	%
Frequency	10	12	6	12	11	4	11	14	4	16	0.091	98
BlockFrequency	15	17	8	7	7	7	12	9	6	12	0.163	99
CumulativeSums	13	6	10	7	10	15	13	9	9	8	0.596	99
CumulativeSums	11	8	13	8	12	7	15	8	11	7	0.637	98
Runs	9	8	8	8	16	6	15	15	7	8	0.172	98
LongestRun	9	12	8	11	16	6	9	10	11	8	0.657	98
Rank	9	12	13	10	12	5	11	9	13	6	0.637	100
FFT	9	12	12	15	8	5	11	10	12	6	0.494	100
NonOverlap. Template	8	4	7	11	13	9	15	13	10	10	0.401	100
Overlapping Template	8	9	11	8	8	15	12	9	8	12	0.817	98
Universal	8	8	8	12	14	12	14	6	8	10	0.616	99
Approximate Entropy	18	10	12	6	8	14	6	12	6	8	0.109	99
Serial	5	10	8	9	11	12	10	9	16	10	0.616	99
Serial	6	8	10	11	6	13	11	10	12	13	0.740	100
LinearComplexity	13	8	4	14	13	7	15	9	8	9	0.249	100

Table 3.1: Results from applying NIST statistical test suite to 100M generated bits shows that the TRNG outputs are evaluated as consistent with being random.

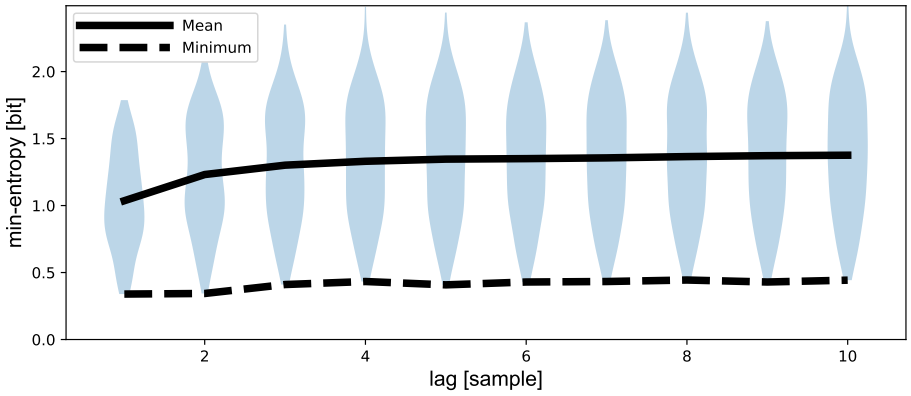


Figure 3.10: Distribution of min-entropy values, with annotations for mean and minimum, when guessing a lagging sample based on the value of a leading sample. There is only a minor temporal relationship, and min-entropy remains above 0.33 even in the worst case.

Work	FPGA type	Throughput (Mbps)	Utilization (slice)	Efficiency (Mbps/slice)	Approach	Testing method	Analysis of attacks resisted	Entropy validation Entropy value Entropy type
[78]	Spartan 6	100	46	2.17	self-timed ring	NIST SP800-22	-	-
[71]	Spartan 6	14.3	67	0.213	ring oscillator	TestU01 DIEHARD NIST SP800-22 ENT	-	ENT 7.99998/byte (postproc) unknown
[60]	Spartan 6	1.15	3	0.383	ring oscillator	NIST SP800-90B AIS-31	-	NIST SP800-90B 0.76/sample (raw) min-entropy
[79]	Virtex-4	12.5	580	0.022	RS latches metastability	DIEHARD NIST SP800-22	-	-
[61]	Virtex-6	50	224	0.223	timing non-uniformity	DIEHARD NIST SP800-22	-	-
[62]	Virtex-5	2	32	0.063	metastability	NIST SP800-22	Unspecified external perturbations	-
[63]	Spartan 6	3.3	27	0.122	ring oscillator	AIS-31	-	stochastic model >0.91/bit (raw) min-entropy
[64]	Spartan 6	1.1	128	0.122	ring oscillator	AIS 20/31	-	AIS-20/31 7.998265/byte (raw) Shannon entropy
[80]	Spartan 6	0.76	1	0.76	latched ring oscillator	NIST SP800-22 AIS-31	Voltage	AIS-31 7.99834/byte (raw) Shannon entropy
[81]	Spartan 3	6	270	0.022	ring oscillator	NIST SP800-22 AIS-31	-	AIS-31 7.9946/byte (raw) Shannon entropy
[82]	Virtex-6	-	9	-	self-timed ring	NIST SP800-22 NIST SP800-90B AIS-31	Power and Thermal Attacks	NIST SP800-90B 7.8869/samp (postproc) min-entropy
our basic TRNG [52]	Virtex UltraScale+	2.43	184	0.013	clock jitter	NIST SP800-22 NIST SP800-90B	PVT	NIST SP800-90B 0.37/sample (raw) min-entropy
TRNG w/ linkable modules	Virtex UltraScale+	6.08	216	0.028	clock jitter	NIST SP800-22 NIST SP800-90B	PVT	NIST SP800-90B 1.45/sample (raw) min-entropy

Table 3.2: Comparison with related TRNGs implemented on Xilinx FPGAs.

dataset, for all Hamming weight values a and b, we find the conditional probability of observing b as the value of the lagging sample, given that a was observed as the value of the leading sample. The maximum conditional probability corresponds to the best lagging sample guess by the attacker, when a leading sample has the value that most benefits the attacker. Min-entropy is calculated from this probability, and this is the empirical lower bound on min-entropy for the tuning setting that generated the 100,000 sample dataset. We obtain one such min-entropy for each tuning, and vary the lag from 1 to 10 samples, and show the results in Fig. 3.10. Across all the tuning settings, the minimum min-entropy never drops below 0.33 bits which still surpasses our security assumption of 0.1 bits per sample.

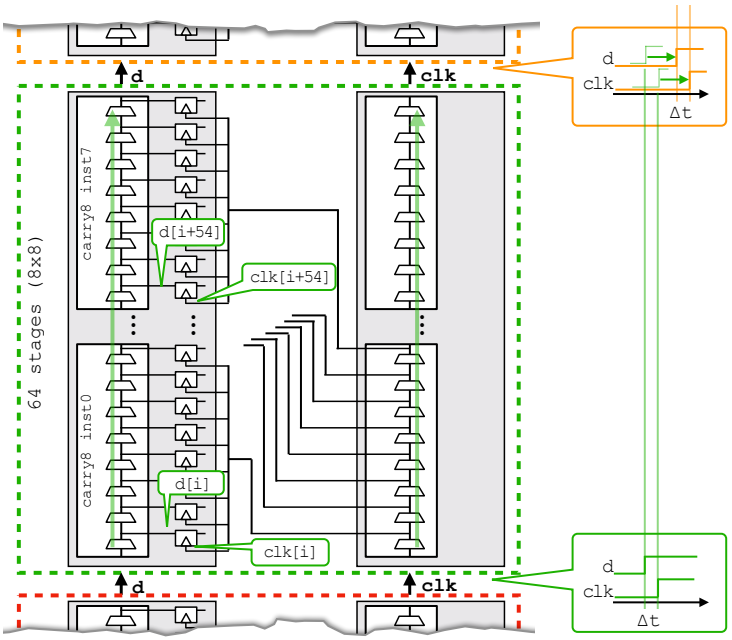
3.4.5 Comparison to Prior Work

The distinguishing feature of our work is its suitability for, and deployment on, cloud FPGAs. As we have described, this imposes limitations on the types of circuitry that can be used, and increases the importance of the TRNG being robust to environmental changes. Despite these challenges, the costs of our TRNG are found to be reasonable for a large cloud FPGA. Table 3.2 compares the throughput, logic utilization, efficiency (throughput/slice), testing methods, resistance to attack and entropy of our TRNG to other recently published TRNGs that are implemented on Xilinx FPGAs. Our TRNG design (Fig. 3.1) consumes 791 LUTs (0.067% of available), 33 CARRY8s, and 559 flip-flops (0.024%) across a total of 184 slices. Among these resources, the controller logic that configures the coarse- and fine-tuning consumes 92 LUTs and 34 flip-flops, while the remainder of the resources are consumed by the TRNG core itself. Our design generates random numbers at a rate of 2.43 Mbps, which is sufficient for most applications, but could be increased through parallelization if needed.

Here we listed the entropy comparisons between our work and other works in Table 3.2. It is worth noting that NIST SP800-20B does not list a minimum value of what constitutes a usable amount of min-entropy per sample.

3.5 Linked Sampling Module

In this section, we extend our basic TRNG to increase entropy per sample while still retaining the stochastic model of randomness. The key idea of the approach is to make a modular version of the sampling chain that can be arbitrarily extended by abutment to increase entropy. We choose to make each sampling module 64 stages in length. The new module is shown within the dashed box of Fig. 3.11a; we instantiate and link four such modules to create the sampling chain for the TRNG. Each module has inputs for a rising data edge and a sampling clock (shown at bottom), and then



(a) Linkable sampling module for TRNG.

	Arrival time [ns]		
Stage	data	clock	skew
i+54	0.368	0.300	-0.068
i+48	0.327	0.332	0.005
i+40	0.286	0.318	0.032
i+32	0.245	0.296	0.051
i+24	0.204	0.262	0.058
i+16	0.163	0.255	0.092
i+8	0.122	0.221	0.099
i	0.081	0.227	0.146

(b) Arrival time relative to module inputs

Figure 3.11: Schematic and timing of 64-stage linkable sampling module used in the TRNG

propagates those signals through parallel carry chains to the outputs at the top of the module. In the first instance, the data input is the rising edge from the tunable delay line as in Fig. 3.2a, and the clock input is attached to the 125 MHz system clock.

3.5.1 Timing Analysis

The simple timing diagrams shown at right in Fig. 3.11a illustrate the operating principle of the design. The matched paths taken by data and clock through the module ensure that whatever timing difference exists between them at module input, is preserved at module output, which is the input to the next module. Therefore, neglecting small imbalances, each module performs the same sampling experiment with the same relative timing. Effectively, the same rising edge is being sampled

within each module. We first consider the behavior of a single module and then how the four modules interact.

3.5.1.1 Single module

The 64 stages of a sampling module span 8 rows. The sampling clock for the 8 FFs in each CLB comes from one tap on the clock path. Relative to the d and clk module inputs, the relative arrival time at the data and clock inputs of sampling FFs are shown in the table of Fig. 3.11b. The position of the clock tap, and its routing to the sampling FFs, is fixed but not optimized. The sampling clock has a different arrival time at each CLB, unlike the original design where the clock tree ensures low skew. Recall from Section 3.3.2 that skew aliases to delay; therefore, even though the propagation delay of data through the 64 stages is reported to be 368 ps per the timing report, the difference in skew is only 214 ps because the clock arrives later at upper stages. That skew is likely the reason that the empirical delay difference through 64 stages is equivalent to around $20 \times \sigma_{jit}$, which is less than was observed in the original design where the clock was synchronous to all stages. Plotting arrival time of each stage against reported timing slack (Fig. 3.12b), we can see that the inferred arrival times of each module are correlated to reported slack with correlations of (0.97, 0.96, 0.97, 0.96), which are slightly lower than the correlation of 0.997 reported in Section 3.3.2.

3.5.1.2 Multiple modules

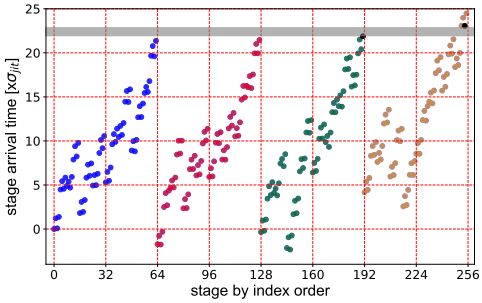
We now consider the timing behavior of all 256-stages, comprising four identical 64-stage modules. When the rising edge is sampled four times, it lands between two stages in each of the four modules. The samples of each module overlap each other, which therefore increases sensitivity and tends to reduce the bin widths as shown in Fig. 3.12c. The worst-case min-entropy calculated by applying the stochastic model to Fig. 3.12c is 1.152 bits, which is 2.7× larger than the min-entropy calculated for our

basic TRNG design. Accordingly, we increase our assumption of entropy from 0.1 bits per sample in the basic TRNG to 0.25 bits per sample in the TRNG that uses four linkable sampling modules, noting that our assumption remains highly conservative relative to worst-case entropy indicated by the model. When considering empirical arrival time from our model against slack, the expected correlation within each module can be observed in Fig. 3.12b, but there is an offset across modules. The reason is as follows. The relative timing of data and clock is the same at the input of each module due to the matched paths, yet timing analysis is conservative and considers adding the same delay to data and clock as making the path more critical. To preserve the same amount of negative slack, the timing analysis would require the added delay on the data path to be larger than the added delay on the clock path.

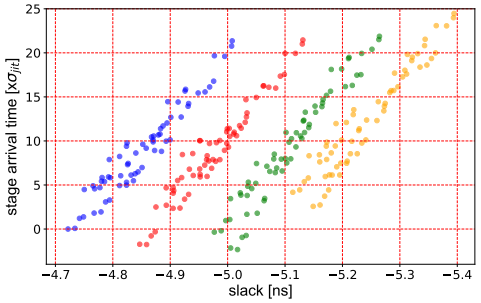
When the feedback control is implemented based on the total Hamming weight, there can be unpredictable changes if one instance entirely misses a rising edge due to poor tuning, so we instead control the delay based on the Hamming weight of a single 64-bit module instance, which also simplifies the control logic. Although we use four sampling modules to keep the total number of stages to 256, in principle an arbitrary number of sampling modules can be instantiated and connected by abutment. If there is systematic variation between data and clock paths, then it could perhaps eventually become difficult to use a common tuning for many instances. There are no indications of any such problems with four modules.

3.5.2 Entropy and Performance

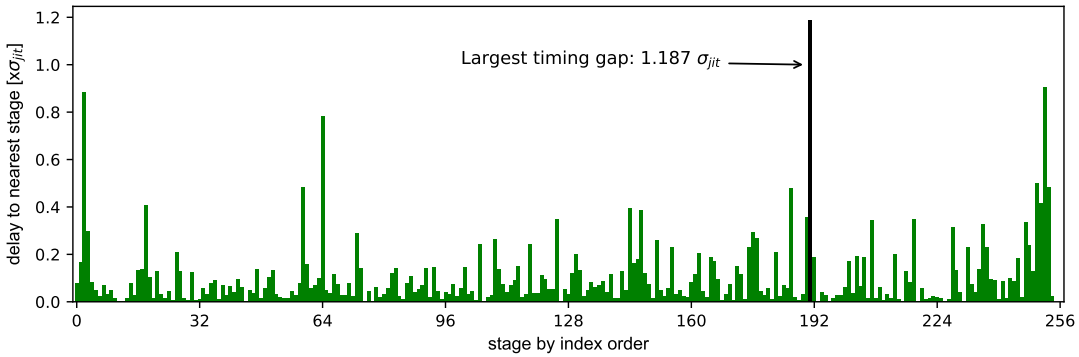
Similar to Section 3.4.1 for the basic TRNG, we load the modified TRNG onto 60 EC2 instances and apply the stochastic model to calculate the min-entropy on each instance. As shown in Fig. 3.13a, the worst case min-entropy of any instance is 0.93 bits. As before, the entropy assumption is conservative; the actual entropy from the model (0.93 bits/sample) is 3.72× higher than what is assumed (0.25 bits/sample).



(a) Arrival time at each stage.



(b) Arrival time vs slack.

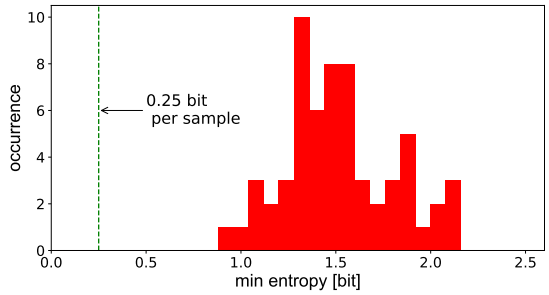


(c) Bin sizes, which determine entropy.

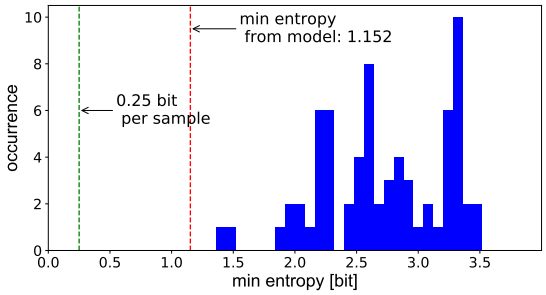
Figure 3.12: The use of four linkable sampling modules causes the samples to overlap in time (in a and b), so that the same edge is sampled once in each of the four modules. This reduces the bin size (in c) compared with the original TRNG. The largest timing gap between two neighboring stages in terms of arrival time is highlighted and annotated in Fig. 3.12a and Fig. 3.12c

As in Section 3.4.2, we also collect 1,000,000 samples from each delay tuning on one instance, and apply the NIST SP800-90B entropy assessment to the data. The distribution in Fig. 3.13b shows the min-entropy for all tunings on this instance, all of which exceed the 1.152 bits calculated by the stochastic model for the same instance.

Our justified assumption of 0.25 bits of entropy per sample enables a 250% increase in throughput relative to the basic TRNG. An 8-bit random number is now generated using only 32 samples instead of 80. Due to the 32 extra CLBs used to route the clock through the four instances of the linkable sampling module, the 250% increase



(a) Min-entropy of 60 instances, from stochastic model.



(b) Entropy assessment of 70 tunings, on one instance.

Figure 3.13: Entropy according to stochastic model, and from NIST assessment, both show that the modified TRNG with linkable sampling modules produces around 1 bit of entropy per sample.

in throughput comes at only 17% resource cost. The linkable sampling modules create an attractive cost vs performance tradeoff. Table 3.2 compares the performance of our TRNGs against eleven other works. The throughput of a linkable sampling TRNG is able to support low-throughput applications aforementioned in Section 3.1. The throughput of any FPGA TRNG can be increased to accommodate high throughput applications by adding more instances, so an important metric to consider is the efficiency, in terms of throughput of random numbers per slice of area. We therefore list the metric of Mbps/slice as well in the table. Finally, we list some of the attacks that each design is claimed to resist, and describe the entropy metrics that are provided for each. Notably, given that most FPGA TRNGs are based on oscillators which are forbidden, the only designs in the table that can be implemented on EC2 F1 are ours and [62, 61].

3.6 Chapter Summary

Cloud FPGAs are commonly used for accelerating computationally expensive cryptographic operations that rely on the generation of random numbers. In this chapter, we introduced and evaluated a TRNG design that is compatible with the

design restrictions imposed by cloud-based FPGA providers. The oscillator-free TRNG design that we propose uses a controllable delay and harvests clock jitter as an entropy source using a circuit that is similar to a TDC. The effectiveness of the design is supported by NIST test results and a stochastic model of the entropy source. Future work can consider further increases in entropy-per-sample and the impact of advanced clocking features.

CHAPTER 4

REMOTE DIFFERENTIAL FAULT INTENSITY ANALYSIS ON AES

Fault attacks (FA) [83, 84, 85] in FPGAs can occur when excessive on-chip switching stresses the power distribution network (PDN), inducing a voltage drop that causes timing faults [86, 87]. Such attacks can happen remotely, without user access using a variety of power wasting circuits [20, 44]. For example, Mahmoud et al. use ROs to create timing faults in random number generators [18].

Fault and side channel attacks have previously been used to extract encryption keys from FPGAs via differential fault analysis (DFA) [20]. However, differential fault intensity analysis (DFIA), which can recover keys by differential analysis on possibly-faulty ciphertexts without requiring the acquisition of correct ciphertexts, has not been explored in the context of remote FPGAs. DFIA [88, 89] induces faults with clock glitches. Recognizing that clock glitching and voltage drops both cause delay faults, in this work we implement a DFIA attack on AES in an FPGA by using a variable number of RO power wasting circuits to cause supply voltage drops. The voltage drop can be shaped by adjusting the number of ROs and the duration for which they are active. The ability to control the shape of a voltage drop is critical for DFIA, which requires varied faults in the 9th round of AES [88] to extract the secret keys. We demonstrate the attack with experiments carried out on a Xilinx Spartan-7 FPGA.

Our work has some similarities to previous on-FPGA DFA experiments using power wasters [20]. This earlier work targets bytes generated before the 9th AES

round and requires an adversary to obtain faulty and fault-free ciphertexts to extract the key. DFIA, which our work employs, does not require targeting of specific bytes, nor does it require any fault-free ciphertexts.

The remainder of this chapter is organized as follows. In Section 4.1, we introduce the attacker model and the process of DFIA on AES. In Section 4.2, we characterize the attack parameters. In Section 4.3, we evaluate our DFIA method on a Xilinx Spartan-7 FPGA. Section 4.4 summarizes the work. The work described in this chapter was presented at the 2022 IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM) [54] and published in its proceedings.

4.1 Attack Model

As mentioned in the previous section, for DFIA, faults must be induced during a particular round of encryption. Ghalaty et al. [88] demonstrated DFIA using a controlled multiplexer to switch to a short-period clock for fault-inducing clock cycles. This type of clock control is challenging to implement in FPGAs since on-FPGA clocks are often derived using a main clock and phase-locked loops (PLLs). In this work, we control fault intensity by enabling differing numbers of single lookup table (LUT) ROs. In general, the number of faults per byte grows with an increasing number of enabled wasters. Using this approach, we implement the following fault attack scenario:

- The victim encrypts plaintext with a secret key that is unknown to the attacker.
- The attacker can trigger repeated encryption of the same plaintexts, but does not know the plaintexts.
- When the attacker triggers an encryption they know the start time of the encryption.
- The attacker collects the ciphertexts, which may be faulty.

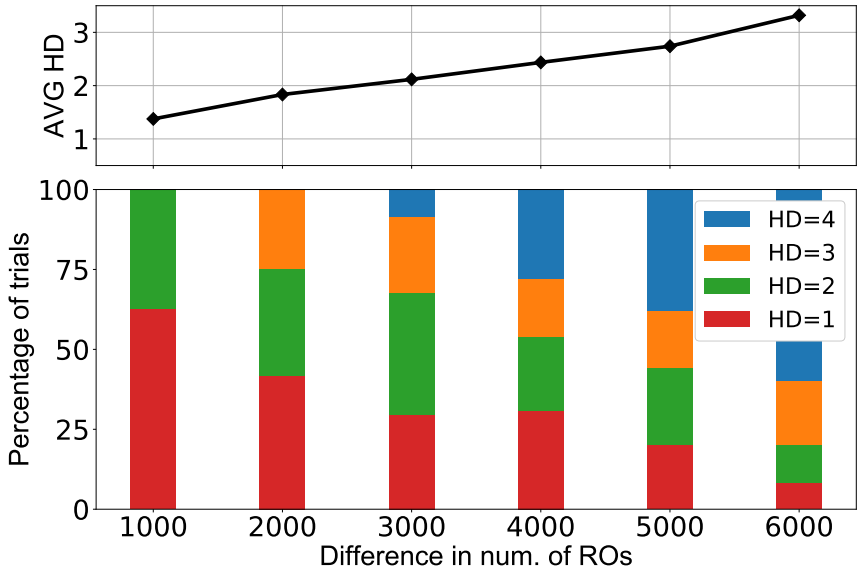


Figure 4.1: Distribution of observed Hamming distances relative to the difference in number of activated ROs. Hamming distances tend to be larger when the number of activated ROs in two trials are dissimilar. Trials in which both encryptions produced the same byte value are excluded from consideration.

For illustration, we give an example by inducing faults in one state byte during the first round of AES. In each trial, the same input value is applied twice to the first round of AES with a differing number of ROs enabled. Fig. 4.1 shows the Hamming distance (HD) between the resulting bytes, with 50 trials performed for each case. The figure shows that HD increases with incremental fault intensity; similar numbers of activated ROs will induce one or two bits of difference, and vastly different numbers of ROs induce larger HDs. The correlation between HD and fault intensity is the information that DFIA uses to identify the value of each AES key byte.

4.1.1 DFIA on AES

AES-128 is an iterative algorithm with 10 rounds; the first 9 include operations *S-Box*, *ShiftRows*, *MixColumns* and *AddRoundKey* while the final (10th) round does not have *MixColumns*. In hardware, one round is typically computed per clock cycle, with a register storing and updating intermediate state in each cycle. An attacker may

know the timing of AES rounds through on-chip voltage sensors [30] or other means. Fig. 4.2b depicts where the fault is injected into AES causing wrong bytes captured in the register. The 10th round, in the next clock cycle, still needs to operate correctly, but it operates on the faulty inputs from the register.

We illustrate the DFIA procedure generically on one of the 16 key and state bytes, as the attack works the same against each byte. Intuitively, the attack exploits the similarity (low HD) between values of S in the state register when the repeated encryptions are subjected to similar fault intensities during the 9th round. The attacker, by controlling the number of ROs, knows when state byte values should be similar, but cannot observe the values directly. Because the state byte at the end of the 9th round can be predicted by reversing from ciphertext based on a key byte hypothesis, the attacker can identify the correct hypothesis by finding the one that predicts state bytes showing the expected similarity. Varying the fault intensities causes the number of faulty bits in S to increase or decrease accordingly, but only one key guess will reveal this.

$$C = sbox(S) \oplus K \tag{4.1}$$

$$S_{i,p} = sboxInv(\mathrm{C_i} \oplus \mathrm{K_p}) \tag{4.2}$$

Formally, a ciphertext byte C can be described by Eq. 4.1 where S is a state byte after the 9th round and K is a byte of the 10th round key. We define fault intensity as an increasing order from level 0 to level n. With one plaintext, multiple possibly-faulty ciphertext bytes are generated as $C_1, C_2, \ldots, C_n$ under different fault intensity levels $(0, 1, 2, \ldots, n)$. Among the levels, we use I_D to denote the set of levels that produce a ciphertext different from the preceding level. There are 256 key byte hypotheses (K_0, K_1, ... ,K_{255}) available to the attacker. For a given plaintext, the byte $S_{i,p}$ is predicted by Eq. 4.2 using possibly-faulty ciphertext byte C_i and key byte hypothesis K_p. Only the correct key byte hypothesis predicts $S_{i,p}$ which matches the actual

byte captured in the state register at the end of 9th round; other key hypotheses yield arbitrary predictions that do not correspond to the computation performed. An exemplary case with real data is illustrated in Fig. 4.2a; the predictions and resulting HD are shown for the correct key guess, and one incorrect key guess.

For a given plaintext, the average Hamming distance between adjacent fault intensity levels according to each key hypothesis p is given by Eq. 4.3. The analysis is extended to m plaintexts in Eq. 4.4. Provided that fine control of fault intensities causes similar state bytes to be captured (see Fig. 4.1), avgHD_p will be small when K_p is the correct key byte. The diffusion of *sboxInv* in Eq. 4.2 causes mispredicted bytes to be random, so avgHD_p for all incorrect key guesses will be centered around 4, which is the expected Hamming distance between random 8-bit strings. The attacker decides the correct key to be the one that has minimum avgHD_p. In addition to AES, DFIA can also be applied to some other substitution-permutation networks like PRESENT and LED [89].

$$\text{HD}[\text{plaintext}]_p = \frac{1}{|I_D|} \sum_{i \in I_D} \text{HammingDist}(S_{i,p}, S_{i-1,p}) \tag{4.3}$$

$$\text{avgHD}_p = \frac{1}{m} \sum_{j=0}^{m} \text{HD}[\text{plaintext}_j]_p \tag{4.4}$$

4.2 Characterization of Fault Injection

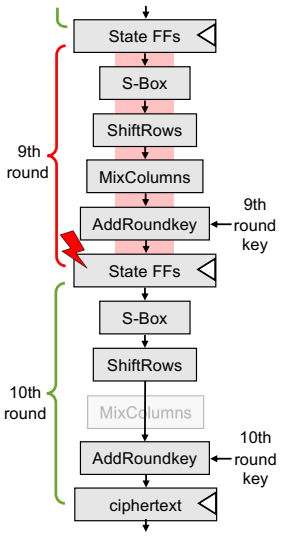
The requirements of DFIA guide our selection of fault attack parameters such as the number of ROs between levels, and when to enable and disable the wasters. We address these parameter choices in this section of the chapter.

4.2.1 Experiment setup

We perform our experiment on an Arty S7 development board featuring a Xilinx Spartan-7 XC7S50-CSGA324 FPGA with 8,159 slices and 65,200 FFs. We put 12,000

	Prediction with correct key byte		Prediction with wrong key byte	
number of ROs	9th round state byte	HD	9th round state byte	HD
100	0x01	-	0x37	-
200	0x01	0	0x37	0
⋮	0x01	0	0x37	0
6600	0x01	0	0x37	0
6700	0x09	1	0xB0	4
⋮	0x09	0	0xB0	0
8300	0x09	0	0xB0	0
8400	0x19	1	0xA1	2
⋮	0x19	0	0xA1	0

(a) Example Results



(b) Schematic

Figure 4.2: DFIA aims to inject faults in the 9th round of AES. Similar fault intensities induce low-HD values in a 9th round state byte. The correct key is identifiable because its predictions reveal low-HD values in the state byte.

ROs separated from AES by at least one row, which can be enabled in groups of 100, on the FPGA to cause excessive switching activity and high power consumption. The 12,000 ROs, AES and control logic collectively consume 72% of LUTs and 87% of slices. More ROs could be placed when DFIA is applied to larger FPGAs. The AES-128 victim design in this work is a compact round-based implementation that consumes 379 slices and 280 FFs. Round keys are pre-computed. Encrypting each plaintext takes 11 cycles; the first cycle XORs plaintext with initial key and the 10 rounds each take 1 cycle to finish.

4.2.2 Coarse Attack Timing and Number of ROs

We target the first round of AES operation as a test case to characterize attack timing. Arbitrary plaintexts are used and clock period is set as 8.5 ns. We activate ROs and then start AES after a certain delay which is swept as the parameter in this test. For each delay value, the percentage of correct bits in the first round output is

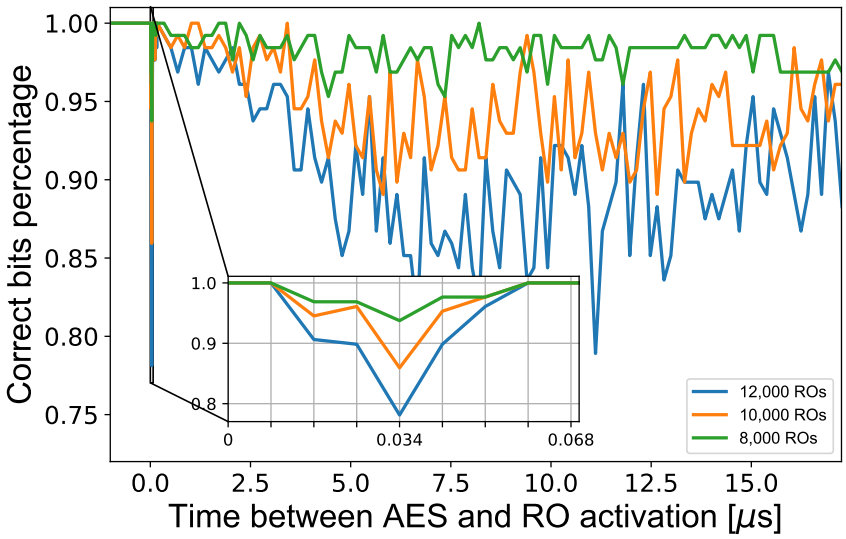


Figure 4.3: Profile of faults versus time since RO activation.

measured. Though the ROs remain on after the first AES round, this does not affect the first round output that is captured. A lower percentage of correct bits implies a larger voltage drop which is causing more path delay faults to occur. The result in Fig. 4.3 shows instant voltage drop and bounce-back after RO activation which is also observed on Intel Arria 10 [90] and Kintex-7 FPGAs [25]. The largest voltage drop happens at the 4th cycle after RO activation, corresponding to a delay of 34 ns between RO activation and the AES round. The quick reaction and recovery provides a promising scenario for injecting faults into a single AES round.

4.2.3 Quantifying Delay Change

To cause timing faults, the slowdown by ROs should be significant enough to induce path delay faults at a targeted clock frequency. Choosing appropriate attack parameters therefore requires understanding of timing margins and the amount of slowdown caused by the power wasters. From the timing report in Vivado, the critical path delay of AES module is 14.48 ns, which is known to be conservative.

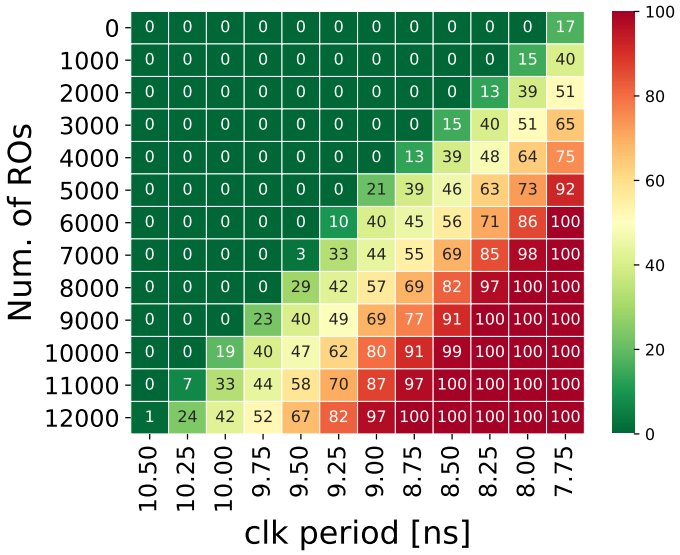
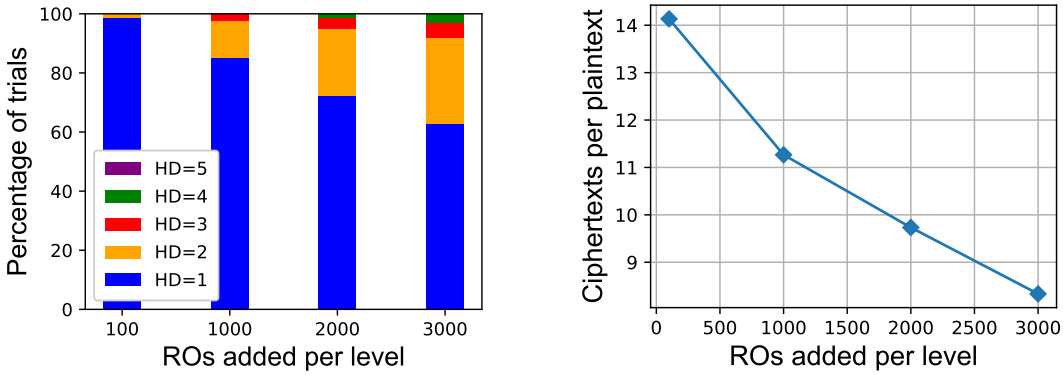


Figure 4.4: Number of faulty ciphertexts across different clock periods and number of activated ROs.

We perform experiments to estimate the timing margins and path delay profile. 100 plaintexts are encrypted across a variety of clock periods and number of activated ROs. The clock periods are swept from 7.75 ns to 10.5 ns in steps of 250 ps, and the number of ROs is swept from 0 to 12,000 in steps of 1,000 ROs. For each intensity and clock period setting, we apply the plaintexts to AES and check the number of incorrect bits produced in the first round output. The result in Fig. 4.4 shows that faults start emerging when the clock period is 7.75 ns without ROs, which gives an indication of the true delay at nominal conditions of the longest path sensitized during the 100 encryption rounds. To equate ROs to a delay change, we look for multiple pairings of clock and ROs that have a similar number of faults. For example, in Fig. 4.4, a clock period of 10.25 ns with 12,000 ROs, and a clock period of 7.75 ns without ROs cause a similar number of timing faults. From this we infer that the impact of 12,000 ROs is roughly equivalent to the 2.5 ns difference in clock period, and hence that 12,000 ROs cause around a 32.2% slowdown.

4.2.4 Hamming Distance with Different Fault Intensities

Although DFIA does not require fault-free ciphertexts, we assume that the design is overclocked but operating correctly at nominal conditions. The 8.75 ns clock period used in this section allows the AES design to be fault-free when the ROs are off, and susceptible to producing a variety of faulty values when ROs are activated. Fault intensities can be ordered as incremental levels by adding activated ROs between each level. A larger difference of ROs between levels bring about larger changes in the voltage drop, which increases the HDs between neighboring levels. To investigate the resolution of HD, 60 plaintexts are repeatedly applied to AES while varying the number of ROs between adjacent levels. The result in Fig. 4.5a shows that the HD distribution tends to be larger when more ROs are activated between levels. Because the ideal outcome is to have 1 as the HD value in order to make the correct key stand out, the larger HDs may diminish the attacker's resolution in distinguishing the correct key. Fig. 4.5b shows how many different ciphertexts arise when sweeping the fault intensity for each plaintext; the percentages in Fig. 4.5a consider only these cases where the ciphertext changes across levels.



(a) HD distribution

(b) Number of ciphertexts

Figure 4.5: Increasing the step size of the fault intensity produces fewer cases with low Hamming distance, and a smaller number of usable ciphertexts per plaintext.

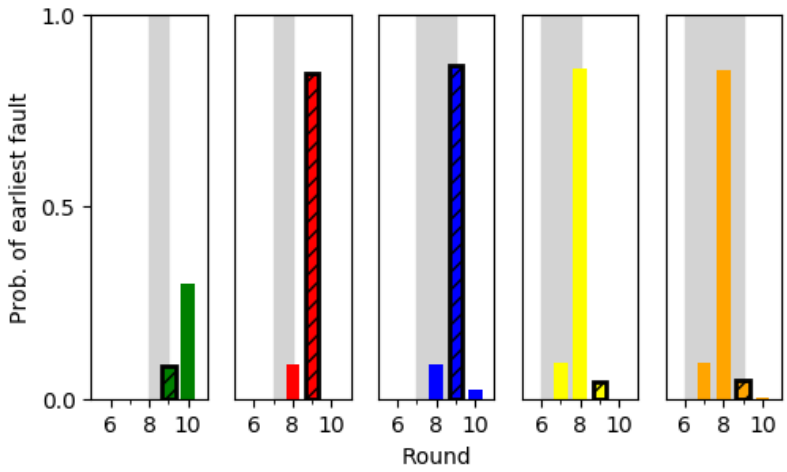


Figure 4.6: Five attack scenarios identified for attacking the 9th round in AES. The gray box indicates the cycles in which the ROs are active for each scenario. Activating the ROs in the 7th round for 1 or 2 cycles causes faults in the 8th round in approximately 9% of encrypted plaintexts, and in the 9th round for 86.6% and 84.6% respectively.

4.2.5 Attack Specificity

Section 4.2.2 showed the existence of a fault spike shortly after RO activation which enables the possibility to attack the 9th round in AES. In this subsection, we determine suitable parameters for the timing and duration of the RO activation, keeping the same 8.75 ns clock period from the prior subsection. To find the optimal parameters, we sweep the timing and duration to activate 12,000 ROs from 1st to 9th round. 1000 random plaintexts are applied to AES and the output of each round is checked to find the earliest round with faults in each trial. Only five suitable configurations that successfully attack the 9th round are identified, and these are summarized in Fig. 4.6. Activating the ROs for 1 or 2 cycles starting from the 7th round of encryption are both successful in causing 9th round faults, as depicted in the 2nd and 3rd subplots of Fig. 4.6. Between these choices, we select the former because it does not induce any faults in the 10th round which would render the 9th round fault useless toward distinguishing the correct key.

4.3 DFIA Evaluation

In this section, we present the result of attacking AES and discuss some trade-offs with the control of fault intensities. Based on the results in the prior section, for DFIA, ROs are activated in the 7th round of encryption for just one cycle and the number of ROs is swept to control intensity. The victim AES is running at 8.75 ns clock period as in the previous section.

4.3.1 Key Extraction from AES

We firstly demonstrate the evaluation of extracting key bytes from AES by DFIA. The 12,000 ROs are configured as 121 fault intensity levels, with 100 ROs added per level. For each randomly generated plaintext, we iterate all RO levels with the selected injection timing and perform the DFIA analysis on the resulting ciphertexts. The result of a successful attack toward four arbitrary key bytes is shown in Fig. 4.7. In order to quantify how many ciphertexts are needed before the correct key stands out, we use Measurements-to-Disclosure (MTD) as the metric. Specifically, we consider a key byte as the correct one when its average HD is lowest among all guesses and remains so for at least 20 consecutive ciphertexts. The threshold for MTD ensures that a key is only declared as correct once it consistently outperforms other guesses, and this increases robustness against randomness in the first few trials. Out of the 16 key bytes in AES, 12 are successfully extracted; the remaining 4 key bytes have shorter paths and produce an insufficient number of faulty ciphertext bytes at this particular clock frequency.

4.3.2 Performance

Section 4.2.4 has shown that the HD tends to be larger when subjected to more ROs added per fault level. The larger HDs might require more plaintexts to extract the keys, because less useful information is derived from each plaintext. On the other hand, when using a finer number of ROs per level, the ciphertexts remain the same

(HD=0) across most fault intensity levels, and these trials do not contribute any distinguishing information about the key and represent wasted work by the attacker. We evaluate the tradeoff between these opposing objectives of minimizing plaintexts and minimizing trials performing the DFIA attack with 12,000 ROs while sweeping the number of ROs per level. For each setting, we make use of all intensity levels and measure the number of plaintexts and trials that are required to extract the key bytes. The result in Fig. 4.8a shows that as the number of ROs per level increases, the number of required plaintexts also increases. This is attributable to two factors: (1) larger HD between levels makes the correct key less distinguishable from other key guesses, and (2) fewer ciphertexts for each plaintext are generated. Therefore, if the goal is to minimize the number of plaintexts, it is desirable to use the finest possible fault intensity levels in order to maximize the information extracted from each. On the other hand, Fig. 4.8b shows that the number of trials to extract a key decreases as number of ROs per fault level increases, because there are fewer fault levels that generate identical and useless ciphertexts. Note that the observations about selecting the number of ROs per level to minimize plaintexts or trials hold universally across all 12 bytes that are broken in the attack.

4.4 Chapter Summary

The work in this chapter presents the first DFIA on AES that is suitable for remote FPGAs. The use of ROs to create precisely controllable voltage glitches provides an attacker with fine control of fault injection, allowing for a diverse set of faulty ciphertexts to be generated. We show experimentally that AES key bytes are successfully extracted in the attack, and furthermore study how to tailor fault injection parameters for the attack, and for various competing objectives that the attacker might have.

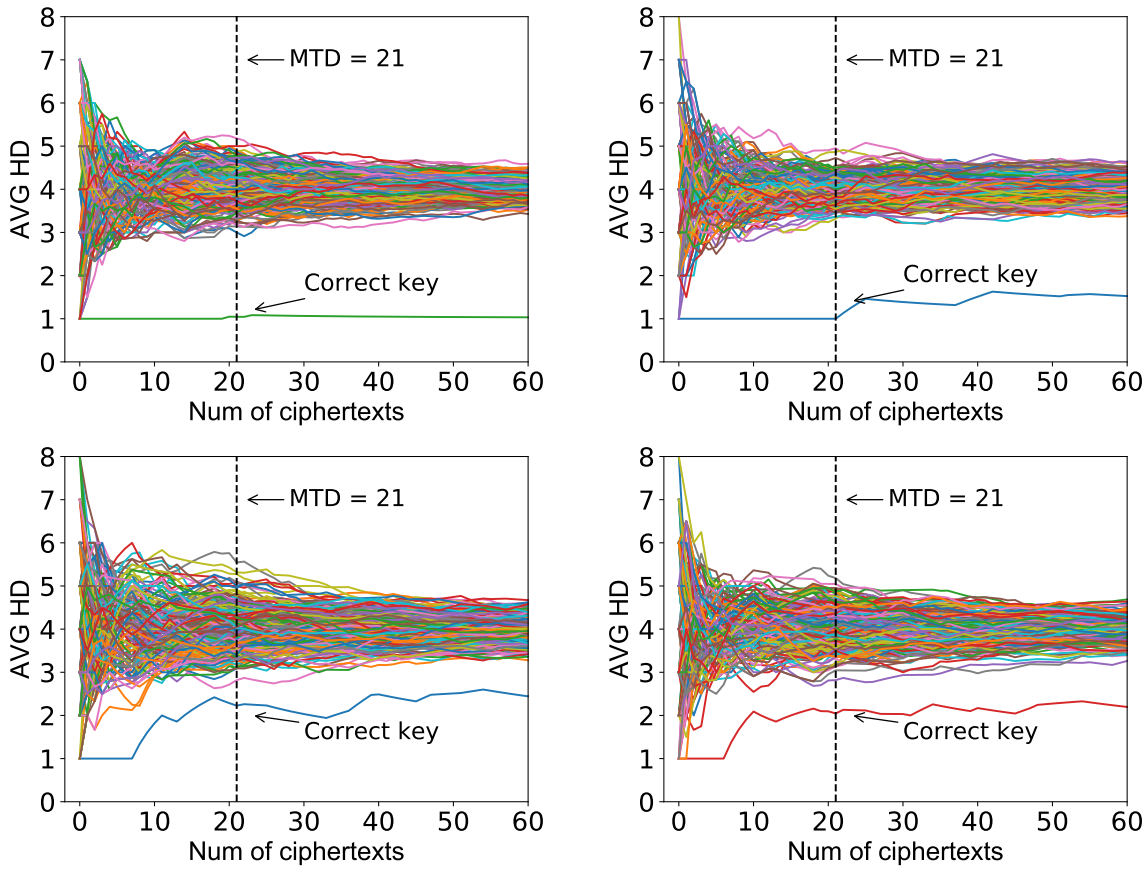
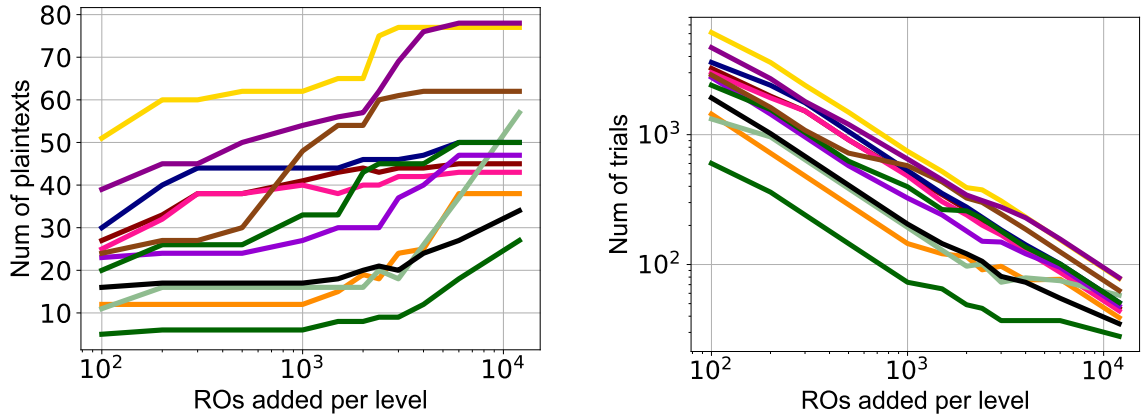


Figure 4.7: Average HD traces under 256 key bytes versus number of ciphertexts. The MTD is 21 in each case, and the extracted key byte is confirmed to be the correct one.



(a) plaintexts (b) trials

Figure 4.8: The traces represent 12 extracted bytes. The number of plaintexts and trials required to extract key bytes vary with the step size of injected fault intensity.

CHAPTER 5

DIFFERENTIAL ANALYSIS FOR ON-CHIP SENSING AND LOCATING

In this chapter, we present a differential sensing method for detecting the location of targets on multi-tenant FPGAs. TDCs are chosen as voltage sensors and their structure is shown in Fig. 3.2a. We apply differential analysis to the Hamming Weight (HW) from TDCs and use contours to characterize how the locations and power consumption of target circuits affects the differential HW; the contours we create enable predicting the location of an unknown target circuit. Then we evaluate the accuracy of prediction on an AWS EC2 F1 instance and the evaluate consistency of the method across instances. As we conclude this chapter, we outline and discuss further research directions for this works.

5.1 Motivation

In Chapter 2, we present a comprehensive review of prior research on remote attacks targeting FPGAs, including side channel attacks and voltage-based fault attacks. The technique in this chapter can be used in support of side channel attacks, and its mechanism is similar to a countermeasure used against fault attacks, so we consider both topics here.

5.1.1 Support for Side Channel Attacks

From the attacker's perspective, to implement remote power analysis and long wire coupling side channel attacks, it can be important to find the location of the victim. Unlike many preceding studies on side channel attacks in multi-tenant FPGAs,

which assume the victim location is known, we assume the victim location is unknown and our primary objective is to find the location of the victim based on its power consumption. For remote power analysis, knowing the victim location allows sensors being placed nearby where they are more sensitive to power consumption. Similarly, for long wire coupling side channel attack, knowing the victim position helps the attacker to place receivers near transmitters as required.

5.1.2 Relationship to Fault Attack Countermeasure

To detect fault attacks, both [44, 46] use ROs to monitor on-chip voltage and identify the location of the malicious circuit following a voltage attack. Generally, the power consumption of the malicious circuit has to be substantial in order to drop the voltage enough to induce faults in a victim circuit that is operating at a legal clock frequency. In [42], the power consumption of different implementations of ROs on Xilinx FPGAs is reported roughly 1 mW - 2 mW. In Chapter 4, we deploy 8,000 - 12,000 ROs to inject faults, resulting in an estimated power consumption of 8 W - 24 W. Circuits consuming such an intentional and substantial amount of power are easier to detect than more ordinary circuits. The difference between our work and previous works [44, 46] is listed below.

- Our method in this work can detect targets with power consumption 0.2 W or less. This amount of power consumption is typical of a circuit that might be a target in side channel attacks. Meanwhile, power wasters to inject faults typically consume much more power. For instance, the ones in Chapter 4 consume 8 W - 24 W.

- It is practical for multiple tenants to run their applications concurrently on the multi-tenant FPGAs. Their switching activities may disrupt the detection of circuits. However, most prior works including [44, 46] only consider the switching

of the victim and attacker on multi-tenant FPGAs. Our method considers the presence of other tenants that switch randomly on the same FPGA.

- Unlike ROs used in [44, 46], which are prohibited on AWS cloud FPGAs, TDCs are deployed in this work and compatible with AWS cloud FPGAs.

The subsequent sections of this chapter are organized as follows. In Section 5.2, we introduce the differential analysis of sensor values and illustrate its application in locating a target circuit. In Section 5.3, we evaluate our method on cloud FPGAs with TDCs and target circuits. Section 5.4, provides future directions for extending and improving this work, and Section 5.5 summarizes the chapter.

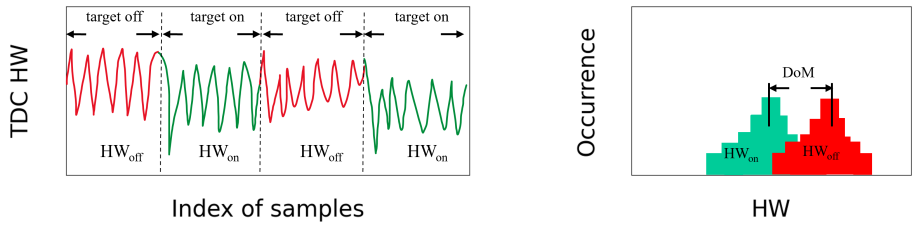
5.2 Methodology

In this section, we articulate differential analysis of data in TDCs and how we apply the analysis to generate contours by characterizing the impact of a target circuit on the HW of TDCs. Then we demonstrate how we apply the contours to predict the location of the target circuit.

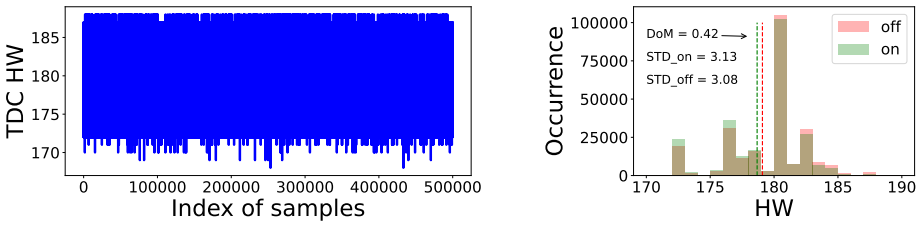
5.2.1 Differential Analysis of Voltage Sensors

The main idea of differential sensing is that the area of the FPGA that contains the target circuit should experience a small voltage drop when the target circuit is active on account of the higher power consumption during that time. The voltage drop slows the propagation of the signal propagating through the TDC circuit, thereby reducing the HW of the samples. By leveraging enough data gathered from TDC-based voltage sensors, it is possible to discern the impact of voltage drop and to eventually learn about position the target circuit and the amount of power it consumes.

We assume that the location of the target is unknown to the attacker. However, we assume that that the attacker knows when the target circuit is active and when it is idle. Additionally, we allow for the presence of other tenants on the same FPGA,



(a) Explanatory illustration.



(b) Analogous plots generated from experimental data.

Figure 5.1: HW distributions when the target switches on and off over time. HW tends to be larger when the target is off. Note that, when DoMs are generated from experimental data, the difference between the HW values is much smaller than the standard deviation of the samples.

assuming their switching activity occurs randomly and it is uncorrelated to when the target is active.

Depending on whether our target is active or idle, we categorize HW values from a TDC into one of two classes. HW values obtained when the target is idle are denoted as HW_{off}, while the HW values obtained when the target is active are denoted as HW_{on}. Illustrative distributions of HW_{off} and HW_{on} are shown in Fig. 5.1a and an example of distributions generated by experimental data is shown in Fig. 5.1b. Here we define the difference of means (DoMs) between the two distributions as $DoM = mean(HW_{on}) - mean(HW_{off})$. Since the active times of all other tenants appear equally in both distributions, the only difference between the two distributions is the power consumption caused by the switching activity of the target circuit while active. For this reason, DoM is a measure of how much the TDC is affected by the target circuit. Fig. 5.2 provides an example of raw HW and DoM derived from HW.

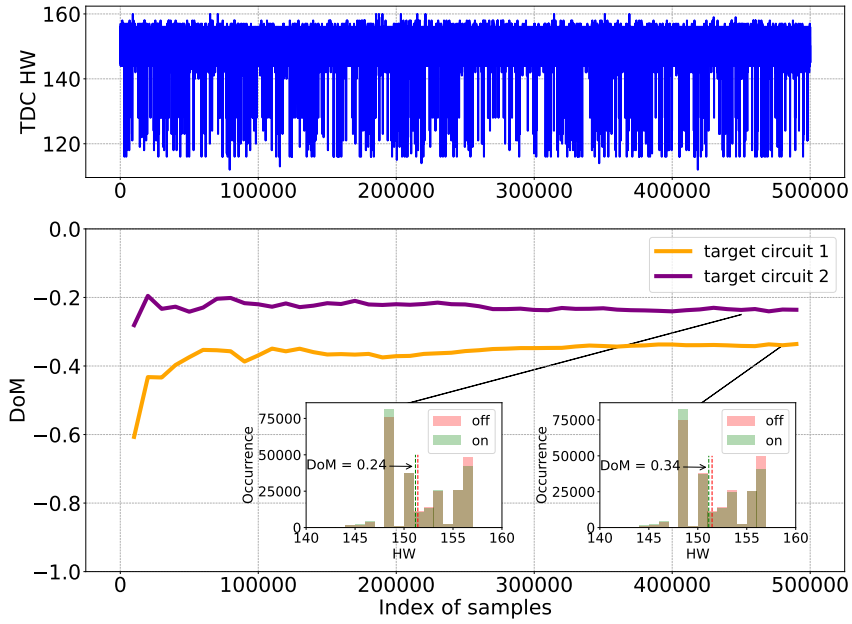


Figure 5.2: An example of raw HW in TDC and DoM derived from raw HW.

Before finding the location of the target, DoM can help us to identify whether the TDC and target circuit are collocated on the same FPGAs. The non-zero magnitude DoM caused by the switching activity of the target circuit only exists when the target shares the FPGA fabric with the sensors. In cases where the target is located elsewhere, HW_{off} and HW_{on} are indistinguishable and DoM tends to converge toward zero as shown in Fig. 5.3d. Therefore, by choosing an appropriate significance threshold, and collecting enough data, it is possible to use DoM to determine collocation. Once the collocation of the target circuit is established, through deployment of a set of TDCs, we use DoMs to locate the target as detailed in the next sections. First, we start by establishing general characteristics relating to DoMs.

- **DoM characterization requires many samples:** The relationship between number of samples and DoM for each TDC is graphically represented in Fig. 5.3a. As can be observed in that figure, a large number of samples must be collected

in order to get a precise estimation of DoM. The reason is that DoM is a small signal which is being extracted from noisy TDC data; the DoM of all four TDCs in this case have magnitude less than 0.60, whereas the HW values from which they are extracted have standard deviations of 3.4, 3.6, 3.1, and 2.1, respectively.

- **DoM depends on power:** When the power consumption of the target increases, the active state of the target causes a larger voltage drop, and has more impact on TDCs. Hence, the magnitude of DoM increases with power consumption of the target, as observed in Fig. 5.3b.

- **DoM depends on distance:** Because voltage drops are localized, proximity between sensor and target results in a more significant voltage drop at the sensor and thus a larger magnitude of DoM as shown in Fig. 5.3c.

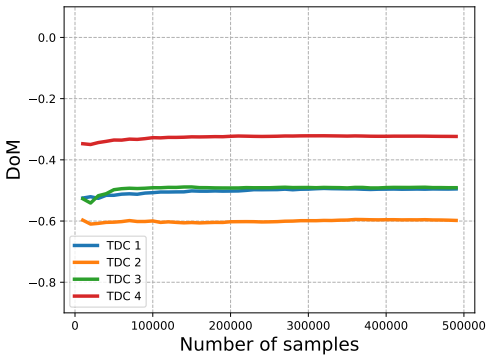
5.2.2 DoM Contours

The impact on a TDC of target circuits at every possible location can be viewed as a DoM contour. Each TDC possesses its own unique DoM contour, which is like a map that shows its sensitivity to targets. After explaining how contours are generated, Section 5.2.4 explains how they can be used with the computed DoM values of a target circuit in order to predict its location.

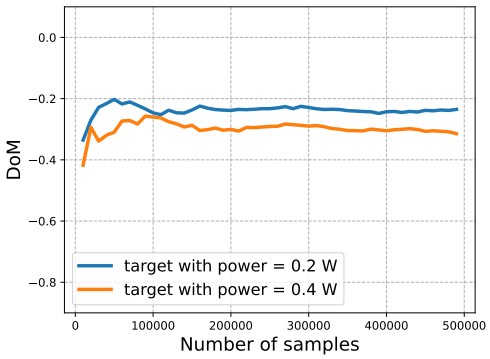
5.2.3 Contour Generation

To generate DoM contours, we place four TDC instances and 20 instances of a calibration circuit on the same FPGA. The calibration circuits, further described in Section 5.3.1, are configured to be randomly switching on and off, and a log is kept of which circuits are active at the time of each TDC sample.

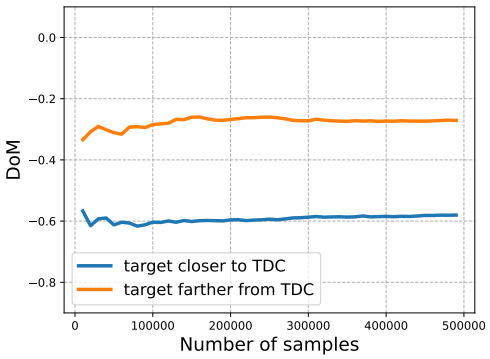
For each calibration circuit, contingent on its power state (on or off), we categorize the HW samples of each TDC into two classes and compute the DoM of each TDC. Once all DoMs are calculated, we generate DoM contours for each TDC utilizing radial



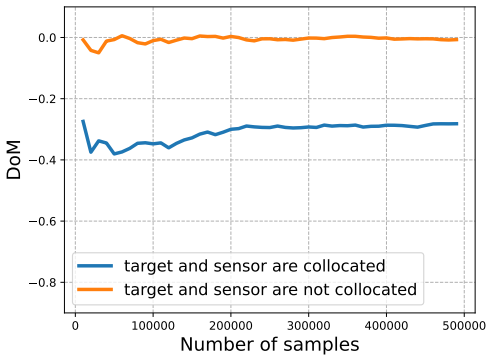
(a) DoM vs. number of HW samples



(b) DoM varies with power consumption

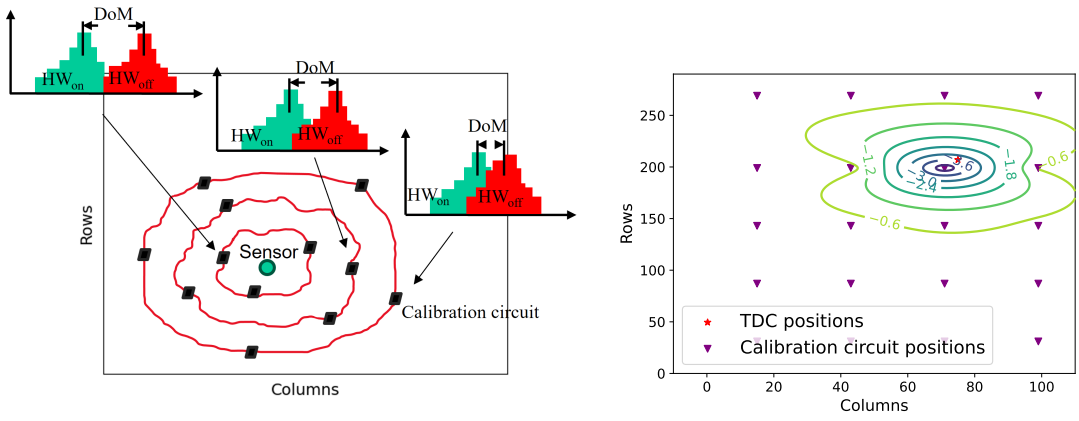


(c) DoM varies with placement



(d) DoM detects collocation of targets

Figure 5.3: Comparison and evaluation of DoMs in different scenarios.



(a) Interpolation from calibration circuits.

(b) An example of a DoM contour.

Figure 5.4: An example of interpolation for contour generation and a DoM contour.

basis function interpolation to interpolate the DoMs of the calibration circuits to all other coordinates. Fig. 5.4 illustrates an example of interpolation (Fig. 5.4a) and a contour generated from experimental data (Fig. 5.4b). The resultant DoM contours for the four TDCs are illustrated in Fig. 5.5. All four contours show larger magnitude of DoM for positions in close proximity to the corresponding TDC, indicating that adjacent circuits can exert more substantial impact on TDCs. The contour generation is a one-time calibration and will be the foundation to subsequently predict the target circuit. Though our method is presented and validated with the same type of IPs for both calibration circuits and the targets being predicted, our method should also be applied to the scenario when they are not the same type.

5.2.4 Predicting Target Positions Using DoM Contours

The interpolated contour assigns a value to each coordinate on the floorplan, representing the expected DoM when a target circuit is placed at that specific coordinate. Given DoMs of a target circuit with unknown location on the FPGA, we can predict the target location by using contours to find the coordinate from which the DoMs are expected to be most similar to the given DoMs. To formalize this prediction process,

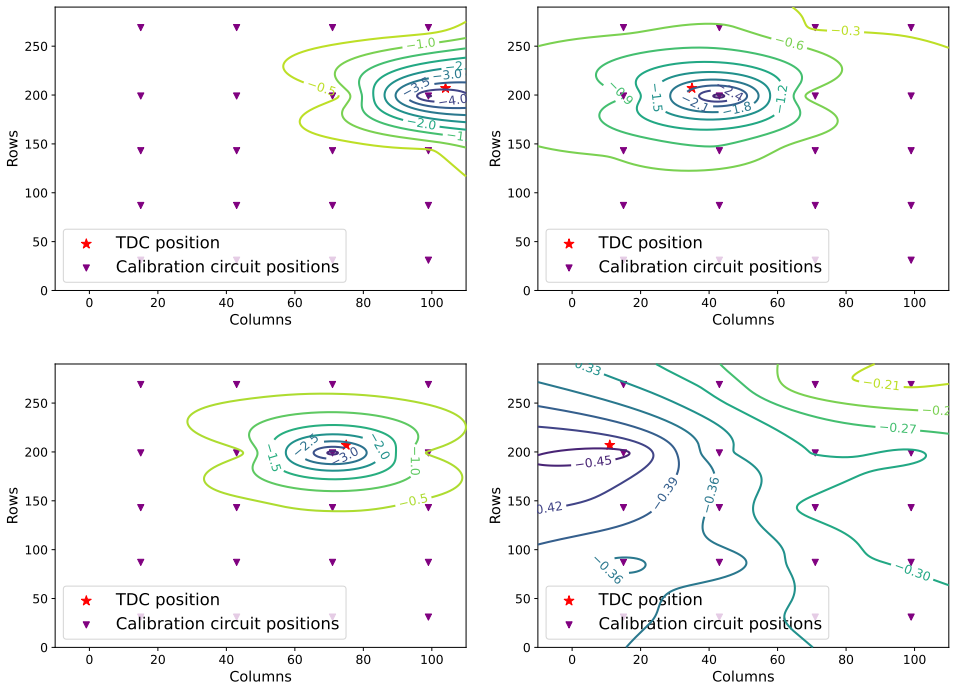


Figure 5.5: DoM contours of four TDCs. Each contour is constructed using DoMs from 20 calibration circuits.

here we aggregate contours from each TDC to construct an error function $f(x, y)$ in Eq. 5.1, which is the summation of squared errors between the predicted DoMs from coordinate (x, y) and the actual DoMs from a target of unknown position. Our approach aims to find the predicted location in terms of (x, y) that minimizes $f(x, y)$. As Fig. 5.6 shows, the DoMs of the target circuit are similar to the contour values at the locations that correspond to accurate predictions of target location, which causes low values of $f(x, y)$; inaccurate prediction leads to larger values of $f(x, y)$, indicating larger difference between DoM and contour values at those locations.

For our accuracy metric, we define the distance error between the target location and its predicted location, based on contours, as follows: if the prediction falls within the Pblock of the IP, the error is zero. Otherwise the error is the Euclidean distance, in terms of FPGA slices, between the prediction and nearest point inside the Pblock, as shown in Eq. 5.2. Here, x and y represent the slice indexes of the column and row

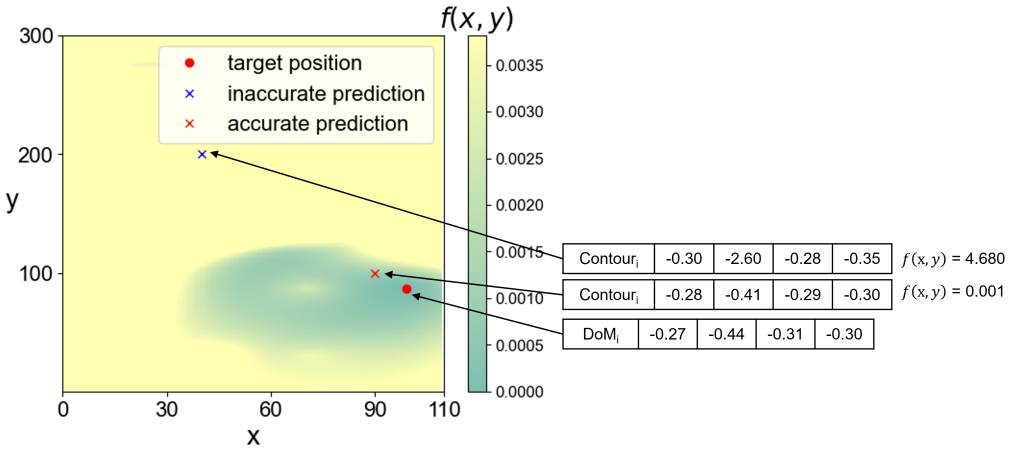


Figure 5.6: An example of prediction with $f(x, y)$.

of the predicted position, while x' and y' are the slice indexes of the column and row of the nearest point within the Pblock.

$$f(x,y) = \sum_{i=1}^{4} (contour_i(x,y) - DoM_i)^2 \tag{5.1}$$

$$Distance = \sqrt{(x-x')^2 + (y-y')^2} \tag{5.2}$$

5.3 Evaluation and Discussion

In this section, we evaluate our approach by assessing the accuracy of prediction and the uniqueness of DoMs. The assessment is performed in the cloud, using AWS EC2 F1 FPGA instances, the details of which are introduced in Section 3.2.

5.3.1 Experiment Setup

In our experiments, we position four sensors along with 20 calibration circuits on the same super logic region (SLR), with the floorplan illustrated in Fig. 5.7. Each calibration circuit comprises four instances of AES-128 single rounds and consumes approximately 0.2 W. These 20 calibration circuits are uniformly distributed, each

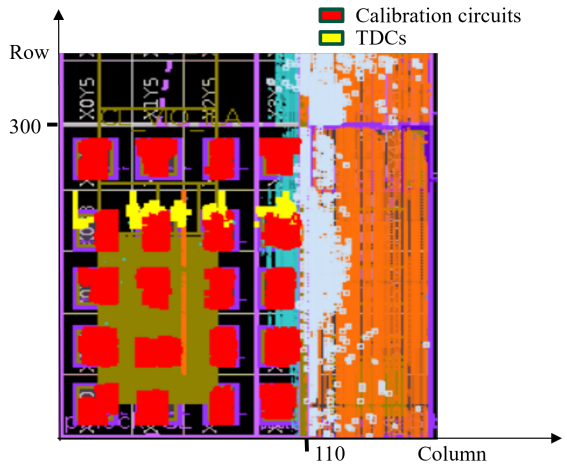


Figure 5.7: Floorplan of the four TDCs and 20 calibration circuits on the same SLR.

assigned to a Pblock covering 18 columns and 36 rows. The clock frequency for both TDCs and the calibration circuits is 125 MHz. Each sensor mainly comprises 32 8-bit carry stages and tunable delay elements as described in Section 3.2.1 and Section 3.2.2. The four sensors share a common control unit that can configure the delay elements in each sensor independently and generate a rising edge to each TDC sensor. The tunable delay in each sensor is configured to ensure the rising edge can be captured by TDCs when the clock edge arrives. The TDC sampling rate is 21 kHz. The calibration circuits have a 20% probability of changing their power states after each sample and the power states are randomly assigned.

5.3.2 Prediction Accuracy

To assess the accuracy of predictions in the presence of other tenants, we exhaust 20 combinations by picking 19 calibration circuits to generate contours and the remaining one as the target to predict. With the prediction error defined using the distance metric from Section 5.2.4, the mean distance is measured as 40.

To explore how accuracy varies when the number of calibration circuits changes, we systematically vary the number of calibration circuits employed in contour generation.

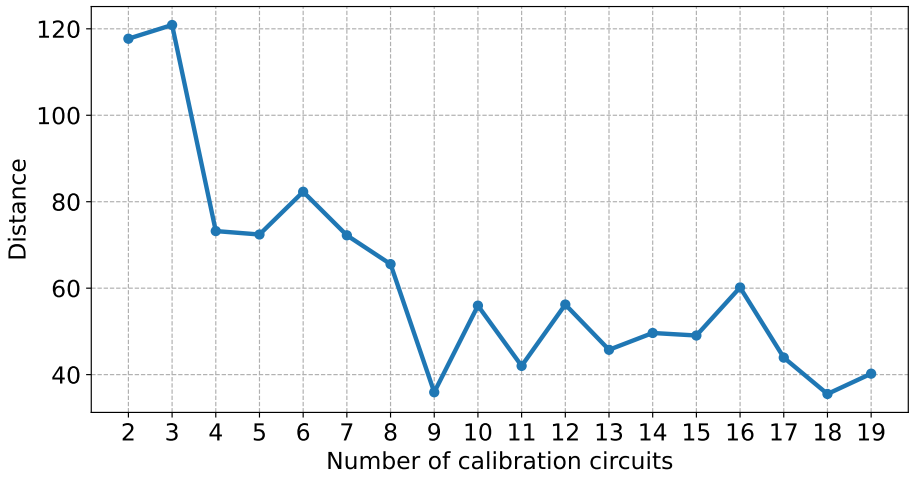
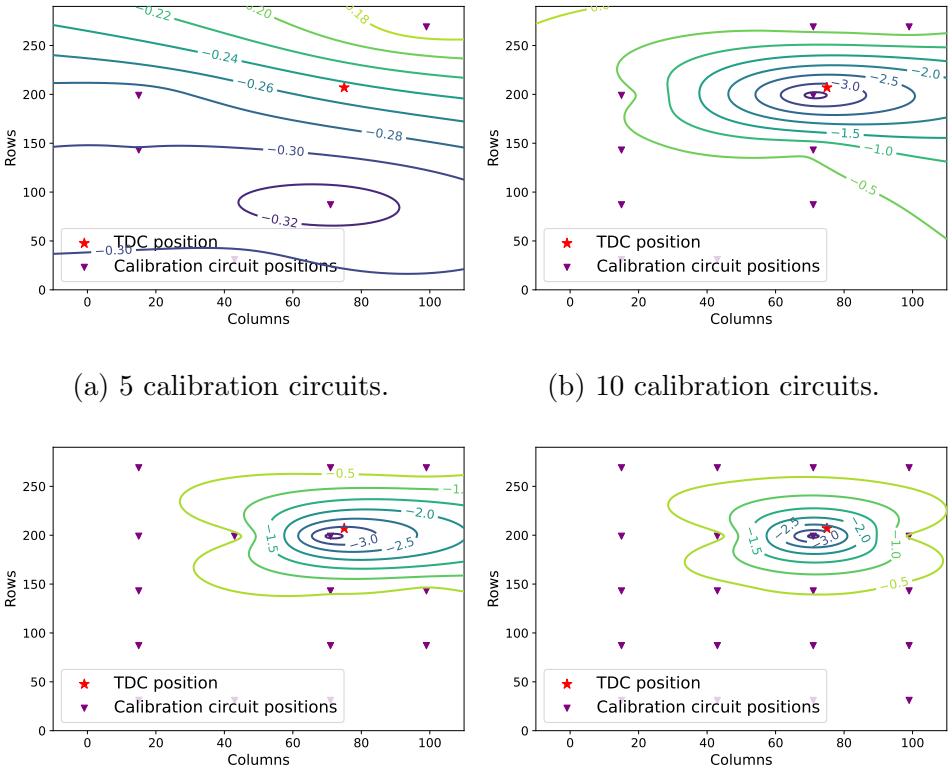


Figure 5.8: Prediction accuracy improves as more calibration circuits are used to generate the DoM contours.

For each number of calibration circuits, we randomly select 20 non-repeating subsets for calibration and one additional from the remaining pool as the target to predict its location. Here calibration circuits not only contribute to generate contours but the calibration circuits that are not being used play the role of other tenants on the same FPGA that are randomly switching on and off. The mean prediction error generally improves with more calibration circuits, but then saturates at around 10 calibration circuits, as shown in Fig. 5.8. One cause is that, when contours are constructed with fewer calibration circuits, the accuracy of the interpolated functions to describe DoMs is compromised (see Fig. 5.9); the basic contour shape seems to be well established when using around 10 or more calibration circuits, which may explain why the prediction accuracy in Fig. 5.8 saturates beyond this number of calibration points. Despite achieving steady distance within the range of 40 to 60, further enhancements and analysis are proposed in Section 5.4.

5.3.3 Uniqueness Evaluation

The HW values produced from a TDC can fluctuate due to process, voltage, and temperature (PVT) variations. Therefore DoM, which is a difference in HW



(a) 5 calibration circuits.

(b) 10 calibration circuits.

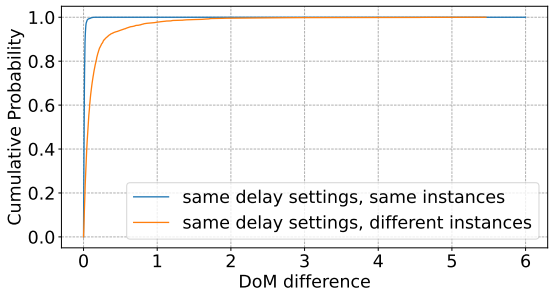
(c) 15 calibration circuits.

(d) 20 calibration circuits.

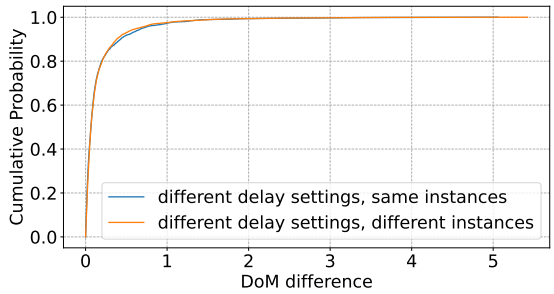
Figure 5.9: Contours of a TDC generated from varied numbers of calibration circuits.

values, can also be impacted by PVT. DoM can additionally be impacted by the delay setting of the TDC (see Fig. 3.2), which changes the flip-flop positions in the TDC chain where the rising edge is captured; the position of the rising edge captured within the TDC is a significant consideration because propagation delay and clock skew together lead to uneven delays between flip-flops in the carry chains [52, 53]. Since both PVT and delay tuning can impact DoMs, it is important to understand which has more impact on DoM variations. If process variation of the PDN between the calibration circuit and TDC is a dominant component of DoM variation, then calibration may need to performed on each FPGA instance, but there would be a possibility of using DoM contours to fingerprint instances, as a new type of FPGA PUF [91, 92, 93, 94, 95, 96, 97, 98, 99]. On the other hand, if simply changing delay settings creates large DoM variation, it would indicate that DoM based on TDC HW will not be invariant with respect to factors that change TDC HW.

To analyze difference between DoMs, we collect data from 12 AWS FPGA instances. Four sensors and 20 calibration circuits are used on each instance following the setup in Section 5.3.2; five delay settings are used for each TDC. Then, we make pairwise DoM comparisons to create the CDFs in Fig. 5.10. Fig. 5.10a shows that, when delay setting is held constant, two DoMs from different TDC instances are less similar than two DoMs from the same TDC instance, as would be expected. However, roughly the same amount of DoM difference can be obtained just by comparing DoMs from a single instance across two delay settings, as Fig. 5.10b shows. Therefore, the observed variation in DoMs is apparently an artifact of which flip-flops in the TDC capture the rising edge, rather than the impact of uniqueness in the voltage drop caused by the power consumption of the target. Section 5.4.1 in future work proposes a new type of analysis of DoMs, which is expected to provide a more accurate measurement of DoMs when HW changes.



(a) Comparison of same delay settings.



(b) Comparison of different delay settings.

Figure 5.10: Pairwise comparison of DoMs across TDC delay settings and instances.

5.4 Future Work

The TDC-based target prediction results are not yet working up to their potential at the time when this thesis is submitted. Based on a developed understanding of TDC limitations, and seeing successful target prediction experiments performed by labmates using RO sensors, in this section, we explain three ways that the TDC target prediction methodology could be improved in order to yield better results.

5.4.1 Computing DoM in Terms of Delay

Our work uses DoM as a measure of how much the TDC HW changes; HW itself is a proxy for delay change caused by voltage drop. Unfortunately, HW is an imperfect proxy for voltage. As shown in the previous result in Fig. 3.6c, because of differences in propagation delay and clock skew, the amount of delay needed for each HW increment is not uniform; Fig. 5.11 shows the distribution of the amount of delay required for each increment in HW, using data from Fig. 3.6c. Among 255 HW increments, 149 cases out of 255 require a delay change of just $0.25 \times \sigma_{jitter}$ or less, while 1 case requires a delay change of more than $2.25 \times \sigma_{jitter}$. This means, for instance, that the delay change needed to increase HW from 95 to 96 might be seven times larger than the delay change that causes the equivalent HW increase from 96 to 97, which illustrates the limitation of using HW as the basis for sensing changes. When we use HW to

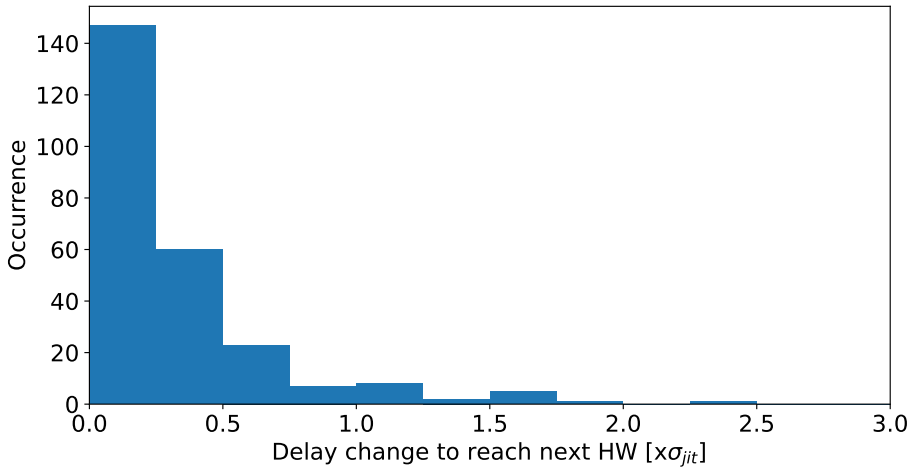


Figure 5.11: Distribution of the amount of delay change required to cause each increment in TDC HW.

further calculate DoMs and contours, the accuracy of DoMs and contours are therefore compromised by the imprecision of HW-based DoM.

We can improve the accuracy and impartially evaluate the uniqueness by following our methods in Section 3.3.2 to map HW changes to delay changes in units of σ_{jitter}, which is the standard deviation of clock jitter. Instead of giving a DoM in terms of HW, the DoM would then be calculated in terms of delay. For example a DoM of $2 \times \sigma_{jitter}$ would indicate that the the rising edge in the TDC has slowed down by exactly this amount, regardless of how it manifests in HW. DoMs recalculated in units of σ_{jitter} should yield more accurate contours and therefore better predictions of target location.

5.4.2 Scaling DoM from Calibration

Additionally, in our TDC experiments, calibration circuits and the target being predicted are both the same type of IP, so there is no need to account for cases where the calibration circuits use a different amount of power from the target circuit. The contours must be scalable if the target IP being predicted is different from the IPs used in calibration. Though ROs are not allowed on AWS cloud FPGAs, on a local

Xilinx Virtex UltraScale+ FPGA, my labmate Aleksa Deric has generated a larger dataset with 16 RO sensors and 57 calibration circuit instances, and I have contributed to processing and analyzing his data.

His work generates contours using RO calibration circuits and then uses the contours to predict the location of four different target circuits, all of which have different power consumption (ranging from 0.4 W to 1 W per instance) from the calibration RO circuit (power of 50 mW per instance). Fig. 5.12 shows his result across various numbers of sensors. When a sufficient number of sensors are employed, high accuracy of prediction is achieved across all four targets. The key insight in his approach is that, instead of searching for (x, y) position that minimizes the error, he searches for x, y, and z that minimize the error shown in Eq. 5.3; x, y are defined as in Eq. 5.1, z scales the DoM contours linearly up or down.

$$f(x,y,z) = \sum_{i=1}^{num\ sensors} (z \times contour_i(x,y) - DoM_i)^2 \tag{5.3}$$

5.4.3 The Need for More Sensors

Furthermore, we can evaluate how the number of sensors influences the accuracy. As shown in Fig. 5.5, each contour exhibits significant changes in DoM values, particularly in the vicinity of the sensor. This suggests that sensors demonstrate higher sensitivity to their nearby areas. When more sensors are put into use, it is possible to have sensors with high sensitivity to the target circuit and facilitate the prediction. Additionally, having more sensors helps to make prediction more robust to unexpected measurements from any single sensor. Prediction accuracy with more sensors is demonstrated in Fig. 5.12, which as in the prior result, uses a dataset featuring 16 RO sensors. The accuracy improves as the number of RO sensors increases. Similar to ROs, increasing the number of TDCs can also possibly enhance the accuracy.

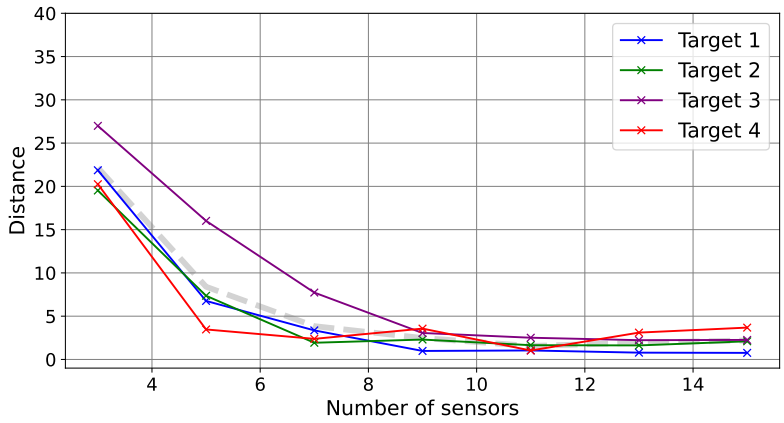


Figure 5.12: Accuracy of prediction vs. number of RO sensors.

5.5 Chapter Summary

In this chapter, we address the important problem of locating a target circuit by sensors on multi-tenant FPGAs, when the target consumes much less power than the power wasters that are typically used for fault attacks. By differential analysis of HW in TDCs, we use contours to characterize how the locations and power consumption of the target circuit impacts HW in TDCs and further predict the target location based on the contours. The accuracy of prediction and uniqueness of the impact by target circuits on HW in TDCs are evaluated in this work as well. Although the quality of the target prediction results is currently low, extensions are proposed that can potentially improve the accuracy.

CHAPTER 6

CONCLUSION

In this dissertation, we focus on security threats and defenses for multi-tenant FPGAs, which is a promising future direction of cloud FPGAs. We design a TRNG on the AWS cloud FPGAs, investigate a type of fault attack and a method to locate target circuits on multi-tenant FPGAs.

In Chapter 3, we design a TRNG on AWS EC2 F1 cloud FPGAs that overcomes the restriction on combinational loops by cloud FPGA vendors. We build a stochastic model, by which min-entropy of TRNG is tested on 60 instances. Also, the stochastic model is validated by NIST entropy assessment suite and the samples from TRNG are evaluated by NIST statistical test. In addition to this, we also extend our work to a linked sampling TRNG by changing the delay of clock paths, which increases throughput and min-entropy per sample. This work can be deployed to secure cryptographic applications on cloud FPGAs.

In Chapter 4, we present DFIA, which is a fault attack, to extract keys from AES on multi-tenant FPGAs. We place up to 12,000 ROs on a Xilinx Spartan-7 FPGA to inject fault into AES. As required by DFIA, we carefully regulate the fault intensity by controlling the number of activated ROs and precisely select timing to inject fault. By analyzing the faulty ciphertexts of AES under different fault intensities, we successfully extract 12 out of 16 key bytes. This work shows a security threat on multi-tenant FPGAs.

In Chapter 5, we use TDCs as voltage sensors to locate target circuits on multi-tenant FPGAs. We apply differential analysis to HW in TDCs and use contours to

describe the impact by the target circuit on HW, according to power consumption of the target and its location. The contours are further utilized to predict the location of the target when it is unknown. The accuracy is evaluated on AWS EC2 F1 FPGA instances. Future extensions are proposed that may improve the work and overcome its current limitations. If successfully realized, this work could facilitate better detection of malicious circuits or offer guidance for implementing remote side channel attacks on multi-tenant FPGAs.

Collectively, the topics covered in this dissertation can help to increase the security of applications on cloud FPGAs and serve to remind of possible security threats that arise with FPGA multi-tenancy.

BIBLIOGRAPHY

[1] Mordor Intelligence, "Field programmable gate array market size & share analysis - growth trends & forecasts (2023 - 2028)," https://www.mordorintelligence.com/industry-reports/field-programmable-gate-array-fpga-market, 2023.

[2] MarketsandMarkets, "Market leadership - FPGA market," https://www.marketsandmarkets.com/ResearchInsight/fpga-market.asp, 2020.

[3] Amazon Web Services, "Amazon EC2 F1 instances," https://aws.amazon.com/ec2/instance-types/f1/, 2020.

[4] Microsoft, "Project Catapult," https://www.microsoft.com/en-us/research/project/project-catapult/, 2017.

[5] A. Putnam, A. M. Caulfield, E. S. Chung, D. Chiou, K. Constantinides, J. Demme, H. Esmaeilzadeh, J. Fowers, G. P. Gopal, and J. Gray, "A reconfigurable fabric for accelerating large-scale datacenter services," *ACM SIGARCH Computer Architecture News*, vol. 42, no. 3, pp. 13–24, 2014.

[6] S. Bhunia, M. S. Hsiao, M. Banga, and S. Narasimhan, "Hardware Trojan attacks: Threat analysis and countermeasures," *Proceedings of the IEEE*, vol. 102, no. 8, pp. 1229–1247, 2014.

[7] C. Jin, V. Gohil, R. Karri, and J. Rajendran, "Security of cloud FPGAs: A survey," *arXiv preprint arXiv:2005.04867*, 2020.

[8] M. Stojilović, K. Rasmussen, F. Regazzoni, M. B. Tahoori, and R. Tessier, "A visionary look at the security of reconfigurable cloud computing," *Proceedings of the IEEE*, vol. 111, no. 12, pp. 1548–1571, 2023.

[9] Amazon Web Services, "AWS EC2 FPGA HDK+SDK errata," https://github.com/aws/aws-fpga/blob/master/ERRATA.md, 2017.

[10] I. Giechaskiel, K. B. Rasmussen, and J. Szefer, "Measuring long wire leakage with ring oscillators in cloud FPGAs," in *2019 29th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2019, pp. 45–50.

[11] T. Sugawara, K. Sakiyama, S. Nashimoto, D. Suzuki, and T. Nagatsuka, "Oscillator without a combinatorial loop and its threat to FPGA in data centre," *Electronics Letters*, vol. 55, no. 11, pp. 640–642, 2019.

[12] A. Boutros, M. Hall, N. Papernot, and V. Betz, "Neighbors from hell: Voltage attacks against deep learning accelerators on multi-tenant FPGAs," in *2020 International Conference on Field-Programmable Technology (ICFPT)*. IEEE, 2020, pp. 103–111.

[13] G. Provelengios, D. Holcomb, and R. Tessier, "Power wasting circuits for cloud FPGA attacks," in *2020 30th International Conference on Field-Programmable Logic and Applications (FPL)*. IEEE, 2020, pp. 231–235.

[14] ——, "Mitigating voltage attacks in multi-tenant FPGAs," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 14, no. 2, pp. 1–24, 2021.

[15] M. M. Alam, S. Tajik, F. Ganji, M. Tehranipoor, and D. Forte, "RAM-Jam: Remote temperature and voltage fault attack on FPGAs using memory collisions," in *2019 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*. IEEE, 2019, pp. 48–55.

[16] D. R. Gnad, F. Oboril, S. Kiamehr, and M. B. Tahoori, "Analysis of transient voltage fluctuations in FPGAs," in *2016 International Conference on Field-Programmable Technology (ICFPT)*. IEEE, 2016, pp. 12–19.

[17] D. R. Gnad, F. Oboril, and M. B. Tahoori, "Voltage drop-based fault attacks on FPGAs using valid bitstreams," in *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2017, pp. 1–7.

[18] D. Mahmoud and M. Stojilović, "Timing violation induced faults in multi-tenant FPGAs," in *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2019, pp. 1745–1750.

[19] Y. Luo, C. Gongye, S. Ren, Y. Fei, and X. Xu, "Stealthy-shutdown: Practical remote power attacks in multi-tenant FPGAs," in *2020 IEEE 38th International Conference on Computer Design (ICCD)*. IEEE, 2020, pp. 545–552.

[20] J. Krautter, D. R. Gnad, and M. B. Tahoori, "FPGAhammer: Remote voltage fault attacks on shared FPGAs, suitable for DFA on AES," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 44–68, 2018.

[21] Y. Luo, C. Gongye, Y. Fei, and X. Xu, "Deepstrike: Remotely-guided fault injection attacks on DNN accelerator in cloud-FPGA," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2021, pp. 295–300.

[22] P. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," in *Advances in Cryptology—CRYPTO'99: 19th Annual International Cryptology Conference*. Springer, 1999, pp. 388–397.

[23] O. Lo, W. J. Buchanan, and D. Carson, "Power analysis attacks on the AES-128 S-box using differential power analysis (DPA) and correlation power analysis (CPA)," *Journal of Cyber Security Technology*, vol. 1, no. 2, pp. 88–107, 2017.

[24] E. Brier, C. Clavier, and F. Olivier, "Correlation power analysis with a leakage model," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2004, pp. 16–29.

[25] K. M. Zick, M. Srivastav, W. Zhang, and M. French, "Sensing nanosecond-scale voltage attacks and natural transients in FPGAs," in *Proceedings of the ACM/SIGDA international symposium on Field programmable gate arrays*, 2013, pp. 101–104.

[26] L. L. Shen, I. Ahmed, and V. Betz, "Fast voltage transients on FPGAs: Impact and mitigation strategies," in *2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 2019, pp. 271–279.

[27] S. Moini, X. Li, P. Stanwicks, G. Provelengios, W. Burleson, R. Tessier, and D. Holcomb, "Understanding and comparing the capabilities of on-chip voltage sensors against remote power attacks on FPGAs," in *IEEE 63rd International Midwest Symposium on Circuits and Systems (MWSCAS)*. IEEE, 2020, pp. 941–944.

[28] M. Zhao and G. E. Suh, "FPGA-based remote power side-channel attacks," in *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2018, pp. 229–244.

[29] F. Schellenberg, D. R. Gnad, A. Moradi, and M. B. Tahoori, "An inside job: Remote power analysis attacks on FPGAs," in *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2018, pp. 1111–1116.

[30] O. Glamočanin, L. Coulon, F. Regazzoni, and M. Stojilović, "Are cloud FPGAs really vulnerable to power analysis attacks?" in *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2020, pp. 1007–1010.

[31] S. Tian, S. Moini, A. Wolnikowski, D. Holcomb, R. Tessier, and J. Szefer, "Remote power attacks on the versatile tensor accelerator in multi-tenant FPGAs," in *2021 IEEE 29th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 2021, pp. 242–246.

[32] S. Moini, S. Tian, D. Holcomb, J. Szefer, and R. Tessier, "Remote power side-channel attacks on BNN accelerators in FPGAs," in *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2021, pp. 1639–1644.

[33] ——, "Power side-channel attacks on BNN accelerators in remote FPGAs," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 11, no. 2, pp. 357–370, 2021.

[34] I. Giechaskiel, K. B. Rasmussen, and K. Eguro, "Leaky wires: Information leakage and covert communication between FPGA long wires," in *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*, 2018, pp. 15–27.

[35] I. Giechaskiel, K. B. Rasmussen, and J. Szefer, "Measuring long wire leakage with ring oscillators in cloud FPGAs," in *2019 29th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2019, pp. 45–50.

[36] I. Giechaskiel and J. Szefer, "Information leakage from FPGA routing and logic elements," in *2020 IEEE/ACM International Conference on International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2020, pp. 1–9.

[37] C. Ramesh, S. B. Patil, S. N. Dhanuskodi, G. Provelengios, S. Pillement, D. Holcomb, and R. Tessier, "FPGA side channel attacks without physical access," in *2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 2018, pp. 45–52.

[38] C. Jin, "8-bit datapath hardware implementation of AES," https://github.com/ChengluJin/8bit_datapath_AES, 2017.

[39] G. Provelengios, C. Ramesh, S. B. Patil, K. Eguro, R. Tessier, and D. Holcomb, "Characterization of long wire data leakage in deep submicron FPGAs," in *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2019, pp. 292–297.

[40] D. R. Gnad, C. D. K. Nguyen, S. H. Gillani, and M. B. Tahoori, "Voltage-based covert channels using FPGAs," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 26, no. 6, pp. 1–25, 2021.

[41] S. Tian and J. Szefer, "Temporal thermal covert channels in cloud FPGAs," in *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2019, pp. 298–303.

[42] T. M. La, K. Matas, N. Grunchevski, K. D. Pham, and D. Koch, "FPGADefender: Malicious self-oscillator scanning for Xilinx Ultrascale+ FPGAs," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 13, no. 3, pp. 1–31, 2020.

[43] J. Krautter, D. R. Gnad, and M. B. Tahoori, "Mitigating electrical-level attacks towards secure multi-tenant FPGAs in the cloud," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 12, no. 3, pp. 1–26, 2019.

[44] G. Provelengios, D. Holcomb, and R. Tessier, "Characterizing power distribution attacks in multi-user FPGA environments," in *2019 29th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2019, pp. 194–201.

[45] ——, "Power distribution attacks in multitenant FPGAs," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 28, no. 12, pp. 2685–2698, 2020.

[46] S. S. Mirzargar, G. Renault, A. Guerrieri, and M. Stojilović, "Nonintrusive and adaptive monitoring for locating voltage attacks in virtualized FPGAs," in *2020 International Conference on Field-Programmable Technology (ICFPT)*. IEEE, 2020, pp. 288–289.

[47] Y. Luo and X. Xu, "A quantitative defense framework against power attacks on multi-tenant FPGA," in *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE, 2020, pp. 1–4.

[48] J. Krautter, D. R. Gnad, F. Schellenberg, A. Moradi, and M. B. Tahoori, "Active fences against voltage-based side channels in multi-tenant FPGAs," in *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2019, pp. 1–8.

[49] Z. Seifoori, S. S. Mirzargar, and M. Stojilović, "Closing leaks: Routing against crosstalk side-channel attacks," in *Proceedings of the 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2020, pp. 197–203.

[50] Y. Luo and X. Xu, "HILL: A hardware isolation framework against information leakage on multi-tenant FPGA long-wires," in *2019 International Conference on Field-Programmable Technology (ICFPT)*. IEEE, 2019, pp. 331–334.

[51] S. N. Dhanuskodi, S. Allen, and D. Holcomb, "Efficient register renaming architectures for 8-bit AES datapath at 0.55 pJ/bit in 16-nm FinFET," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 28, no. 8, pp. 1807–1820, 2020.

[52] X. Li, P. Stanwicks, G. Provelengios, R. Tessier, and D. Holcomb, "Jitter-based adaptive true random number generation for FPGAs in the cloud," in *2020 International Conference on Field-Programmable Technology (ICFPT)*. IEEE, 2020, pp. 112–119.

[53] ——, "Jitter-based adaptive true random number generation circuits for FPGAs in the cloud," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 16, no. 1, pp. 1–20, 2023.

[54] X. Li, R. Tessier, and D. Holcomb, "Precise fault injection to enable DFIA for attacking AES in remote FPGAs," in *2022 IEEE 30th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 2022, pp. 1–5.

[55] S. N. Dhanuskodi, X. Li, and D. Holcomb, "COUNTERFOIL: Verifying provenance of integrated circuits using intrinsic package fingerprints and inexpensive cameras," in *29th USENIX Security Symposium (USENIX Security 20)*, 2020, pp. 1255–1272.

[56] S. Moini, A. Deric, X. Li, G. Provelengios, W. Burleson, R. Tessier, and D. Holcomb, "Voltage sensor implementations for remote power attacks on FPGAs," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 16, no. 1, pp. 1–21, 2022.

[57] J. Soto, "Statistical testing of random number generators," in *the 22nd National Information Systems Security Conference*, vol. 10, no. 99. NIST Gaithersburg, MD, 1999, p. 12.

[58] G. Zheng, G. Fang, R. Shankaran, and M. A. Orgun, "Encryption for implantable medical devices using modified one-time pads," *IEEE Access*, vol. 3, pp. 825–836, 2015.

[59] C. Meijer and B. Van Gastel, "Self-encrypting deception: Weaknesses in the encryption of solid state drives," in *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2019, pp. 72–87.

[60] B. Yang, V. Rožic, M. Grujic, N. Mentens, and I. Verbauwhede, "ES-TRNG: A high-throughput, low-area true random number generator based on edge sampling," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 267–292, 2018.

[61] X. Yang and R. C. Cheung, "A complementary architecture for high-speed true random number generator," in *2014 International Conference on Field-Programmable Technology (ICFPT)*. IEEE, 2014, pp. 248–251.

[62] M. Majzoobi, F. Koushanfar, and S. Devadas, "FPGA-based true random number generation using circuit metastability with adaptive feedback control," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2011, pp. 17–32.

[63] A. Peetermans, V. Rozic, and I. Verbauwhede, "A highly-portable true random number generator based on coherent sampling," in *2019 29th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2019, pp. 218–224.

[64] J. Balasch, F. Bernard, V. Fischer, M. Grujić, M. Laban, O. Petura, V. Rožić, G. Van Battum, I. Verbauwhede, M. Wakker, and Y. Bohan, "Design and testing methodologies for true random number generators towards industry certification," in *2018 IEEE 23rd European Test Symposium (ETS)*. IEEE, 2018, pp. 1–10.

[65] L. Gaspar, V. Fischer, L. Bossuet, and R. Fouquet, "Secure extensions of FPGA soft core processors for symmetric key cryptography," in *6th International Workshop on Reconfigurable Communication-Centric Systems-on-Chip (ReCoSoC)*. IEEE, 2011, pp. 1–8.

[66] S. Zeitouni, J. Vliegen, T. Frassetto, D. Koch, A. R. Sadeghi, and N. Mentens, "Trusted configuration in cloud FPGAs," in *2021 IEEE 29th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 2021, pp. 233–241.

[67] P. F. Wolfe, R. Patel, R. Munafo, M. Varia, and M. Herbordt, "Secret sharing MPC on FPGAs in the datacenter," in *2020 30th International Conference on Field-Programmable Logic and Applications (FPL)*. IEEE, 2020, pp. 236–242.

[68] P. Kohlbrenner and K. Gaj, "An embedded true random number generator for FPGAs," in *Proceedings of the 2004 ACM/SIGDA 12th International Symposium on Field Programmable Gate Arrays*, 2004, pp. 71–78.

[69] A. Maiti, R. Nagesh, A. Reddy, and P. Schaumont, "Physical unclonable function and true random number generator: A compact and scalable implementation," in *Proceedings of the 19th ACM Great Lakes Symposium on VLSI*, 2009, pp. 425–428.

[70] V. Rozic, B. Yang, W. Dehaene, and I. Verbauwhede, "Highly efficient entropy extraction for true random number generators on FPGAs," in *2015 52nd ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2015, pp. 1–6.

[71] N. Deák, T. Györfi, K. Márton, L. Vacariu, and O. Cret, "Highly efficient true random number generator in FPGA devices using phase-locked loops," in *2015 20th International Conference on Control Systems and Computer Science*. IEEE, 2015, pp. 453–458.

[72] J. L. Danger, S. Guilley, and P. Hoogvorst, "High speed true random number generator based on open loop structures in FPGAs," *Microelectronics Journal*, vol. 40, no. 11, pp. 1650–1656, 2009.

[73] T. J. Yamaguchi, K. Ichiyama, H. X. Hou, and M. Ishida, "A robust method for identifying a deterministic jitter model in a total jitter distribution," in *2009 International Test Conference*. IEEE, 2009, pp. 1–10.

[74] L. Xu, Y. Duan, and D. Chen, "A low cost jitter separation and characterization method," in *2015 IEEE 33rd VLSI Test Symposium (VTS)*. IEEE, 2015, pp. 1–5.

[75] M. S. Turan, E. Barker, J. Kelsey, K. A. McKay, M. L. Baish, and M. Boyle, "Recommendation for the entropy sources used for random bit generation," *NIST Special Publication*, vol. 800, no. 90B, p. 102, 2018.

[76] C. Celi, "NIST SP800-90B entropy assessment," https://github.com/usnistgov/SP800-90B_EntropyAssessment, 2019.

[77] A. Rukhin, J. Soto, J. Nechvatal, M. Smid, E. Barker, S. Leigh, M. Levenson, M. Vangel, D. Banks, A. Heckert, J. Dray, and S. Vo, "A statistical test suite for random and pseudorandom number generators for cryptographic applications," National Institute of Standards and Technology, Tech. Rep., 2001.

[78] J. Choe and K. Shin, "A self-timed ring based TRNG with feedback structure for FPGA implementation," in *2020 International Conference on Electronics, Information, and Communication (ICEIC)*. IEEE, 2020, pp. 1–4.

[79] H. Hata and S. Ichikawa, "FPGA implementation of metasability-based true random number generator," *IEICE Transactions on Information and Systems*, vol. 95, no. 2, pp. 426–436, 2012.

[80] R. Della Sala, D. Bellizia, and G. Scotti, "A novel ultra-compact FPGA-compatible TRNG architecture exploiting latched ring oscillators," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 69, no. 3, pp. 1672–1676, 2021.

[81] N. N. Anandakumar, S. K. Sanadhya, and M. S. Hashmi, "FPGA-based true random number generation using programmable delays in oscillator-rings," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 67, no. 3, pp. 570–574, 2019.

[82] Y. Luo, W. Wang, S. Best, Y. Wang, and X. Xu, "A high-performance and secure TRNG based on chaotic cellular automata topology," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 67, no. 12, pp. 4970–4983, 2020.

[83] A. Moradi, O. Mischke, C. Paar, Y. Li, K. Ohta, and K. Sakiyama, "On the power of fault sensitivity analysis and collision side-channel attacks in a combined setting," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2011, pp. 292–311.

[84] A. Spruyt, A. Milburn, and Ł. Chmielewski, "Fault injection as an oscilloscope: Fault correlation analysis," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 192–216, 2021.

[85] Y. Li, K. Sakiyama, S. Gomisawa, T. Fukunaga, J. Takahashi, and K. Ohta, "Fault sensitivity analysis," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2010, pp. 320–334.

[86] K. Matas, T. M. La, K. D. Pham, and D. Koch, "Power-hammering through glitch amplification–attacks and mitigation," in *IEEE 28th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 2020, pp. 65–69.

[87] M. M. Alam, S. Tajik, F. Ganji, M. Tehranipoor, and D. Forte, "RAM-Jam: Remote temperature and voltage fault attack on FPGAs using memory collisions," in *2019 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*, 2019, pp. 48–55.

[88] N. F. Ghalaty, B. Yuce, M. Taha, and P. Schaumont, "Differential fault intensity analysis," in *2014 Workshop on Fault Diagnosis and Tolerance in Cryptography*. IEEE, 2014, pp. 49–58.

[89] N. F. Ghalaty, B. Yuce, and P. Schaumont, "Differential fault intensity analysis on PRESENT and LED block ciphers," in *International Workshop on Constructive Side-Channel Analysis and Secure Design*. Springer, 2015, pp. 174–188.

[90] G. Provelengios, D. Holcomb, and R. Tessier, "Power distribution attacks in multitenant FPGAs," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 28, no. 12, pp. 2685–2698, 2020.

[91] A. Deric and D. Holcomb, "Know time to die–Integrity checking for zero trust chiplet-based systems using between-die delay PUFs," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 391–412, 2022.

[92] A. Maiti and P. Schaumont, "Improved ring oscillator PUF: An FPGA-friendly secure primitive," *Journal of Cryptology*, vol. 24, pp. 375–397, 2011.

[93] J. Guajardo, S. S. Kumar, G.-J. Schrijen, and P. Tuyls, "FPGA intrinsic PUFs and their use for IP protection," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2007, pp. 63–80.

[94] M. Majzoobi, F. Koushanfar, and S. Devadas, "FPGA PUF using programmable delay lines," in *2010 IEEE International Workshop on Information Forensics and Security*. IEEE, 2010, pp. 1–6.

[95] X. Xin, J.-P. Kaps, and K. Gaj, "A configurable ring-oscillator-based PUF for Xilinx FPGAs," in *2011 14th Euromicro Conference on Digital System Design*. IEEE, 2011, pp. 651–657.

[96] J. H. Anderson, "A PUF design for secure FPGA-based embedded systems," in *2010 15th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2010, pp. 1–6.

[97] R. Maes, A. Van Herrewege, and I. Verbauwhede, "PUFKY: A fully functional PUF-based cryptographic key generator," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2012, pp. 302–319.

[98] U. Rührmair and D. Holcomb, "PUFs at a glance," in *2014 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2014, pp. 1–6.

[99] M. A. Usmani, S. Keshavarz, E. Matthews, L. Shannon, R. Tessier, and D. Holcomb, "Efficient PUF-based key generation in FPGAs using per-device configuration," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 2, pp. 364–375, 2018.