INVESTIGATING REMOTE DYNAMIC POWER ATTACKS FOR SECURING FPGA SYSTEMS

A THESIS SUBMITTED TO THE UNIVERSITY OF MANCHESTER
FOR THE DEGREE OF DOCTOR OF ENGINEERING
IN THE FACULTY OF SCIENCE AND ENGINEERING

2023

Tuan M. La
School of Engineering
Department of Computer Science

Contents

Word Count: 22765

List of Tables

List of Figures

Abstract

INVESTIGATING REMOTE DYNAMIC POWER ATTACKS FOR SECURING FPGA SYSTEMS
Tuan M. La
A thesis submitted to The University of Manchester
for the degree of Doctor of Engineering, 2023

As FPGAs are now offered on the cloud, this exposes many potential security issues. This project investigates current security issues and challenges when deploying FPGAs in the cloud as well as using FPGAs in a multi-tenancy scenario. An in-depth investigation of recent FPGA architectures has been carried out to study the possibility to create and customise malicious circuits to exploit power side-channel and denial-of-service attacks on FPGAs. This thesis identified that self-toggling circuits such as ring-oscillators and glitch amplifiers not only pose a threat to the confidentiality but also to the reliability of an FPGA system. On the one hand, ring-oscillators could be used to sense electrical activities due to their sensitivity to voltage fluctuation. On the other hand, when the self-toggling circuit is tuned, it could draw excessive power to effectively overwhelm the power supply circuit of a system in case of a denial-of-service attack.

Therefore, it is of paramount importance to assess the power attack potential of a design as early as possible. With the detailed information studied, we could accurately detect malicious sources at a very early stage before the design is programmed onto the FPGA board. Moreover, this thesis provides a characterisation of waste power potential on modern data centre FPGAs to assist in analysing the level of power attack threat which could also come from the abuse of power-hungry circuits such as cryptographic algorithm calculation.

Declaration

No portion of the work referred to in this thesis has been submitted in support of an application for another degree or qualification of this or any other university or other institute of learning.

Copyright

Acknowledgements

First of all, I would like to express my deep gratitude to my supervisor, Dr. Dirk Koch, for his guidance, enthusiastic support, and encouragement. He has been guiding me since the beginning with his long-term vision, invaluable experience and expertise. I also thank Dr. James Garside, my co-supervisor, for his valuable support.

I would like to thank my friends Khoa Dang Pham, Kaspar Matas, Anuj Vaishnav, and Joseph Powell for their help, support, encouragement, deep insights, thorough discussions, and collaborations during my study. People from our APT group are also thankful for the positive and professional research environment.

Finally, I would like to thank my parents, my family, and my Vietnamese friends who helped me to achieve this milestone.

This work is kindly supported by the UK National Cyber Security Centre through the project rFAS (grant agreement 4212204/RFA 15971). We also thank the Xilinx University Program for providing tools and boards.

Chapter 1

Introduction

1.1 Motivation

FPGAs are widely offered in cloud data centres (e.g., [13]), and there is consequently a strong need to investigate FPGA hardware security in the remote environment. This is because of the difference in the environment where FPGAs are utilised. Traditionally, FPGAs are used by individual entities so FPGA security was related to IP theft and attacks that could be done with sophisticated lab equipment such as oscilloscopes [20] and electromagnetic-field sensors [41]. In the cloud environment, virtually anybody can access an FPGA as an accelerator. In 2016, Amazon rolled out F1 instances which are the FPGA-based cloud computing services. From that, there has been an increase of studies on remote security of FPGAs. Figure 1.1 clearly depicts the movement of studies on security of FPGAs from physical proximity to remote deployment. Crosstalk coupling is a distinctive example where the crosstalk effect has been known for a long time but only when FPGAs are deployed remotely, the security aspect of the effect is starting to emerge. On the other hand, we have not seen electromagnetic vulnerabilities on the cloud because it usually requires the electromagnetic-field sensor to be in close contact with the FPGA fabric.

In the cloud, registered users are allowed to run their FPGA designs on the cloud hardware. This may enable someone to cause potential harm to the FPGA equipment [99] (see Figure 1.2). To prevent potential malicious designs from being deployed on an FPGA instance, Cloud Service Providers (CSP) commonly demand uploading netlists to run Design Rule Checks (DRCs). Tests mostly include static timing analysis and detecting *Combinatorial feedback paths*. These violations are currently used for FPGA attacks [99]. For example, the FPGA vendor Xilinx has a DRC to indicate

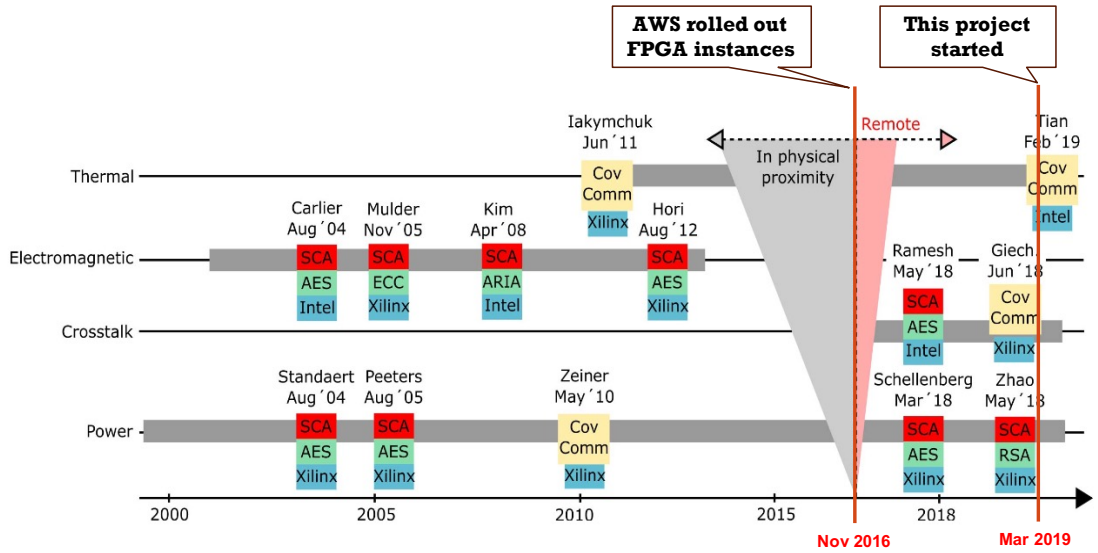


Figure 1.1: The main figure is taken from a survey in [99] where it showed the timeline of the key research contributions (not an exhaustive list). Gray horizontal bars start at the earliest reported successful attack. The added information are markers when Amazon introduced FPGAs on the cloud and when this project was started.

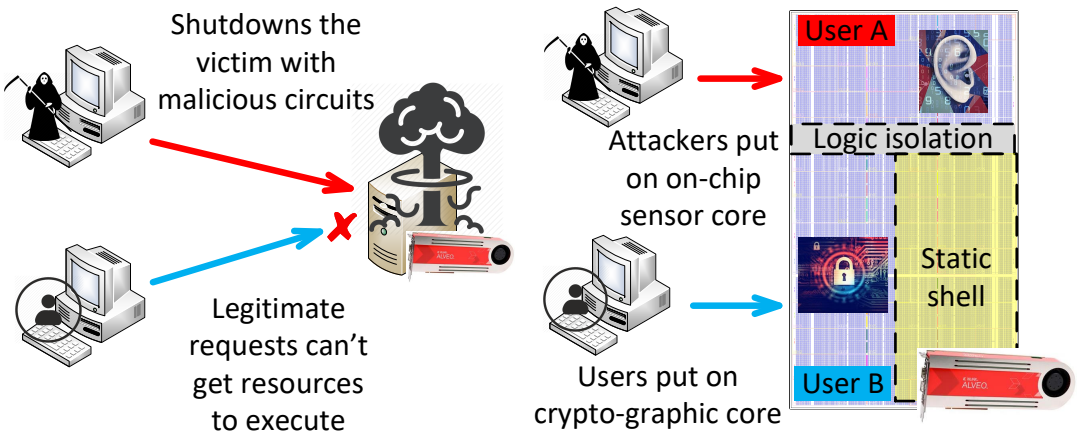


Figure 1.2: Attacks on system availability and system confidentiality.

when a LUT-based combinatorial loop is used to prevent implementing free-running oscillators in the design (*DRC LUTLP-1*) which can be used to leak information or environmental parameters of a system. Additionally, accepting netlists only ensures that generated bitstreams are not corrupted through any post-processing step. During operation, power monitoring can be used to warn or stop potential malicious activities (e.g., through clock gating if the supply power consumption reaches a critical level [12]).

Current DRCs provided by the FPGA vendors are insufficient for security, as will be described later in Chapter 3 with several examples of malicious FPGA designs that can be deployed on AWS instances. Moreover, monitoring is passive as it only detects malicious circuits *after these circuits are deployed*.

This thesis will survey current security issues of using FPGAs in the cloud, characterise power-hammering threats relating to self-oscillating circuits, and propose additional countermeasures to mitigate power-hammering attacks. The thesis will later focus on the availability property especially in Denial-of-Service attacks. This work not only aims to provide the base for potential multi-tenancy scenarios but also to move FPGA usage from the present FPGA-enabled Acceleration-as-a-Service (AaaS) offerings [13] to full FPGA-as-a-Service (FaaS) offerings [94, 11], where users may upload their own bitstreams without the risk that user IPs are possibly leaked by CSPs. Providing a holistic study on the power-hammering potential is the key to deciding what level of detection and mitigation strategies are needed in order to allow that FaaS.

1.2 Objectives and challenges

This project aims to characterise the risk of power-hammering, which is the process of damaging equipment, denying services, or introducing faults through imposed excessive waste power consumption. In detail, this project aims at providing a holistic view of power-hammering on FPGAs by providing a model to detect and evaluate the threat of various scenarios an attacker may use for wasting power.

This research includes an in-depth investigation into FPGA resources such as Look-Up Tables (LUT), Switch Matrices (SWBOX), Cascading Multiplexers (MUX), Digital Signal Processing Blocks (DSP), Carry Chain logic (CARRY), Block RAMs (BRAM), routing resources, and clock generation and distribution resources. A common source of power in a digital circuit is dynamic power which results from the switching activity of a circuit. The most excessive sources of switching possible on an FPGA result from oscillators. Oscillators are not used in digital circuits that follow the Register

Transfer Level (RTL) model. In order to catch any kind of oscillator, this work does not only consider ring-oscillators (as the FPGA vendor DRCs are limited to) but we will explore *any possible combination to create self-oscillating circuits on FPGAs*. Throughout the exploration, related information is obtained from the circuits such as frequency and power consumption. This is then used to establish a way to quantify the power-hammering potential.

To emphasise the importance of the problem, we not only need to conduct experiments in a lab-controlled environment but also need to demonstrate real-world attacks in a responsible manner. Using public knowledge, we need to investigate the implementation of FPGA security infrastructure implemented by cloud providers and derive an attack plan accordingly. It should be noted that we do not use social engineering to gain advantages to infiltrate the cloud infrastructure.

In order to propose an effective countermeasure, this work will investigate holes in present power-hammering mitigation strategies and will develop complementary techniques that provide substantially better protection and that can, for example, prevent an attacker from deploying any kind of self-oscillating circuits. It should be emphasised that the primary countermeasure should be preventing the malicious circuits from being loaded on the FPGA fabric, not just monitoring after loading it. This project aims at providing flexible security checks that could be implemented either before or after the bitstream generation stage. The former could be done by embedding a custom DRC to the existing Xilinx FPGA design flow. The latter is implemented as an additional step that scans the bitstream before it is being loaded onto the FPGA fabric. Our solution enables cloud service providers to easily and flexibly integrate the checks into the existing FPGA design flow.

1.3 Contributions

Throughout the course of the research, this project has made the following contributions:

- **In-depth study on self-oscillators in FPGAs**: We provide an in-depth study on self-oscillating circuits for the most commonly used FPGA devices provided by the vendor Xilinx (now AMD) for data centre acceleration. It not only extends the known variants of possible self-oscillating sources but also examines individual self-oscillators in terms of frequency, power consumption, and power-hammering potential. The study contributes to the publication [110].

- **Dynamic power model for wiring resources**: We examine the energy consumed for each toggle on the main wiring resources in Xilinx UltraScale+ FPGAs. This reveals that by using enough routing resources, a power-hammering attack could be mounted using standard RTL designs.

- **Real-world case study of power-hammering attacks**: We provide a demonstration of the vulnerability of FPGA-based cloud infrastructures by deploying carefully designed malicious circuits. The results show that the current defence mechanisms are not adequate to protect the cloud infrastructure from the new type of Denial-of-Service attack. Moreover, our attacks demonstrated that it is possible to leak sensitive information, such as the cloud service providers scheduling decisions. The study contributes to the publication [109].

- **FPGA malicious scanning mechanism**: Given the severity of current security issues in FPGA-based cloud systems, we propose a scanning mechanism that can effectively detect all currently known self-oscillating circuits among other malicious circuits as well as easily maintain, update, and embedded into FPGA-based cloud infrastructure. The Python implementation of the scanning mechanism was done by the PhD Student Kaspar Matas. My main contribution is the bitstream decoding for generating the FPGA architecture graph and TCL implementation of the FPGA virus scanner to embed it as a custom DRC. This study contributes to the publication [110]. The FPGA virus scanner can detect:

  - **Combinatorial feedback loops** which include more complex oscillators which were not discovered previously.
  - **Abnormal fanouts** which could be used to draw extra power through branching one oscillator to multiple routing resources.
  - **Prohibited ports** and **antennas** which could indicate misused connection such as restricted IO ports.
  - **Prohibited paths** indicate the potential use of the crosstalk coupling effect.
  - **Short-circuit** is another way to draw extra power through static power consumption.

1.4 Thesis outline

The rest of this thesis is organised into the following chapters:

- **Chapter 1**: provides the introduction, motivation, and context for this project.
- **Chapter 2**: describes the background and related works of current security issues of FPGAs at the electrical level. We especially focus on attacks and defence related to power-hammering.
- **Chapter 3**: characterises the FPGA power-hammering potential. This chapter describes and discusses the study on self-oscillating circuits. We also explore the contribution of routing resources to the total dynamic power consumption. Furthermore, we provide a way to quantify the power-hammering potential.
- **Chapter 4**: demonstrates a case study of a Denial-of-Service attack on Amazon Cloud Service F1 instances which are equipped with FPGAs.
- **Chapter 5**: describes a malicious circuit scanning mechanism. In this chapter, we also evaluate our system and discuss interesting results when scanning both legitimate and malicious circuits. Additionally, we integrate the malicious scanning mechanism into the existing FPGA development flow via a custom DRC.
- **Chapter 6**: summarises this thesis and discusses future work, which has been enabled by this project.

1.5 Publications

The research conducted throughout this project has been produced through the three research articles listed below.

1. K. Matas, **T. M. La**, K. D. Pham, and D. Koch, **Power-hammering through Glitch Amplification - Attacks and Mitigation**, in *IEEE International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2020

2. **T. M. La**, K. Matas, N. Grunchevski, K. D. Pham, and D. Koch, **FPGADefender: Malicious Self-Oscillator Scanning for Xilinx UltraScale+ FPGAs**, in *ACM Transaction Reconfigurable Technology (TRETS)*, 2020

3. **T. M. La**, K. D. Pham, J. Powell, and D. Koch, **Denial-of-Service on FPGA-based Cloud Infrastructure - Attack and Defense**, in *IACR Transactions on Cryptographic Hardware and Embedded Systems (TCHES)*, 2021

Additionally, this project also contributes abstracts and demos to several venues listed below.

1. K. Matas, **T. M. La**, N. Grunchevski, K. D. Pham, and D. Koch, **Invited Tutorial: FPGA Hardware Security for Datacenters and Beyond**, in *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA)*, 2019

2. **T. M. La**, K. Matas, K. D. Pham, and D. Koch, **Securing FPGA Accelerators at the Electrical Level for Multi-tenant Platforms**, in *International Conference on Field-Programmable Logic and Applications (FPL)*, 2020

3. **T. M. La**, K. Matas, J. Powell, K. D. Pham, and D. Koch, **Demo: A Closer Look at Malicious Bitstreams**, in *International Conference on Field-Programmable Logic and Applications (FPL)*, 2020

1.6 List of Abbreviations

Abbreviation	Meaning
AWS	Amazon Web Service
AaaS	Acceleration as a Service
BRAM	Block Random Access Memory
CARRY	Carry Chain logic
CLA	Carry Look Ahead
CSP	Cloud Service Provider
DFA	Differential Fault Analysis
DPA	Differential Power Analysis
DRC	Design Rule Check
DSP	Digital Signal Processing block
DoS	Denial of Service
EM	Electromagnetic
FaaS	FPGA as a Service
HALT	Higher Abstraction Level Threat
IP	Intellectual Property
LUT	Look Up Table
MELT	Malicious Electrical Level Threat
MUX	Multiplexer
RO	Ring Oscillator
SPA	Simple Power Analysis
SWBOX	switch box/switch matrix
TDC	Time-to-Digital Converter

Chapter 2

Background and related works

The flexibility and power efficiency provided by hardware configuration have led to the integration of FPGAs on cloud infrastructures. Studies have shown that FPGA-based hardware acceleration can achieve up to 25X better performance per watt and 50-70x latency improvement compared to CPU/GPU implementations in some applications [118]. This has led to the start of FPGA-enabled Cloud Computing Services (e.g., Amazon F1 instances [59]). The introduction of FPGA instances by Amazon Web Service has caused a large body of related FPGA security research which we could identify as the "Amazon Effect".

In the dawn of the FPGA era, vendors believed that the security of an FPGA was primarily about protecting designs in terms of the Intellectual Property (IP) in configuration data (bitstream) in commercial electronic equipment [116]. Attack requirements vary from less physical modifications (such as bitstream manipulation in Malicious Electrical Level Threat (MELT) [53]) to more complicated setups that analyse electromagnetic fields [41]. However, as FPGAs become popular, research showed that FPGAs are exposed to more severe threats than unauthorised uses of an IP. For instance, when a bitstream is manipulated, it can cause short-circuit and lead to irreversible damage [53]. Due to their low-level programmability, FPGAs introduce new threats far beyond what is commonly known from traditional CPU/GPU systems. For instance, modules running on an FPGA may include circuits being able to measure system states at high accuracy which may open physical side-channels that are not available in commonly used attack scenarios.

Before FPGAs emerged in cloud computing, those who wanted to explore FPGAs required physical access and mostly used them locally. In that scenario, owners have full accessibility and responsibility to the data that is processed inside FPGAs, thus

it is virtually impossible for others to steal the IP or compromise the processed data. However, this usage scenario also omits the advantages of resource pooling which is commonly used in the cloud environment. After the integration of FPGAs into the cloud, clients who want to utilise FPGAs are not required to possess an FPGA board physically. Scheduling policies allow cloud service providers to assign FPGA instances to individual clients. Therefore, clients do not have physical accessibility nor can take full responsibility to protect their IPs and data. Moreover, future multi-tenant scenarios are expected to provide better utilisation and power efficiency compared to a single user per fabric scheme. As a result, attack scenarios gradually change from local attacks to remote attacks as a reflection of the FPGA's immigration to the cloud.

Previous research has established the foundation for FPGA security [95] and also specifically focused on the implementation of cryptographic algorithms on FPGAs [113]. In response to the threats, FPGA vendors have been providing solutions to hinder and mitigate the attack channels in order to balance the cost of investing in countermeasures and the cost of breaking the system [96, 98, 104]. For instance, the Device DNA was introduced with Xilinx Spartan-3A for tracking purposes and preventing design cloning [117]. However, as a consequence of the *Amazon Effect*, new vulnerabilities give attackers less dependence on expensive equipment to perform attacks which lowers the cost of breaking the system. Therefore, the security in FPGAs needs to be reinvestigated and addressed again. This chapter aims to classify security threats focusing on exploiting the uniqueness of FPGAs' natures that makes them more vulnerable in the cloud compared to other hardware platforms.

In personal computing devices (e.g., PCs, laptops, smartphones), cyber security usually refers to reducing the threats of stealing, exposing, altering, disabling, or destroying information through unauthorised access to computer systems [81]. To ensure the security of a hardware system, administrators need to ensure an infrastructure to be trustworthy. In FPGA-based systems, configuration data - whether or not created by users - is considered as a part of the infrastructure then that must be protected. It should be noted that although social engineering remains the most prominent way of committing cyber crimes [74], trustworthy infrastructure plays an equally important role before putting any secure applications on top of that. Therefore, we will not discuss social engineering in this thesis.

There are several ways to categorise physical side-channel attacks. Either by physical phenomena such as power, electromagnetic, thermal, sound, crosstalk coupling, photonic emission or by proximity [99]. Regarding power attacks, we classify them

based on the aggression of the attacks.

- Aggressive attacks are designed to cause damage and severe breach of integrity and availability of the whole system.

- Nonaggressive attacks are designed to stealthily capture useful information and ultimately cause a breach of confidentiality.

In this chapter, we will discuss attacks and countermeasures on FPGA-based systems with a focus on power attacks

2.1 Security of FPGA systems

In terms of confidentiality in FPGA systems, most attacks are originated from implementation threats of security elements itself such as side-channel attacks. Currently, compromising confidentiality such as leaking information is drawing special attention (e.g., [72, 102, 112]). Side-channel attacks are well-known attacks in FPGAs in which attackers gain information from the implementation of a system instead of striking the weaknesses of the implemented algorithm. In addition to side-channels to the shared memory location (e.g., rowhammer in [132]), various further physical effects have been used to implement side-channels on FPGAs, including power analysis, electromagnetic analysis, thermal channel, fault analysis, and crosstalk coupling.

In terms of compromising the integrity of a system, it can be classified into two types: data integrity and system integrity. Data integrity refers to the accuracy and consistency of data over its entire life cycle. A system is said to have system integrity if it can be trusted to correctly execute its desired functions [82]. In FPGAs, we focus on the most current attacks that are related to improper data modification or fault injection in order to reveal a secret key that eventually affects the confidentiality of a system. For a classification of fault injection attacks in general hardware implementation, we refer to the study in [55].

Regarding the availability of a system, there has not been a remote attack that specifically affects the stability and reliability of an FPGA-based cloud infrastructure. However, in this thesis, we will discuss the possibility of altering power consumption to deploy an attack on availability.

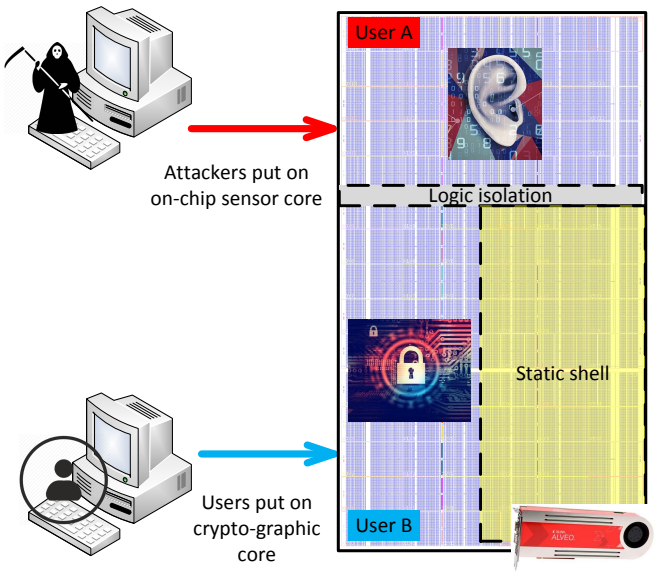


Figure 2.1: An illustration of the *eavesdropping* scenario in an FPGA.

2.1.1 Power analysis

Power side-channel attacks which exploit the inherited power patterns of typically a cryptographic core was first introduced by Kocher in 1999 [86]. The research was based on the fact that most operations in a CPU or a custom accelerator have data-dependent or operation-dependent power consumption. For instance, a Flip-Flop (in a CPU register file or an accelerator running on an FPGA) will consume more power if its state changes as compared to if the state does not change. Additionally, regarding static power, driving a stable value on a wire also consumes different level of power. This is because different transistors are active to hold the value of a wire. When driving a wire high, the parasitic capacitance of a wire gets charged from PMOS but when driving a wire low, it gets discharged through NMOS. Thus, the different in the characteristics of the transistors causes the different in static power consumption even when the state does not change. The research had shown two attack scenarios that are effective to two different cryptographic implementations. In Simple Power Analysis (SPA), attackers use just one power trace to exploit the operation profile of a system. However, SPA may not be practical if there is significant noise in the system. When the power consumption difference is small and interfered with other power-consuming blocks, Differential Power Analysis (DPA) yields better results compared to SPA. DPA requires more power traces through longer observation windows. Correlation analysis is then applied to retrieve information (e.g., an AES key). Moreover, DPA can also be applied

to almost any symmetric or asymmetric cryptographic algorithm including DES [46] and AES [45], RSA [111] and elliptic curve based public key encryption [20]. Therefore, attackers can perform the DPA attack to cryptographic implementations on any hardware platform without exception to FPGAs. The first experimental results of DPA on FPGAs were published by Ors in 2003 [20]. In general, power consumption can be measured directly by using a shunt resistor with an oscilloscope in the original work of Kocher, but it can also be measured indirectly through the supply voltage fluctuation as shown in recent research [44]. We would like to stress that this type of attack is fundamentally based on correlation analysis instead of precise power measurement. Therefore, attackers can use any means of correlating power traces in order to perform the attack such as current variances, voltage variances or even electromagnetic emissions. A scenario where two tenants share the same FPGA is indicated in Figure 2.1. Moreover, if attackers can exploit indirect power measurements, they can potentially carry out remote attacks which are threatening the trustworthiness of the FPGA-based system in cloud computing. A study in [80] constructs an on-chip power monitor using ring-oscillators and demonstrates a successful power analysis attack running on an RSA cryptomodule on an FPGA fabric. This is based on the fact that ring-oscillators, a variant of the self-oscillating circuit, oscillate in a frequency regime defined by the delay line which constructs them; and the delay line is dependent on physical parameters such as temperature and voltage drops. This property makes ring-oscillators an ideal tool to sense voltage drops and temperature remotely.

Countermeasures for power analysis attacks could be divided into passive and active strategies. Active actions include upgrading encrypting modules to mask or hide the power patterns, as shown in [97]. On the other hand, passive actions include monitoring and detecting when a power attack is happening in the system [2, 70]. In a cloud configuration, we summarise the advantages and disadvantages of those countermeasures in Table 2.1. Additionally, since physical access to the FPGA board is not permitted on the cloud, we think it could be easier to take preventative measures to identify the potential voltage sensing circuits such as ring-oscillators and reject the design.

2.1.2 Electromagnetic analysis

Physically, a movement of electric charges creates a current that induces an electromagnetic field. The electromagnetic (EM) emanation may be intentional as a result of

	Advantages	Disadvantages
Active actions	Normalise or obfuscate power patterns caused by cryptographic core such that power analysis attacks cannot be deployed.	Either costly or risky to modify the standard cryptographic core designs. Moreover, it could add more power consumption.
Passive actions	No need to change the cryptographic core designs.	Further actions need to be performed after detection. It may compromise the availability of system if an administrator decides to power cycle the system.

Table 2.1: Advantages and disadvantages of countermeasures to power analysis attacks

short bursts of current flow or unintentional as a consequence of electrical and electromagnetic coupling between components in close proximity [30]. Thus, currents flowing on internal FPGA routes can be used to characterise components inside the FPGA fabric. The measured signature can be used to analyse the intentional operation of a module in the same manner as power analysis attacks such as Simple Electromagnetic Attacks (SEMA) and Differential Electromagnetic Attacks (DEMA). The feasibility of EM attacks in FPGA has been proved in [41]. However, in order to capture emitted signals, electromagnetic probes are required to be placed close to the device (ideally no more than a wavelength away from the die). This makes those attacks infeasible to deploy remotely in the cloud.

Due to the same correlation analysis principle with power analysis attacks, countermeasures for power analysis attacks could be used for protection against EM attacks. Additionally, physical shielding such as additional metal layers can offer further resistance to EM attacks at extra cost. These physical countermeasures can be applied to any chip manufacturing process as described in [84]. Further investigation on magnetic shielding [79] showed to reduce as much as 6dB of EM emission. However, chip manufacturers need to balance the induced cost with the effectiveness of shielding.

2.1.3 Thermal channel

Thermal radiation is another physical effect that correlates with a running electronic device. It can be used to implement a covert channel for data communication, as shown in 2009 [60]. The following research shows that the channel enables bidirectional transmission and exchange of an arbitrary bitstream between two electrically separated parts of an FPGA circuit during its normal operation [108]. By taking advantage of the thermal sensitivity of ring-oscillators [73], temperature sensors can be created to form

a covert channel [102]. Not only does it transfer data at a slow speed of 1bps, but we also believe this attack is difficult to deploy and unfeasible for information leakage of states inside a module. It is because warming up and cooling down a device takes time as well as other modules could possibly heat up the FPGA die and create thermal noise. Nevertheless, thermal coupling allows reverse-engineering of the physical proximity of FPGA cloud instances, which may be used for coordinated attacks involving multiple instances simultaneously.

As we discussed above, the thermal attack is likely impractical. A countermeasure for it could align with heat dissipation which is also essential in server cooling systems. On the other hand, another approach can be preventing circuits from being deployed in data centres that are sensitive to temperature such as ring-oscillators.

2.1.4 Crosstalk coupling

Crosstalk is a physical issue studied in the design of integrated circuits for a long time. Crosstalks occur due to capacitive coupling between adjacent wires on a chip. The phenomenon was first noticed for FPGAs in 2001 by Wilton [105]. They showed that by acknowledging crosstalk in FPGAs, the average routing delay in the presence of crosstalk can be reduced by 7.1% in a representative FPGA architecture implemented in a 0.18μm technology. Therefore, the effect was initially considered as it would only affect performance and data integrity. In 2018, an experiment on FPGA routing resources showed that a long wire carrying a logical 1 reduces the propagation delay of other adjacent, but unconnected, long wires more than if that wire carries a logical 0 [51]. As a result, by detecting the propagation delay of a long wire, we can identify the state of a corresponding adjacent wire. To detect the propagation delay, ring-oscillators had been used once again because the frequency of a ring-oscillator fundamentally depends on the total propagation delay of components constructing the ring which includes wire segment delays and look-up-table delays (represent for a combinatorial logic element). This type of attack has been successfully deployed to recover the encryption key from an AES circuit on Intel Cyclone IV and Stratix V FPGAs [28]. However, this attack strictly requires both an attack wire and a victim wire to be located directly adjacent because the signal will be 20x weaker if the wires are separated by another wire [51]. This makes the attack sometimes impractical in the cloud as attackers may not be able to do the place and route freely.

Possible countermeasures for this type of attack can be using guard wires to separate sensitive signals from other adjacent wires or routing the sensitive module separately. Although it may come with area costs, it could be a reasonable trade-off if security is out weighted. On the other hand, preventing unreasonable circuits that can be used to exploit the difference in propagation delays such as ring-oscillators and enforcing strict bounding boxes can be a feasible solution.

2.1.5 Differential fault analysis

Fault attacks have long been an active area of research in cryptography. Those attacks exploit faults that are introduced during computation such as in key-scheduling or encrypting periods. The attack mechanism deeply depends on understanding the cryptography algorithms (refer to [76]). Differential Fault Analysis (DFA) was first introduced in 1997 by Eli Biham [40]. It can be considered as the most popular fault analysis method targeting cryptographic implementations. Originally, they used physical stimuli to directly interfere with the implementation of cryptography modules such as applying gentle physical stress to leak electric charges from memory cells which are assumed to hold the secret key or even using a narrow laser beam to cut wires or to manipulate the state inside a chip. On an FPGA, even though such kinds of physical interference are possible as shown in recent researches [21, 100], attackers can use a more gentle way to inject faults into an FPGA fabric by exploiting critical timing paths as shown in [63]. The study shows that ring-oscillators were used again to cause the voltage drops to increase the propagation delay in a controllable way by an attacker in order to create timing failure on critical paths. This attack has been demonstrated on a cryptographic core running at 111MHz. Compared with 20MHz of the recent attack using power analysis [80], it effectively shows the practicality of DFA attacks in FPGA systems.

Similar to EM attacks, physical protection methods during the manufacturing process could enhance the FPGA fabric's robustness so that it is less susceptible to voltage drops. Other countermeasures such as module redundancy, dual rail encoding, and error detection methods can be explored further in [76]. We present the most relevant defence mechanisms that can be applied in FPGAs in Table 2.2.

	Description
Increasing timing margins	This is to reduce the risk of timing failure but it may come with the cost of reducing operating frequency.
Module redundancy	This can be done by executing encryption/decryption several times in parallel or sequential and compare the results. If the results are equal, the computation is assumed to be correct. However, it may come with performance and/or area cost.
Detecting possible timing fault injection	We can use the same techniques to detect voltage drops that caused timing violations in power analysis section.
Dual rail implementation	To make it difficult for attacker, one could implement dual rail encoding. That is one wire always carries the inverted value of the other. Therefore, an attacker needs to flip the signal values of both rails simultaneously in order to induce a fault signal.

Table 2.2: Some possible defense mechanism for fault analysis attacks

2.1.6 Bitstream fault injection

Instead of generic hardware attacks like fault injection as discussed previously, bitstream fault injection (BiFI) is a new class of attack that can in particular be deployed on FPGAs. In general, one would reveal the secret key when having full knowledge about the internal bitstream structure. Many studies have been trying to reverse-engineer the proprietary bitstream structure of FPGAs such as [23, 39, 58, 43, 107, 67]. However, as vendors change the bitstream configuration together with the development of FPGAs, fully reverse-engineering the whole netlist from bitstream is still considered to be a difficult problem. Xilinx and Intel consider that the non-documented bitstream serves as a type of design obfuscation [98]. Therefore, instead of attempting to recover the entire netlist, current research focuses on specific parts of the bitstream. Studies in [89] and [1] target on modifying the substitution table used in AES cryptographic cores which provides the confusion operation - one of the two primitive operations in which a strong encryption algorithm should be built on according to Claude Shannon - in a cryptographic algorithm [27]. If an attacker can tamper with the substitution table (S-box), they can either decrypt the data without a secret key as it becomes a key-independent permutation in the DES or turn any cryptographic algorithms to be a linear function. This results in a weakened, less secure substitution which gives a wrong impression of security provided by a cryptographic core.

Although that type of attack is feasible, identifying the S-box in a bitstream requires major effort and calibration on different devices. A later study in [88] shows that manipulating the bitstream targeting LUT values only is not only easier to perform for

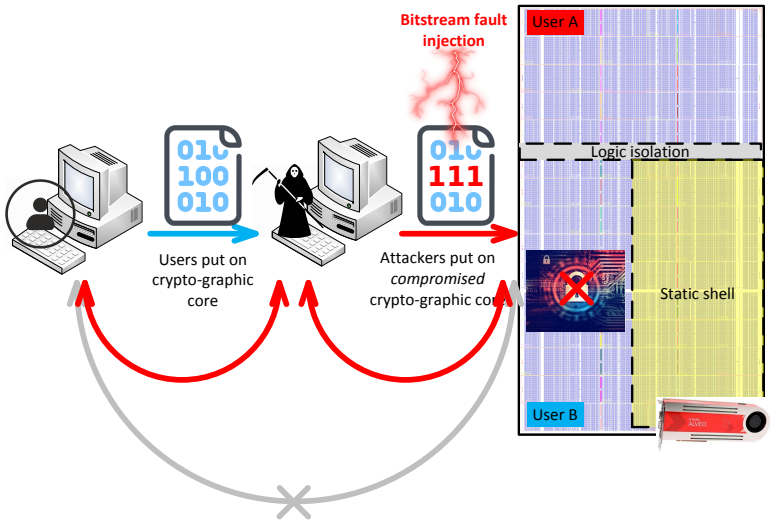


Figure 2.2: An illustration of the *bitstream fault injection* threat model in a multi-tenant computing environment.

a variety of FPGAs but also powerful and it does not require any reverse-engineering. In this research, a set of 15 bitstream manipulation rules to pollute the bitstream had been proposed. Then the collected ciphertexts are analysed by 11 hypotheses such as the ciphertext is simply the plaintext XORed with the key, therefore the secret key can be extracted with only a single pair of plaintext-ciphertext. Based on a Spartan-6 with 16 different AES encryption designs including the standard design on the NSA homepage, it shows promising results as some rules only need a few thousand random manipulations to get an exploitable faulty ciphertext. It corresponds to approximately 1.8 hours on average of bitstream manipulations to retrieve an exploitable ciphertext. Although the attack time varies dramatically depending on the targeted design, the maximum attack time in 16 different attacks is 20.82 hours (6.33 hours to attack the AES implementation from NSA) which makes the attack to be a severe problem in many practical realistic scenarios.

Additionally, S-boxes are often implemented with BRAMs to optimise the logic resource utilisation or performance. However, this still exposes the design to the fault injection vulnerability and it requires extremely low effort to extract the secret key. Study in [38] shows that the key from BRAM-based AES implementations can be extracted even faster than from LUT-based implementations, as shown in [88].

Although there is no evidence that these attacks could be deployed remotely as it normally requires more accessibility to manipulate the bitstream, these attacks could be prevented by checking the integrity of the bitstreams before programming them to

the FPGA fabric. It should be noted that bitstream encryption would not necessarily prevent this kind of attack because there have even been attacks on the integrated bitstream protection engine (e.g., Xilinx Virtex-II Pro [3], Xilinx Virtex-4 and Virtex-5 [6], Altera Stratix II and Stratix III [5, 87], Xilinx 5, 6, 7 series [4, 101], and Xilinx UltraScale [50]).

2.1.7 Configuration data leakage

Currently, cloud service providers introduced dedicated policies to protect the FPGA configuration data. Although there has not been a leak of user configuration data, it is possible for the cloud service provider to extract the configuration data. Study [106] proposed the idea of establishing a trustworthy development environment for FPGA-based cloud infrastructure. According to the study, by establishing a trusted execution environment (including the trusted configuration shell), configuration data from clients will be protected from leaking while cloud service providers could be assured that the bitstreams are thoroughly examined for illegal designs and malicious circuits.

2.1.8 Attacks on system availability

Denial-of-service (DoS) attacks are used to bring down operating infrastructures or to compromise states in other system components that stay outside the scope of an attacking module. An illustration of the DoS attack on an FPGA-based system is shown in Figure 2.3. At the electrical level, two means for DoS attacks had been utilised: short-circuits and power-hammering.

Short-circuits on FPGAs have a long history [53, 92, 91]. In 1999, a study shows that there are three levels of attacks on system availability: 1) electrical conflicts (MELT - Malicious Electrical Level Threat), 2) logical signals or corner case behaviours (SALT - Signal Alteration Logic Threat), and 3) software attacks (HALT - Higher Abstraction Level Threat) [53]. While SALT and HALT can be deployed on any hardware platform, electrical conflicts (MELT) are easier to deploy on FPGA platforms. The electrical conflict was caused by internal conflicts through shared column interconnect routing resources via pass transistors in the Altera Flex 8000 family. As it is a short-circuit attack, it could exceed the current limitation of a device and result in accelerated wear or the physical destruction of the whole system. Another research [22] demonstrated short-circuits within the multiplexers inside a switch matrix using a manipulated configuration bitstream resulting in a substantial current increase

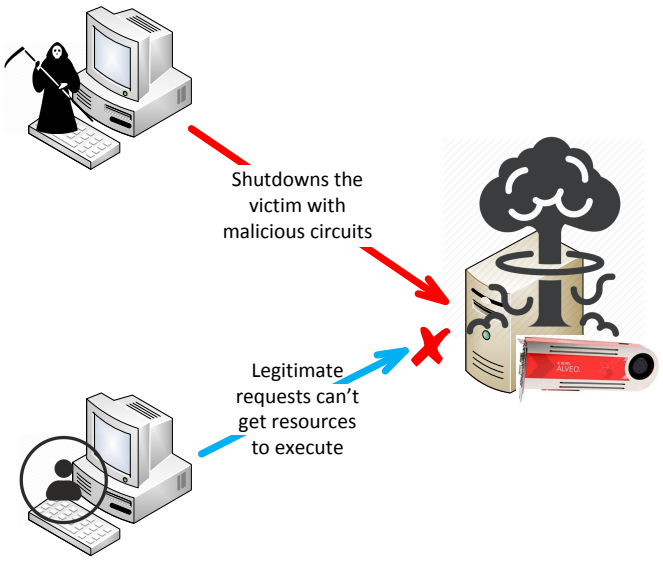


Figure 2.3: An illustration of the denial-of-service scenario. A user may try to shutdown an FPGA service in a data centre by sending malicious circuits such that legitimate requests from other users cannot use the FPGA resources. Short-circuits and power-hammering designs can be utilised for such attacks on the system availability. Furthermore, this kind of attack may potentially age or damage the equipment.

(with several *mA* extra current for a single routing multiplexer [22]).

Alternatively, power-hammering is another mechanism to carry out DoS attacks. All current power-hammering attacks [36] are based on fast toggling circuits in order to draw a substantial amount of dynamic power. As we will show in Chapter 3, it is possible to implement self-oscillating circuits running in the GHz frequency domain with a corresponding dynamic power footprint. In [36], a grid of ring-oscillators could be activated at an adjustable rate to stimulate resonance effects in the power supply regulation circuit. With this, several FPGA platforms such as Xilinx Virtex 6, Kintex 7, and Zynq-7000 FPGAs were crashed and, in some cases, required power cycling for bringing up boards back into service. Although ring-oscillators are usually flagged with a warning by the vendor design tool flows and hence, are not allowed to be deployed on common cloud or data centre infrastructures, recent research [52] has reported new ring-oscillator designs which can bypass such a Design Rule Check (DRC).

A proposed countermeasure like a bitstream antivirus scanner was implemented in Lattice iCE40 FPGAs [37]. We will discuss this concept in detail later in Chapter 5. On the other hand, we can monitor the system's health by analysing voltage parameters. For instance, Xilinx provides a way to monitor system temperature and voltage with

	Aggressive	Nonaggressive
Examples	Power-hammering, Fault Analysis	Power Analysis
Main Target	Integrity and Availability	Confidentiality
Countermeasure	Voltage Monitoring	Design Scanning, Power Trace Obfuscation

Table 2.3: Advantages and disadvantages of countermeasures to power analysis attacks

a resolution of 10-bit and a sample rate of 1 MSPS as in the UltraScale+ devices. However, voltage transient speed can be relatively high depending on attack scale, electrical characteristics, and process variation; therefore, the system monitor may not be able to capture the change. A novel soft-logic implemented voltage sensor is believed to be better than built-in ADCs as stated in [70]. The principle in the study is that voltage drops can increase the propagation delay of a delay chain. We can use that to detect the latency difference and therefore spot the correlation with voltage transients. The experimental results show that the monitoring is 500x faster than the 28nm Xilinx ADC, thus it can be a potential monitoring method to detect power attacks.

2.2 Attacks and defence on power aspect

As discussed above, the power attack can be categorised by aggression. Table 2.3 summarises power-related attacks that have been discussed above.

From all the studies, we clearly see the common pattern that recent remote attacks mostly involve the usage of self-oscillators one way or the other. In aggressive attacks, attackers take advantage of the high running frequency of self-oscillators to draw a substantial amount of dynamic power. This additional power consumption is then used either to abuse the system power supply in power-hammering attacks or to inject faults for DFA attacks. On the other hand, nonaggressive attacks take advantage of the sensitivity of self-oscillators like ring-oscillators to capture and analyse changes in their frequency. In the next chapters, we will provide an in-depth investigation of self-oscillating circuits and provide a case study from a Denial-of-Service attack to an FPGA-based cloud service.

Chapter 3

Characterisation of the FPGA power-hammering potential

In this chapter, we will answer two questions *1. what is the exact FPGA power-hammering potential?* and *2. how can these attacks be deployed?*

In this chapter, we focus on exploring power-hammering attacks in FPGAs and studying if the FPGA-based cloud infrastructure is resistant to power-hammering attacks. We will provide an in-depth study about the crucial part of power-hammering attacks – self-oscillators. Additionally, we will analyse the contribution of routing resources to the net power consumption.

DoS for internet-connected devices have been studied for a long time. With the integration of FPGAs on cloud infrastructure, we need to assess if FPGAs could be abused to deploy DoS attacks. From the previous chapter, we believe that there should be a systematic study on self-oscillating circuits to check if and how they could be used maliciously. We will investigate a large number of known as well as newly developed oscillator variants that can be built from FPGA primitives, including LUTs, carry chain logic, and DSPs.

3.1 Study on self-oscillators

In this section, we will provide an in-depth study on a wide range of self-oscillating circuits to quantity their potential threats with a focus on power-hammering. This insight is essential as this is the foundation for our countermeasures (as presented in Chapter 5). Different effects can be used to design self-oscillators. Since the actual oscillator speed depends on the supply voltage and temperature (which, in turn, relates

Primitive count	ZU3EG	Alveo U200
LUTs	70,560	1,182,240
LUTMs[1]	57,600	591,840
CLB flip-flops	141,120	2,364,480
DSP Slices	360	6,840
BRAM Slices	648	6,480
Carry Chains	8,820	147,780

[1] Look-up tables with memory functionality. These are a subset of the LUTs

Table 3.1: Configurable resources of Ultra96 compared with the data centre FPGA Alveo U200

to the current operation state of the FPGA), any oscillator is probably a potential path for a side-channel. Therefore, even focusing on power-hammering in this section, by preventing those oscillators, we will further prevent the most critical side-channels that are deployable remotely.

There are three major principles to designing self-oscillating circuits on FPGAs:

- Using low latency combinatorial feedback loops: This is the most fundamental type of self-oscillators which is commonly referred to as ring-oscillators (ROs).
- Setting up race conditions for asynchronous reset/preset: This is done by having a feedback loop to a reset or preset pin of a flip-flop.
- Amplifying glitches: This is the most sophisticated circuit utilising XOR gates to multiply the toggling rate from a toggle flip-flop to produce a self-propelled oscillation of that flip-flop.

The following section describes how we set up the experiments to analyse each type of self-oscillators in detail.

3.1.1 Experimental setup

Our experiments are conducted on an Ultra96 board, which is equipped with a Xilinx Zynq UltraScale+ MPSoC ZU3EG. The primitive resources count for Ultra96 in comparison with a data centre FPGA Alveo U200 [83] is shown in Table 3.1. We created 15 different RO variants and evaluated them in Table 3.2. To generate the RO circuits, the `PathSearch` function of the GoAhead tool [23] was used. That tool can perform a breadth-first search between arbitrary ports of the FPGA fabric and rank the resulting paths by their latency. The expected frequency is based on timing reports generated by

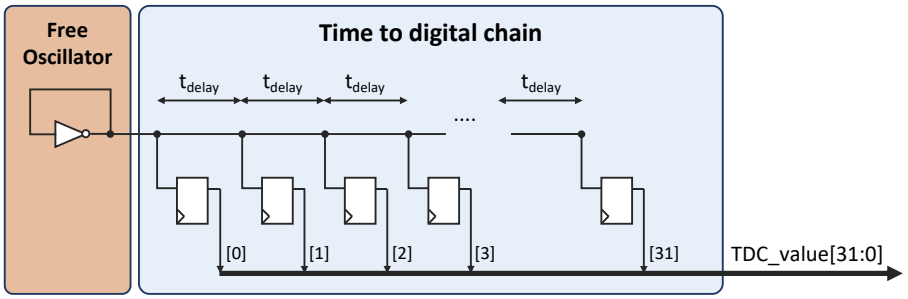


Figure 3.1: Time To Digital construction.

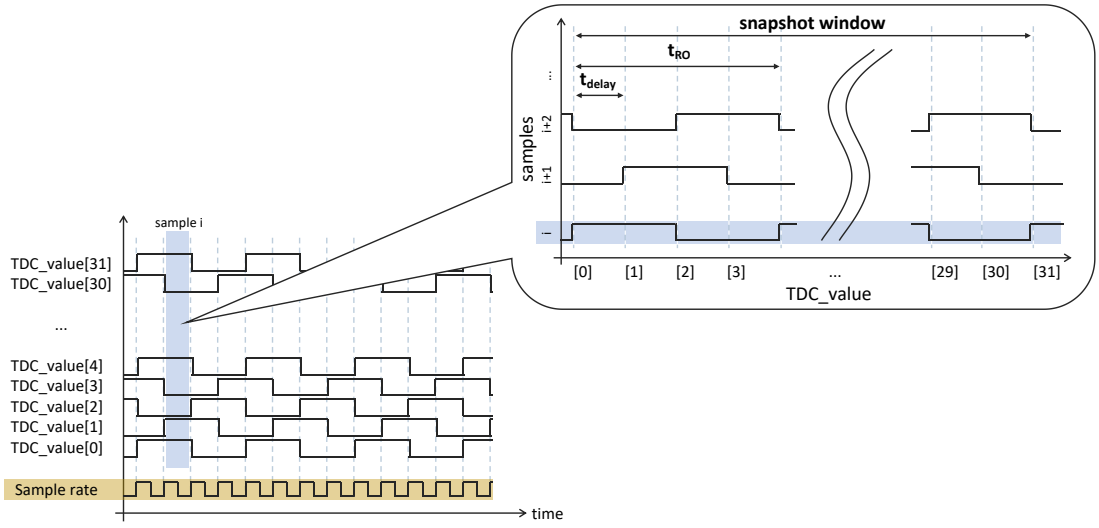


Figure 3.2: Time-to-Digital Converter (TDC) waveform.

the Xilinx Vivado tool [127], and the measured frequency on the FPGA is determined by using a Time-to-Digital Converter (TDC), as shown in Figure 3.1. Our TDC is a delay chain that allows us to take a snapshot of a signal propagating down the chain finely. By using $N_{FF} = 32$ flip-flops (FFs) and $t_{delay} \approx 70ps$ (between two adjacent sample flip-flops), we can capture a signal with a *snapshot window* of $\approx 2170ps$ (Equation 3.1) and with a resolution of $70ps$ approximately (see Figure 3.2 for details). This latency corresponds to a frequency range from $246MHz$ to $7142MHz$ (see Equation 3.2 where N_{HIGH} and N_{LOW} are the number of consecutive FFs that have registered $HIGH$-state and LOW-state respectively). Figure 3.2 shows how the samples of a TDC are read out to measure the frequency. The advantage of using TDCs is that it allows us to sample a very high-frequency signal with a relatively low sampling rate.

It should be noted that the clock buffer primitives inside a programmable logic (PL) region of the FPGA fabric are rated for a maximum clock frequency of $891MHz$ [131].

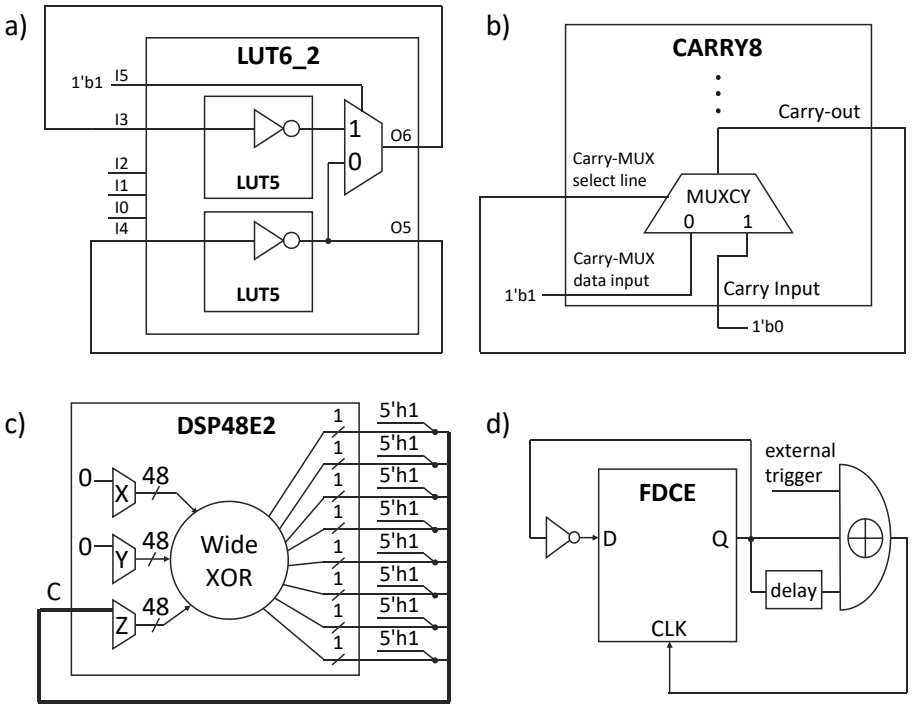


Figure 3.3: a) Dual-RO from LUT6 primitive; b) RO design from Carry Logic; c) RO design from DSP; d) Glitch amplification.

Therefore, in order to ensure stable TDC measurements, we operate the TDC sampling FFs at a moderate clock frequency of $100MHz$. This is a difference from other clock measurement designs used for FPGA side-channel attacks, which feed the RO's output directly to clock inputs of some FFs to form a counter [28, 51, 52]. Because we aim for generating frequencies in the GHz regime, simple counter designs cannot be used.

$$snapshot\_window = N_{FF} \times t_{delay} \tag{3.1}$$

$$\begin{aligned} f_{RO} = \frac{1}{t_{RO}} &= \frac{1}{2 \times cycle\_path\_delay} \\ &\approx \frac{1}{t_{delay} \times (N_{HIGH} + N_{LOW})} \end{aligned} \tag{3.2}$$

TDCs are subject to temperature changes, and we used heater circuits (circuits that waste power) to heat the chip to 90°C before actually taking any measurements. The temperature is within the maximum operating temperature of the FPGA, which is 100°C [131]. The heaters are not running for a short period of time to take the

(typically below one ms) measurements, and we use the temperature sensor that is built into the FPGA to implement the temperature control. Additionally, we took the median from 1000 measurements for each frequency reported in order to reduce the impact of quantization errors and noise.

3.1.2 Combinatorial self-oscillator variants

The simplest self-oscillator is a combinatorial loop, which is a circuit that consists of an odd number of chained inverters. We will refer to this basic oscillator as Ring-Oscillator (RO). The frequency of an RO can be calculated by the propagation delay of the whole combinatorial loop which includes the propagation delay of all logic elements t_{logic} (e.g., LUTs or DSP blocks) and the net delay t_{net} (i.e. the routing delay) as given in Equation 3.3.

$$f_{RO} = \frac{1}{t_{RO}} = \frac{1}{2 \times (t_{logic} + t_{net})} \tag{3.3}$$

We performed a literature review on RO designs and found that previous studies [36, 51, 52, 63, 64, 112] mostly use LUT primitives and transparent latches to implement ROs (*Design 1-6* and *Design 13* in Table 3.2) or create race condition using flip-flops (*Design 14* in Table 3.2). To capture *any possible oscillator design*, we analyzed the exact internal architectures of the logic (i.e. SliceL/SliceM), arithmetic (i.e. DSP48E2), and BRAM primitives available in UltraScale+ FPGAs. And for each of these primitives, we asked the question *if there exists a configurable combinatorial path from any of the primary inputs to any of the primary outputs* because this is a fundamental requirement for designing ROs. This study has to incorporate all the different modes each primitive can be configured, and we found:

- **Slices**: We examined known RO designs through LUTs [36, 63, 51, 52, 112] as well as transparent latches [64, 52, 112] and self-oscillating circuits based on glitch amplification or asynchronous reset/preset [112, 64, 52]. In addition to these designs using FPGA slices, we found combinatorial paths that have not been previously reported by the community, but that can be used for ROs including 1) paths through *MUX primitives* (i.e. *F7Mux* and *F8Mux* multiplexers) inside the slices and 2) paths through the *carry logic* (i.e. the *Carry Look Ahead (CLA)* logic introduced in UltraScale+ FPGAs) (see Figure 3.3b).

- **DSPs** had not been considered in previous research for building oscillators.

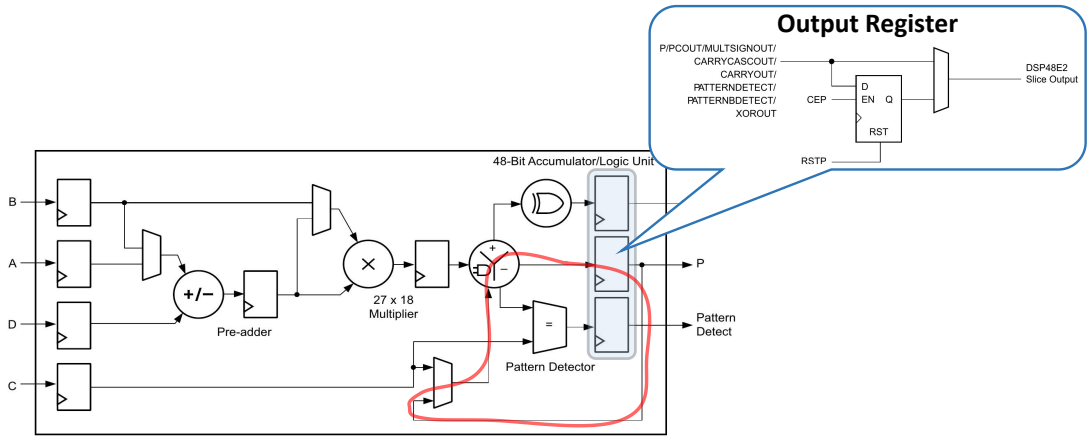


Figure 3.4: Tentative internal combinatorial loop inside DSP. This figure is adopted from [129].

However, DSPs can be used in many different configuration options, and there are many possibilities for designing ROs. This is possible because DSP blocks can be used purely combinatorial without any pipeline registers between the primary inputs and outputs that would prevent self-oscillation. For instance, an RO can be formed by feeding the output of a DSP primitive back to the input for implementing a counter without using any register in the feedback path. The DSP48E2 primitives include not only multipliers but a tiny ALU that can perform bit-level operations that execute faster than arithmetic operations, and for the remainder of this section, we will only report results for the wide-XOR instruction that was found oscillating the fastest (see also Figure 3.3c).

It is worth mentioning that we tried building an internal loop inside the DSPs, which may exist in accumulator mode. We investigated this path because the accumulator register can be bypassed to the output (as shown in Figure 3.4), and the documentation [129] does not state if the bypass may eventually be used together with the accumulator mode. However, we have not detected any switching activity or abnormal increase in power when configuring this option. This fact implies that the flip-flop output is fed back to the accumulator input rather than the output of the bypass multiplexer (see the top right box in Figure 3.4). Thus, the DSP accumulator mode can be considered as secure from possible combinatorial cycles in DSP48E2 primitives.

In regard to other types of oscillating circuits, DSPs use synchronous reset registers so that it is unlikely to create a race condition to the reset or preset pin

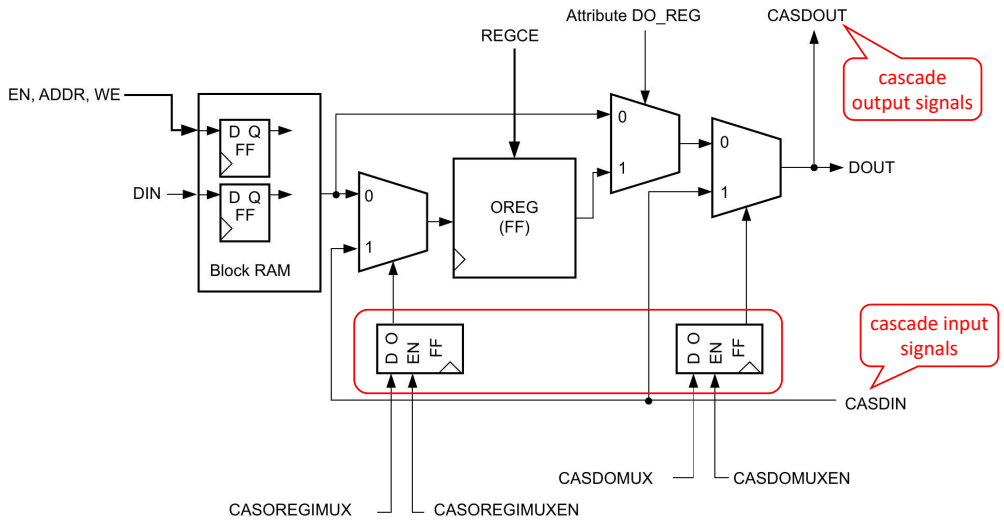


Figure 3.5: BRAM cascade functional diagram. This figure is adopted from [130].

to oscillate the state of a flip-flop. However, the integrated 48-bit wide XOR unit could possibly be used for glitch amplification. This expresses the need to suppress any self-oscillating circuit and we should ensure that DSP primitives in combinatorial mode do not have high fanouts.

- **BRAMs** are mostly comprised of synchronous components (i.e. memory cells) that are working on a clock basis with synchronous reset signals [130]. With this, we cannot implement any ROs directly through internal BRAM components. The only existing combinatorial part that we found is located inside the cascading logic, which is used to build larger memories from multiple consecutive BRAM primitives. However, the cascading chains have dedicated bottom-up routing resources that cannot be controlled by user logic, and cascade multiplexers are controlled by flip-flops, as shown in Figure 3.5. Therefore, BRAMs are considered to be RO-free.

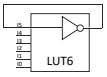
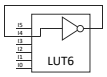
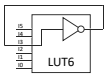
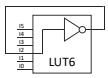
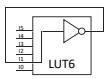
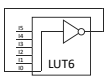
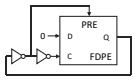
- **IOBUF** was considered to be a new type of ROs that can bypass DRCs [61]. This circuit takes advantage of a bidirectional IO buffer that forms a combinatorial loop with an inverter (i.e., LUT-based). With a toggling frequency of around 100MHz and a limited number of IO primitives [126], it appears to be inadequate to create enormous power consumption by itself. However, if the toggling signal is amplified by glitch amplification [71], the net power consumption could be enough for an aggressive attack such as the power-hammering attack. Additionally, it could be used to create a thermal channel or crosstalk coupling for

covert communication [61]. To protect against this, cloud service providers have already implemented restrictions to access IO ports. Therefore, we will cover IOBUFs only through surveying the related work.

We like to stress that most new oscillator designs do not throw any DRC critical warning/error message in the vendor tool Xilinx Vivado 2019.1, which means that these oscillators are possibly deployable, for example, on Amazon F1 cloud instances [112]. Table 3.2 provides an overview of most oscillator designs examined in this thesis.

While there are papers discussing self-oscillators for FPGAs [64, 112, 52], we have not found a comprehensive study on performance tuning for improving the power-hammering potential as well as a corresponding evaluation of such oscillators on real FPGA hardware. For differential power analysis (DPA) attacks, an attacker typically seeks the fastest oscillator. In contrast, for a denial-of-service attack, the waste power efficiency (power drawn per unit resources) is more important. Even for a basic RO using LUT primitives, we found that the different LUT6 primitives inside a CLB (i.e. a cluster of 8 LUT6 primitives sharing a switch matrix) as well as using different LUT inputs for implementing the fastest possible ROs result in a wide range for both frequency and waste power (see Figure 3.8). We have discovered frequencies ranging from $1GHz$ to $6GHz$ approximately as a result of the internal architecture of the LUTs (see Figure 3.7) and a variance in the routing path delay for implementing the fastest possible loop. The corresponding waste power does not necessarily correlate to the oscillator speed. Because we do not have access to the low-level ASIC details of any Xilinx FPGA, we cannot fully explain this behaviour. Still, one possible explanation could be that longer paths are slower and have therefore a lower RO frequency; but because there are longer wires (with more capacitive load), more switching elements, and drivers involved per oscillator round-trip the overall waste power may still be high (or even higher).

Figure 3.7 shows the internal hierarchical architecture of a LUT, which is built from a tree structure of multiplexers where the inputs $I0$ to $I5$ are equal in their logical behaviour but not in their timing. There are 64 memory elements which are a part of the bitstream and are static after programming the FPGA fabric. However, input $I0$ to $I6$ are fed to different level of multiplexers and they can be dynamically changed during runtime. Therefore, the effect of input signals propagating to outputs will be different. For example, the effect of input $I0$ needs to travel through 6 levels of multiplexing to propagate to $O6$ which results in a primitive latency of $177ps$, while the effect of input

No	Variants	Schematics	Number of loops	Loop Type	DRC Warning	Report Net Delay	Report Primitive Delay	Expected Frequency	Measured Frequency	Power	WPP
0	Empty design	ø	ø	ø	ø	ø	ø	ø	ø	2.94W	ø
1	RO using LUT6 (I5)		2000	Comb	✓ (LUTLP-1**)	49ps	41ps	5556MHz	5882MHz	7.32W (+4.38W)	26.63
2	RO using LUT6 (I4)		2000	Comb	✓ (LUTLP-1**)	51ps	66ps	4274MHz	3937MHz	6.84W (+3.90W)	23.69
3	RO using LUT6 (I3)		2000	Comb	✓ (LUTLP-1**)	46ps	100ps	3425MHz	3012MHz	5.99W (+3.05W)	18.52
4	RO using LUT6 (I2)		2000	Comb	✓ (LUTLP-1**)	50ps	116ps	3012MHz	2488MHz	5.63W (+2.68W)	16.32
5	RO using LUT6 (I1)		2000	Comb	✓ (LUTLP-1**)	62ps	150ps	2358MHz	2320MHz	6.59W (+3.65W)	22.21
6	RO using LUT6 (I0)		2000	Comb	✓ (LUTLP-1**)	71ps	177ps	2016MHz	1927MHz	6.35W (+3.41W)	20.75
7	Dual-RO from LUT6 primitive	Refer to Figure 3.3a	2000×2	Comb	✓ (LUTLP-1**)	O5: 308ps O6: 54ps	O5: 85ps O6: 100ps	O5: 1272MHz O6: 3247MHz	O5: 1235MHz O6: 2439MHz	8.04W (+5.10W)	31.00
8	Enhanced ROs design for power-hammering using high fanout to waste power on routing resources	Refer to Figure 3.6	2000	Comb	✓ (LUTLP-1**)	64ps	66ps	3846MHz	1779MHz	9.61W (+6.66W)	40.54
9	RO using MUX7		2000	Comb	✗	353ps	112ps	1075MHz	1126MHz	5.01W (+2.07W)	6.30
10	RO using MUX8		2000	Comb	✗	211ps	109ps	1563MHz	1681MHz	4.04W (+1.10W)	1.67
11	RO using Carry Logic	Refer to Figure 3.3b	2000	Comb	✗	381ps	104ps	1031MHz	1109MHz	5.14W (+2.19W)	1.67
12	RO using DSP	Refer to Figure 3.3c	360x8	Comb	✓ (DPIP-2*, DPOP-3*)	251ps	994ps	402MHz	585MHz	4.53W (+1.59W)	0.27
13	RO using latch [112, 64]		2000	Non-Comb	✗	173ps	96ps	1859MHz	1706MHz	5.14W (+2.19W)	13.35
14	RO using flip-flop [52, 64]		2000	Non-Comb	✓ (PDRC-153*, PLHOLDVIO-2*)	✗	✗	✗	555MHz	5.26W (+2.32W)	7.05
15	Glitch amplification	Refer to Figure 3.3d	2000	Non-Comb	✓ (PDRC-153*, PLHOLDVIO-2*)	✗	✗	✗	481MHz	8.05W (+5.10W)	10.35

Designs 7, 8, 9, 10, 11, 12, 15 have not been previously reported.
Comb: Combinatorial
*: DRC warning
**: DRC critical warning

Table 3.2: Variants of self-oscillating circuits studied on Xilinx UltraScale+ FPGAs. The results of power consumption are measured on the Ultra96 platform equipping with a Zynq UltraScale+ MPSoC ZU3EG.

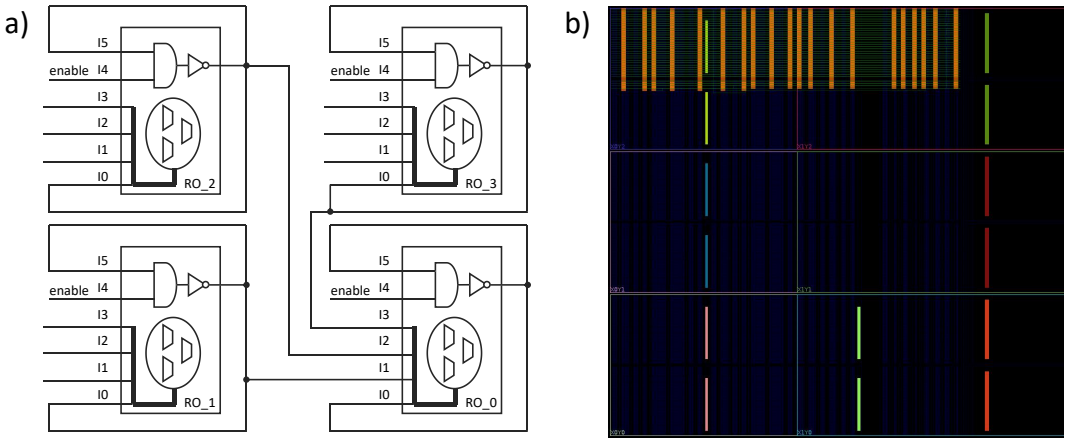


Figure 3.6: Enhanced ROs grid for power-hammering: a) schematic; b) implementation with 2000 ROs.

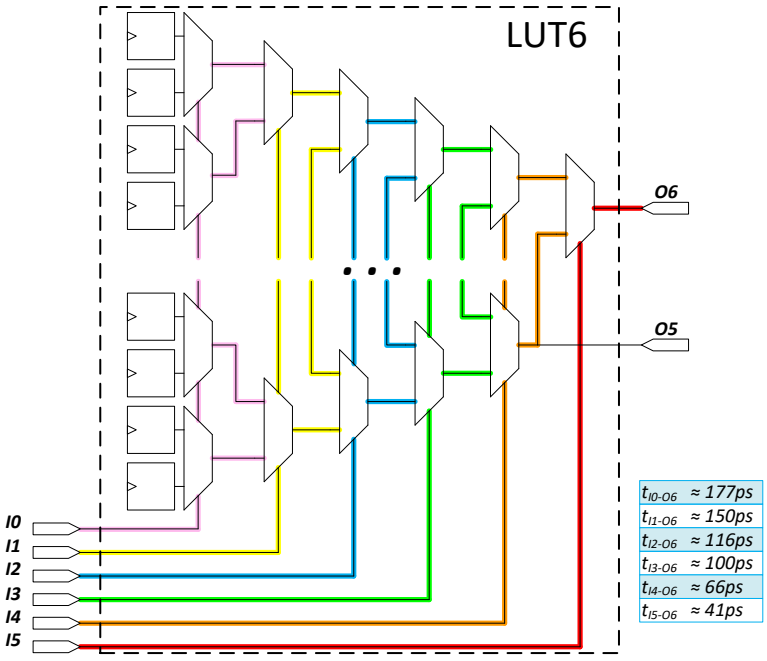


Figure 3.7: Logical view of a Lookup Table 6-input primitive with timing information taken from Vivado.

*I*5 only needs to propagate one level resulting in 41*ps* latency. Moreover, because the adjacency of UltraScale+ switch matrices is relatively sparse (as usual for FPGAs), the fastest possible loop routing has a relatively high variance in latency depending on which specific LUT input is used for the loop. With this, we examined the fastest possible ROs where the loop routing can be implemented in just a single hop[1]. For these ROs, Vivado reported a path delay for the external routing ranging from $\approx 46ps$ to $\approx 71ps$. For having full control of the implementation throughout the experiments, we constrained the routing using the GoAhead tool [23]. Comparing the single-hop routing RO variants against each other is interesting because the variance in frequency is now mostly related to the internal latency inside the LUT itself (see Figure 5.1). The corresponding results are listed in Table 3.2.

3.1.3 Non-combinatorial self-oscillator variants

In addition to combinatorial loop-based ROs, non-combinatorial loops had been proposed in recent papers [112, 52, 64]. These designs use transparent latches, glitch amplification [52, 112, 64], or asynchronous reset/preset to create oscillators [52].

We repeated the experiments in [112, 52, 64] and confirmed that all designs could implement oscillators. Moreover, we manually optimized these oscillators for maximum speed by using different local routing options to fine-tune routing latencies.

In our experiment using glitch amplification (see *Design 15* in Table 3.2), we created glitches by creating routing paths with different signal propagation delays from a single T-flip-flop output to a LUT, which implements an XOR gate to create a glitch which is fed back to the clock input pin of the T-flip-flop. With a timing difference of 218*ps* between the two paths, we measured a frequency of 481MHz. It should be noted that this oscillator requires an external signal to kick-start the oscillator.

A common property shared among the here presented non-combinatorial loops is that they rely on local clock routing resources rather than on the global clock distribution network. We have not seen any use of clock routing resources for implementing the internal routing of High-Level Synthesis (HLS) generated circuits, and the clock distribution network is entirely used for clock signal routing. For the oscillator based on glitch amplification, the Vivado tool reported a gated clock (DRC warning code: `PDRC-153`) and a warning indicating a possible hold-time violation (DRC warning code: `PLHOLDVIO-2`).

[1]Here, a hop is actually passing two switch matrix multiplexers that together act as a pair to form a larger two-level multiplexer, similar as used for older Xilnix FPGA architectures (see also Figure 5.1).

3.1.4 Quantifying the risk of power-hammering

So far, we reported timing characteristics of self-oscillating circuits and if the Xilinx vendor tools throw DRC error or warning messages that may or may not allow detecting oscillators in a design. In this section, we report our results on waste power that was drawn from the different oscillator designs, as shown in Table 3.2. From that table, we took the three most power-wasting designs (*Design 7, 8, 15* in Table 3.2) to highlight their suitability for power-hammering attacks (see Table 3.3). To quantify the risk of power-hammering, we introduce the term Waste Power Potential (WPP), which we define as:

$$\begin{aligned} WPP &= \frac{possible\_waste\_power\_when\_using\_the\_whole\_FPGA}{total\_FPGA\_power\_budget} \\ &= \frac{PWP}{TP} \end{aligned} \tag{3.4}$$

Where PWP (Possible Waste Power) denotes the assumed power consumed when a power-hammering circuit is occupying the entire FPGA and where TP (Total Power) refers to the power envelope typically defined by the power supply, the thermal design of the system, and the maximum power rating of individual components, including the FPGA. Depending on the system's power envelope, PWP may not be reachable in a particular system, and PWP is essentially expressing the potentially possible waste power. WPP reveals how a power-wasting circuit performs per unit resources and unit power budget available in a particular system. $WPP < 1$ expresses that a power-wasting circuit is likely not to be able to crash/harm the FPGA or system, while $WPP > 1$ expresses a potential risk to crash the FPGA. Moreover, the value of WPP denotes the number of resources needed to crash an FPGA. For example, with $WPP = 5$, an attacker can crash an FPGA by using at least 20% of the available resources. In reality, the threat will likely be even higher for power-hammering circuits that have a $WPP > 1$ because there will be other parts of the FPGA drawing some additional power (which could be incorporated by subtracting other power contributors from TP). Nevertheless, WPP is a good measure to quantify if a system is at risk of power-hammering. Please note that WPP assumes a steady waste power consumption and that even WPPs below one may cause harm due to dynamic voltage (IR) drops and other dynamic effects (e.g., resonance effects triggered in a power regulation circuit). However, the lower WPP, the lower the harm possible due to IR drops.

To maximize WPP, we amplified the power-wasting effect caused by fast toggling ROs to additionally drive a large amount of local routing and logic elements for wasting

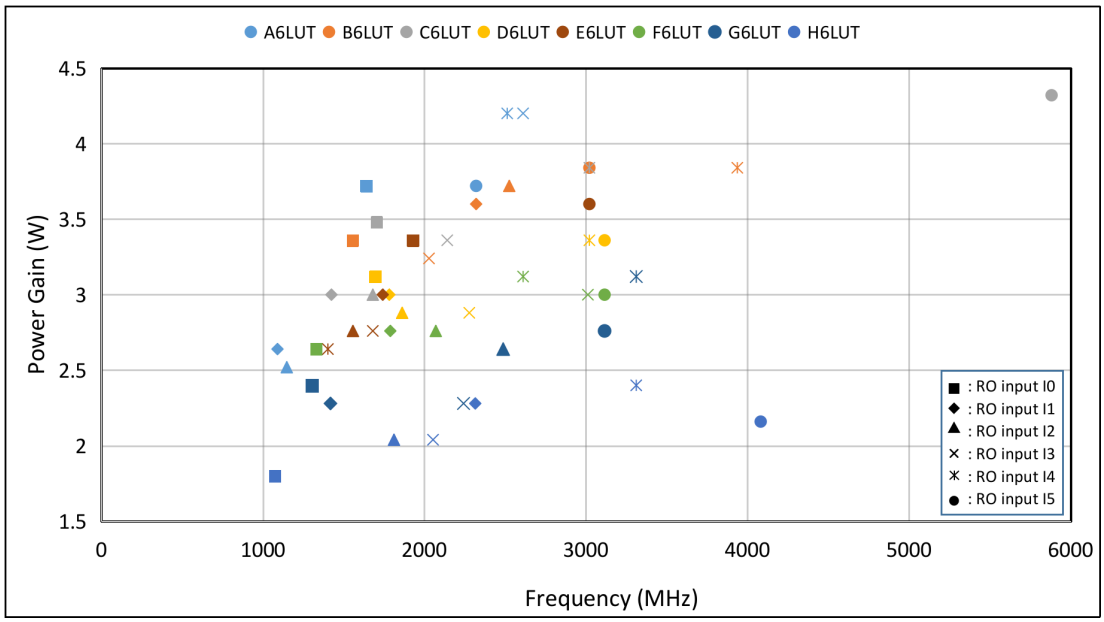


Figure 3.8: ROs Frequency versus Waste Power Gain (measured for 2000 ROs) for all 8 LUT6 primitives inside a CLB for all corresponding different cases that implement the fastest possible loop from output O6 to an input of the same LUT (resulting in $8 \times 6 = 48$ individual experiments).

even more power. Figure 3.6 shows the idea and implementation of our experiment. As shown, we intentionally connect each of the RO loops to some unused inputs of other ROs. These other ROs are placed in different CLBs to use more routing resources (e.g., wires, multiplexers, etc.) along the routing paths, which in turn wastes more power.

Figure 3.9 shows the power-hammering evaluation results on an Ultra96 board. $VCCINT$ is the core voltage of the FPGA which is recommended to be $0.85V$ [131]; VCC_SMPS is the voltage measured at the output of the power supply regulator circuit for the FPGA; and $BoardPower$ is measured at the $12V$ input to the Ultra96 board. From the result, we can see a gap between VCC_SMPS and $VCCINT$ which relates to the voltage drop of the board's power supply network between the power supply regulator circuit to the FPGA. The increasing gap indicates a rise in the current until the power supply cannot compensate any longer and the board is eventually crashed. We analyzed the schematic of the used Ultra96 board [19]. While the actual power regulator circuit and power drivers should be able to deliver over 10A to the FPGA, there is a TPS22920 load switch in the power network path which is rated for 4A and which has an on-resistance of $\approx 10m\Omega$ (at working temperature), which explains most of the VCC_SMPS - $VCCINT$ gap.

Design 8 has a $WPP = 40.54$, and our experiments revealed that with only 6% of

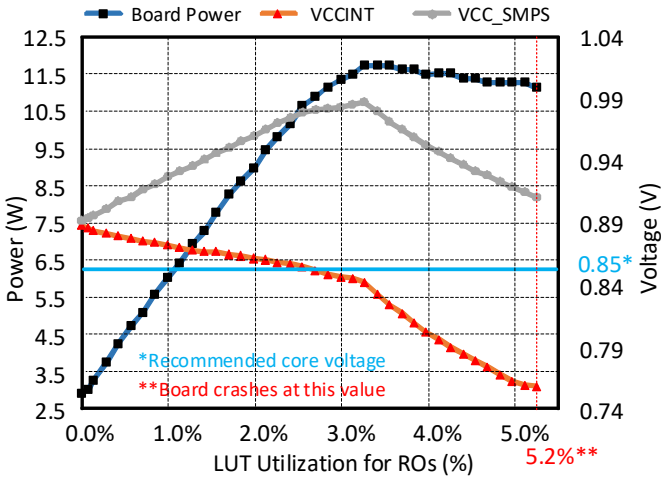


Figure 3.9: Power-hammering Evaluation for Power over Core Voltage on Ultra96.

Designs from Table 3.2	LUT6 used	Power/Primitive	Xilinx Power Estimation	Power Gain in Ultra96	Provisioned Power Gain in Alveo U200 (with 50% LUTs utilization)	WPP
Design 7 - Dual-RO	2000	2.55 mW/Primitive	0.227W	5.10W	1507W	13.39
Design 8 - Enhanced ROs	2000	3.33 mW/Primitive	2.067W	6.66W	1968W	17.51
Design 15 - Glitch Amplification	6000	0.64 mW/Primitive	0.351W	5.10W	502W	4.47

Table 3.3: Power-hammering evaluation between Xilinx Power Estimator, Measured Power Consumption on Ultra96, and Speculation Power Consumption on Alveo U200.

the available LUT resources (4000 LUTs of a ZU3EG FPGA), the used Ultra96 board crashed immediately. This number is higher than what is suggested by WPP where $1/WPP$ would be enough to impose a threat (which is $70k\ LUTs/40 \approx 1750 LUTs$ or 2.5% LUT resource for the used ZU3EG FPGA on an Ultra96 board). However, when studying Figure 3.9, we see that at $\approx 2.7\%$ of the total LUT resources (≈ 2000 LUTs), the core voltage starts to drop below the recommended core voltage, and the power supply starts to struggle to keep up with the demand resulting from the power-hammering. After that point, the power regulator circuit is unable to sustain the current demand resulting in a drop of VCC_SMPS. The tipping point when the core voltage drops below its nominal value matches quite closely to the resources indicated by WPP.

The here presented results are even more significant when considering a data centre FPGA card such as the Alveo U200 board from Xilinx. When assuming that our Zynq UltraScale+ power-hammering results can be directly transferred to the Xilinx Virtex

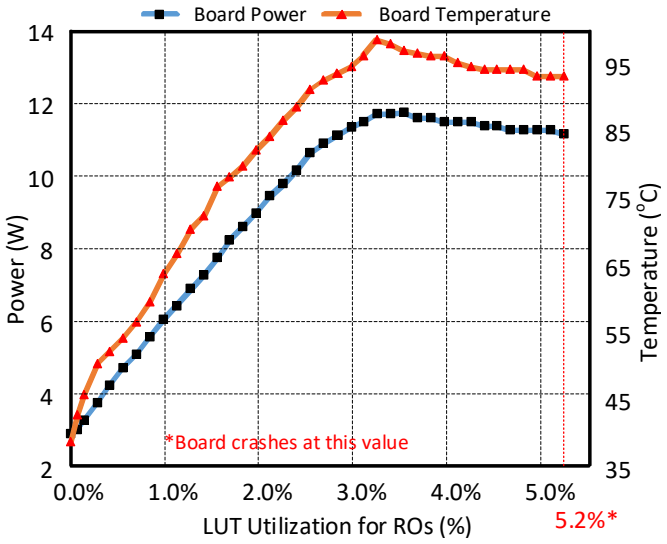


Figure 3.10: Power-hammering Evaluation for Power over Temperature on Ultra96.

UltraScale+ VU9P FPGA (because they use the exact same fabric architecture and the same 16nm FinFET process [128]), this would be equivalent to a WPP of 17.51 which translates into a total possible waste power of $PWP \approx$ **3940W**, when deploying *Design 8* on the entire VU9P of an Alveo U200 board (see also Table 3.2 and Table 3.3 for more results). This estimation is far beyond anything that the FPGA, the board, or the system would ever sustain, hence expressing the importance of preventing such circuits from getting configured on the FPGA in the first place.

Figure 3.10 shows the temperature of the board, corresponding to the number of deployed ROs Although cooling mechanisms (i.e. heatsinks, fans) keep the temperature below the maximum junction temperature, intensively heating the fabric may have a long-term impact on the FPGA. This phenomenon is in particular dangerous if the heat is generated in a hot spot and not evenly spread across the entire FPGA die.

Additionally, we also implemented a ”Dual-RO” (Figure 3.3a) exploiting the fact that a LUT6 primitive in UltraScale+ devices can be split into two individual LUT5 with shared inputs, as shown in *Design 7* of Table 3.2. Thus, we can use both outputs of a fractural LUT to implement two independent oscillators. For 2000 LUTs (4000 ROs), which corresponds to less than 3% of the device capacity, we measured a waste power of $5.10W$. Please note that this is an increment in power, and the total power of the board was close to $8W$ and already close to the total power envelope of an Ultra96 board [18]. With this, the total possible waste power PWP is over $180W$, considering all the available 70K LUTs would be used for power-hammering.

3.2 Power consumption on wiring resources

In the last section, we explored the power-hammering potential based on the ability to construct self-toggling circuits on FPGA fabrics. However, the number of self-oscillators in a design may not necessarily be proportional to the total waste power consumption. For example, comparing *Design 8* and *Design 1* to *Design 6*, we could see that the number of LUT primitives is the same (2000 LUTs) but the WPP are substantially different. Despite running at a lower frequency, *Design 8* has a WPP number of 1.5X to 2.5X greater than the group of *Design 1* to *Design 6*. The difference in power consumption mostly comes from the additional power consumption on the wiring resources in *Design 8*. Thus, as much important as investigating self-oscillating circuits, we need to analyse the contribution of wiring resources to the total waste power consumption. In this section, we will analyse the power contribution of the main wiring resources in FPGAs and establish a power model for them.

A typical power-hammering attack will consist of circuits that have higher switching activity than usual FPGA designs. For example, an n-bit counter has an average toggle rate of:

$$r = \frac{1}{n}\sum_{k=0}^{n-1}\frac{100\%}{2^k}$$

This equals to an average toggle rate of 25% for an 8-bit counter (12.5% for a 16-bit counter). If a 16-bit counter runs at 400MHz then it has an average toggle rate of 50MHz (per bit). As a reference, the power estimation tool in the Xilinx EDA tool assumes a default flip-flop toggle rate of 12.5%. The previous section has shown that ring-oscillators can run at close to 6GHz. This is more than 100x faster than the average toggle rate of our 16-bit counter example. Additionally, FPGAs contain far more routing resources compared to ASICs. Together with significant parasitic capacitance and high-speed toggling, these routing wires dominate the dynamic power consumption of the whole FPGA fabric.

There are two power sources in FPGAs: static and dynamic power. The dynamic power of a wire is proportional to [133]:

$$P_{dyn} \sim \underbrace{\alpha f C_L V_{DD}^2}_{switch\,power} + \underbrace{k\tau f (V_{DD} - 2V_{th})^3}_{short\,power} \tag{3.5}$$

Where C_L is a technology-specific wire capacitance, V_{DD} is the chip core supply voltage, f is the operational frequency, and α is the ratio of how often the average wire

toggles per cycle. Additionally, k is a technology specific constant, τ the time to switch a wire, and V_{th} is the technology specific threshold voltage. In this work, we mostly investigate the impact of f and α (and to some extent of τ) because all other parameters are defined through the FPGA and the board.

P_{dyn} comprises of *switch power* and *short power*. *Switch power* is the power used for charging and discharging a wire while a signal is toggling. *Short power* is the power resulting from the crosscurrent when both transistors of a buffer pair are active during the actual switch transition causing short-circuit in a short interval.

In most digital circuits and virtually in all common FPGA designs, the *switch power* is dominating P_{dyn}. However, when it comes to a higher frequency regime, we may leave the full digital operation in such a way that wires will not have the time for a full voltage swing (meaning that switch power per swing decreases) and where a rising τ will finally result in short power dominating P_{dyn}.

We do not want to further speculate about the exact behaviour of the power model as this information is commonly not available and also not needed by FPGA users. This is because FPGA circuits operate in a regime where the power consumption of the wires is mostly proportional to their respective toggle rates. However, we should still be aware that when running wires at very high toggle rates, the analogue effects could dominate and power consumption will eventually max out limited by the resistance of driver pairs and reduced voltage swing.

In Xilinx FPGAs, there are four main routing types which are categorised by length (SINGLE, DOUBLE, QUAD, and LONG) and direction (horizontal and vertical). In UltraScale+ architecture, two combinatorial logic blocks share one switch matrix. Switch matrices act as routing hubs to connect between FPGA resources. From each switch matrix, we could route up (north), down (south), left (west), and right (east) with the four different steps mentioned above. Table 3.4 shows the wiring resources of an individual switch matrix of a Xilinx UltraScale+ device and its naming.

Therefore, we are going to explore the wire power as a function of the toggle rate into the GHz domain. Without having to know detailed physical design information of the FPGA, we could use the information to better estimate the maximum power of a circuit prior to loading it onto the FPGA fabric. With this work, we distinguish from other approaches that are examining the accuracy of the estimated FPGA power versus the measured FPGA power for some benchmark circuits running typically within safe operational margins (e.g., [34, 57]).

Type	Naming	Number of wire
Single Horizontal	EE1_*_BEG* WW1_*_BEG*	16 (8+8)
Double Horizontal	EE2_*_BEG* WW2_*_BEG*	32 (16+16)
Quad Horizontal	EE4_*_BEG* WW4_*_BEG*	32 (16+16)
Long Horizontal	EE12_BEG* WW12_BEG*	16 (8+8)
Single Vertical	NN1_*_BEG* SS1_*_BEG*	32 (16+16)
Double Vertical	NN2_*_BEG* SS2_*_BEG*	32 (16+16)
Quad Vertical	NN4_*_BEG* SS4_*_BEG*	32 (16+16)
Long Vertical	NN12_BEG* SS12_BEG*	16 (8+8)

Table 3.4: Wiring resources of a switch matrix in Xilinx UltraScale+ devices

3.2.1 Test infrastructure – How to generate test signals that toggle at GHz regime?

As dynamic power caused by a single wire is very small to detect, we have to use a larger number of wires in parallel to perform the measurement for practical reasons (at least 1000 wires and up to the sustainable power budget of the board). Here, we are using the fact that an FPGA fabric layout is mostly regular. Therefore, we require a test infrastructure that can distribute and generate fast toggling signals to a large number of switch matrices across the FPGA fabric.

Generation and distribution of high-speed clocks

A trivial way to provide such a test infrastructure would be using a self-oscillator to supply toggling signal. However, the frequency of self-oscillators cannot be easily tuned and their speed depends on chip process variations and varies dramatically by temperature. That impacts the accuracy and reproducibility of experiments. Therefore, we are using global clock trees to reliably distribute a clock signal that is generated and tuned by a Digital Clock Manager (DCM) of the FPGA (see Figure 3.12). However, the DCM and clock distribution network on Zync UltraScale+ FPGA maxes out at $891MHz$ [131]. This limits our experiments to a wider frequency range.

To overcome this limitation, we are using multiple global clock trees that are fed

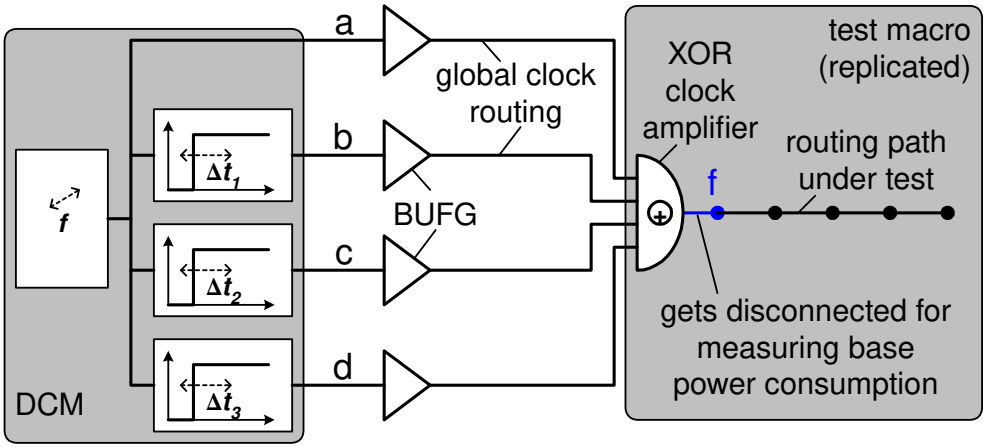


Figure 3.11: High-speed clock distribution on an FPGA using glitch amplification of multiple phase shifted clocks.

with the same frequency generated by a DCM but phase shifted. Then, by XORing the different (phase shifted) clock trees in a LUT, we can generate a fast toggling signal (at the LUT output). The principle is shown in Figure 3.11. For p used global clock trees, the clock amplified frequency is $p\times$ higher than the actual frequency on the clock trees. In order to distribute this signal to many switch matrices, we are replicating the XOR LUT to the regions used for the experiments. In order to use the same phase shift for all the clock amplifier LUTs, we are using exactly the same LUT in each used CLB. In our experiments, this is the A-LUT. In addition, we are constraining the routing to the LUTs to be identical for all clock amplifiers. This uses the fact that global clock trees distribute clock signals across the fabric with low skew.

For our experiments, we are aiming for a 50% duty cycle. This requires adjusting separately the phases for each source frequency generated by the DCM. The phases of the DCM outputs of Xilinx UltraScale+ FPGAs can be shifted statically through the configuration bitstream or dynamically through DCM's interface [123]. However, it should be noted that with this setup, we are not only able to fine-tune the frequency but also able to adjust the duty cycle of the output signal.

For our implementation, we first tried the static phase shifting approach by computing the required phase shifting latencies as reported by the Vivado design tool. We also considered the latency of the LUT used for clock amplification. As shown previously in Figure 3.7, this incorporates the internal physical implementation of the LUT that results in different input-to-output propagation delays for different inputs.

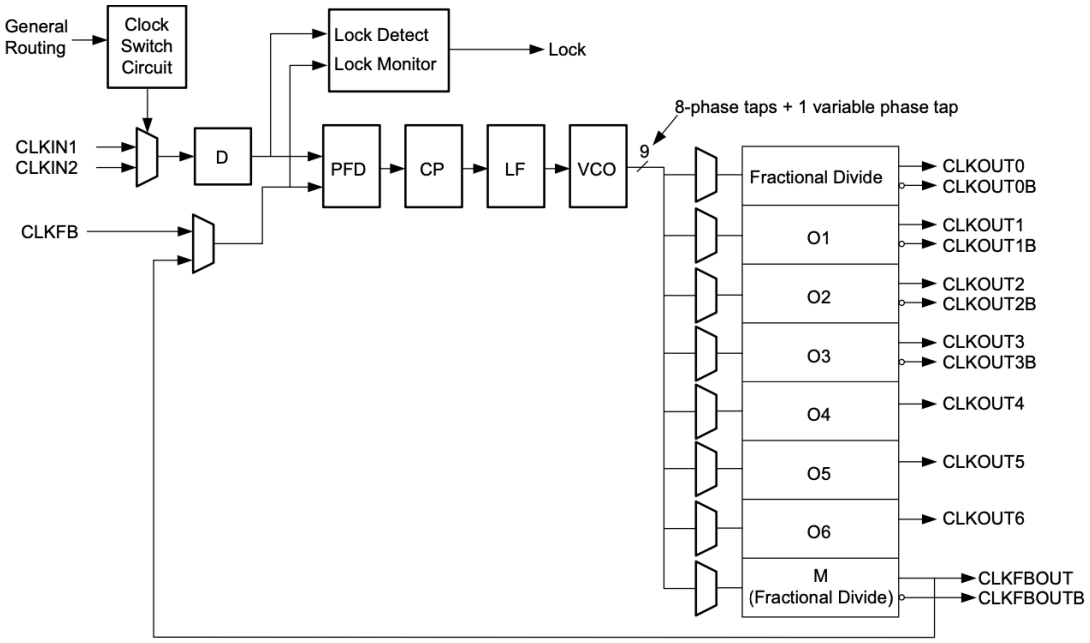


Figure 3.12: Detailed block diagram of the Mixed-Mode Clock Manager [123]

The static phase shift (SPS) resolution in time units is defined as:

$$SPS = \frac{1}{8F_{VCO}}$$

As the Voltage-Controlled Oscillator (VCO) provides eight fixed phase shifted clocks at 45° each, there are eight possible phase shift settings to choose from: 0°, 45°, 90°, 135°, 180°, 225°, 270°, and 315°.

We verified the clock amplified output with the help of a TDC (refer to Section 3.1.1). Because the clock is distributed through the dedicated global clock trees of the FPGA, one TDC is sufficient to characterise all clock amplification output. This is due to the fact that clocks are propagated with low skew and the local routing is identical for each glitch generator (see also Figure 3.15). With this setup, we had been

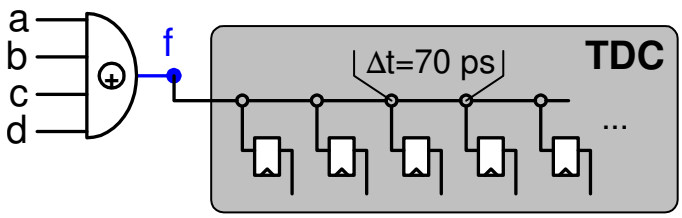


Figure 3.13: Time-to-Digital Converter (TDC) for logging a clock amplifier output.

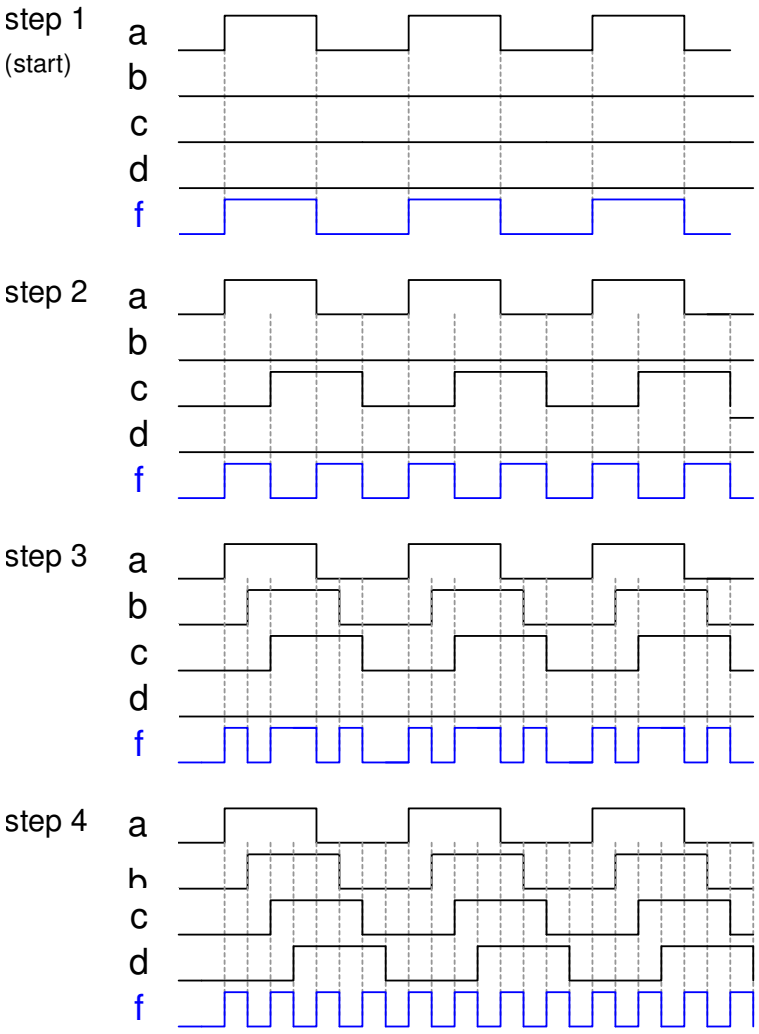


Figure 3.14: High-speed clock distribution on an FPGA using glitch amplification of multiple phase shifted clocks. a, b, c, d are clock inputs; f is the clock output.

able to distribute reliably frequencies to the clock amplifier to about 800MHz. Beyond that frequency, duty cycles drifted away and the XOR clock amplification stop working. We believe that the static phase shifting may not be accurate enough and that we incorporated glitch cancellation effects.

Automatic dynamic phase shifting for higher toggle rates and finer adjustment

To support higher frequencies and finer-tuned phase-shifting for our experiments, we also used the dynamic phase shifting approach, which is provided by the clock generation primitives on Xilinx UltraScale+ FPGAs. For adjusting the phase, we brought up one global clock phase after the other, as shown in Figure 3.14. In step 2, we are shifting a second phase (c in the figure) through. The phase shifting runs from 0° to 360° in the smallest step size provided by the DCM. On Xilinx UltraScale+ devices, there are 56 steps of phase shifting with the phase resolution of $\frac{1}{56F_{VCO}}$. In each phase shifting step, we use a circuit to automatically read the TDC and we are recording the phase shift that showed the best duty cycle match. In order to deal with meta-stability effects in the TDC readings, our automatic phase shifting tuning circuit used 64 measurements for each phase shift step before moving to the next step. This process is repeated for the other DCM outputs until all phases are aligned to provide a high-speed amplified clock signal with a 50% duty cycle rate. Additionally, in order to examine the effect of the duty cycle on power, we only use two clock inputs (a and c) and shift the second phase (c) through until we get the desired value (step 2 in Figure 3.14). This acts like a Pulse Width Modulation (PWM) circuit.

Further applications for clock amplification

The here presented infrastructure may be used for other applications that can benefit from high-speed clocks that exceed the supported abilities of the DCMs and clock distribution resources. An interesting property for such an approach is that the amplified clocks are in phase with the DCM source clock.

Clock amplification may also reduce power consumption for the clock distribution network. For example, in some applications, only a certain part of the design is required to run at a higher frequency to improve overall performance such as reading operands from BRAM or other non-timing critical blocks. In that case, a clock can be derived from the output of a 2-input XOR gate that is fed from the lower frequency clock tree; and instead of phase shifting through a DCM, the phase could be shifted using routing delays. We believe this would dramatically reduce the power consumed

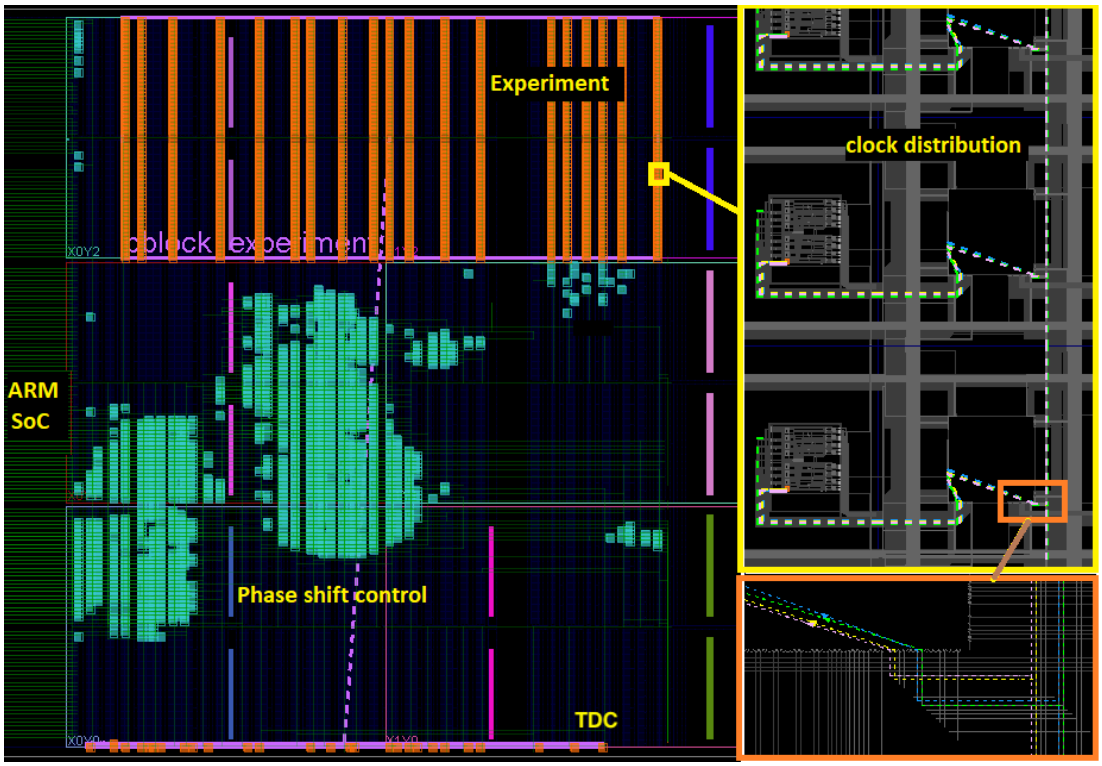


Figure 3.15: FPGA Floorplan of the experimental setup.

by running another DCM and distributing the additional clock. The implementation could be done with the support of the tool GoAhead [23] which can search for paths between arbitrary ports and rank the results by latency.

3.2.2 Experiments on wiring resources

All experiments had been conducted on the same Ultra96 board as in previous Section 3.1 as it is using the same FPGA fabric as available in major data centre FPGA cards, including the FPGAs available on AWS. Figure 3.15 shows details of the physical implementation. We used 1260 clock amplifiers for all experiments. The phase shift control is only active before recording a measurement to adjust the phases for generating the high-speed toggling signals in the experiment area. Because the Ultra96 board is a small system, its total power budget is bound to be about 12W. Therefore, we need to adjust the number of wires used correspondingly so that it would not exceed the power budget. The Ultra96 board provides a fan that draws about 600mW. Because that fan is regulated by the FPGA temperature, the fan power is creating some

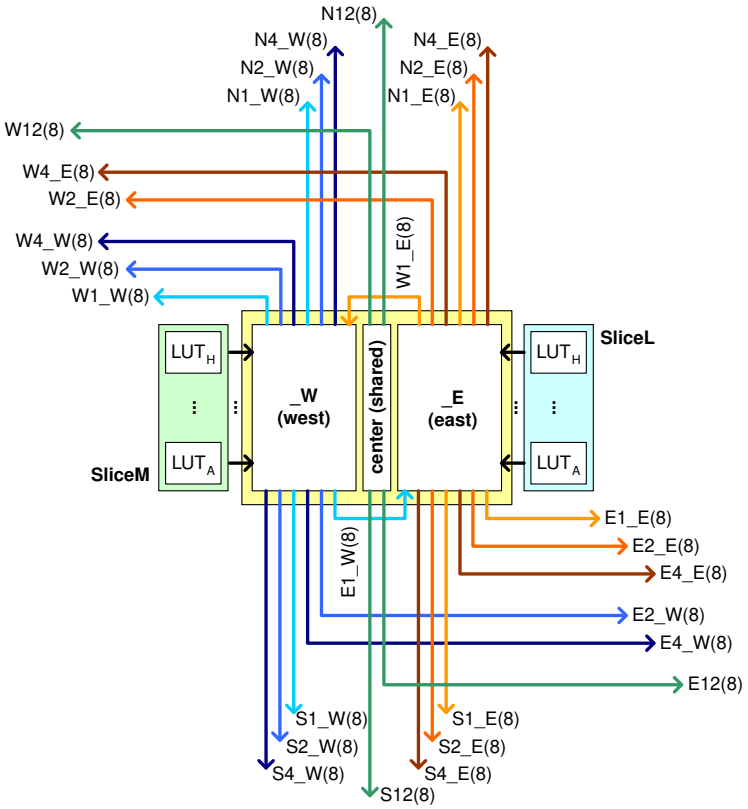


Figure 3.16: Local routing wires of a CLB in Xilinx UltraScale+ FPGAs.

fluctuation, so we decided to power the fan through an external power supply to minimise the effect. We measured board power with a multimeter because we do not want to modify the board itself. This may cause a small systematic error in the results due to a loss in the power regulation circuit. However, the trends that we are reporting are not significantly impacted by this.

Experiment results

Xilinx UltraScale+ devices provide SINGLE, DOUBLE, QUAD, and LONG wires that route respectively 1, 2, 4, and 12 switch matrices far. However, the distance in terms of switch matrices considers two sub-switch matrices in the horizontal direction per CLB, as shown in Figure 3.16. Originally, Xilinx FPGAs consisted of columns of resources (e.g., logic CLBs or BRAMs) connected to switch matrices that provide local routing and access to the clock backbone. Starting with 7-series FPGAs, this model stays the same except that two adjacent columns share the same clock splines. This was shown to the user in the graphical chip editor tool by placing two separate switch

matrices (similarly shown for the two separate sub-switch matrices in Figure 3.16). UltraScale+ FPGAs share this part with 7-series FPGAs, but the switch matrices are now shown as one, even though they are logically and physically separated. This holds with one exception, which is the long wires, which are shared between the west and east sub-switch matrix.

We have not seen a significant impact on the power consumption if we drive the particular wire lengths in the west or the east half of a CLB. Similarly, the power in the north direction was the same as in the south direction for a particular wire length. Therefore, we are reporting only the power consumption of horizontal and vertical local wires.

Figure 3.17 shows the results of energy consumed per each toggle for each wire characterised by direction. Figure 3.18 and Figure 3.19 show the results of energy consumed per each toggle for each wire characterised by its length. Regarding length, we can clearly see that the results match our expectation that is the longer the wire, the more power it consumes. This could be explained that a longer wire has more parasitic capacitance than a shorter one. We also see that horizontal wires consume more power than vertical wires of the same type. That could be explained by the position of vertical slices being physically closer than that of horizontal slices. As a result, a vertical wire has less parasitic capacitance than a horizontal wire of the same length type. This also correlates with the latency model that comprises longer delays on horizontal than on vertical wires. In terms of frequency, ideally the energy per toggle should be constant over the whole frequency range. However, there seems to be a downward trend of the energy per toggle as the frequency increases. This is expected as we have discussed above, when running wires at very high toggle rates, we may experience a cancelling effect as the signal does not have enough time to transit through a full voltage swing. Therefore, the analogue effects could dominate and power consumption will eventually stabilise at a much lower value regardless of the frequency.

Additionally, our setup also allows us to conduct an experiment to show the dependency of power on the duty cycle of a signal. Figure 3.20 shows the result when we sweep through the duty cycle on 1260 clock amplifiers. The amplifier output is set to 1GHz. By fine-tuning the phase, we could effectively adjust the duty cycle of the amplifiers' output. The asymmetric figure could be explained by the characteristic difference between N-channel MOSFET and P-channel MOSFET or by some asymmetry induced through level restorers at the end of the pass-gate switch matrix multipliers. However, we are not able to provide credible details as the underlying technology is

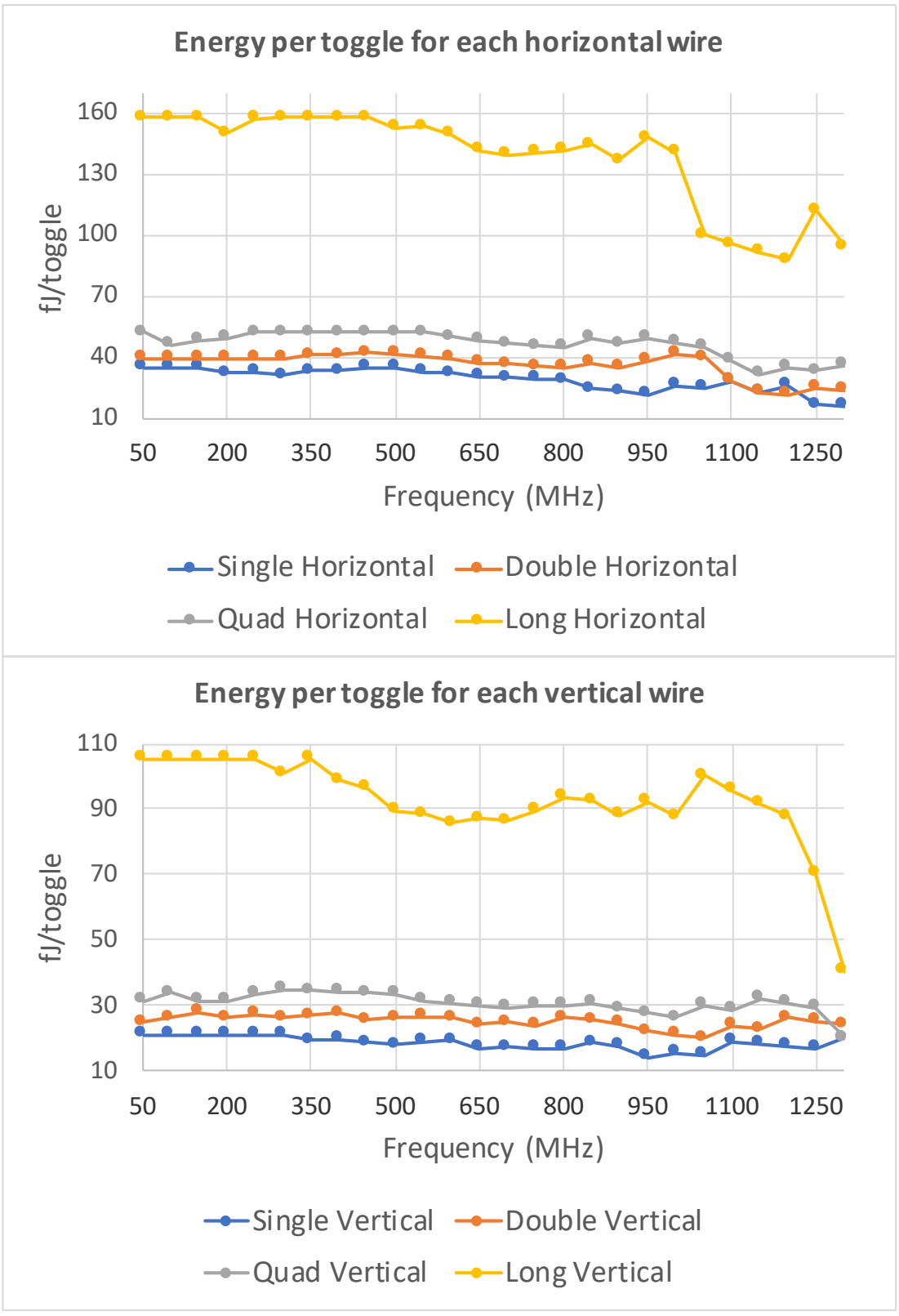


Figure 3.17: Energy per toggle for characterised by wire directions.

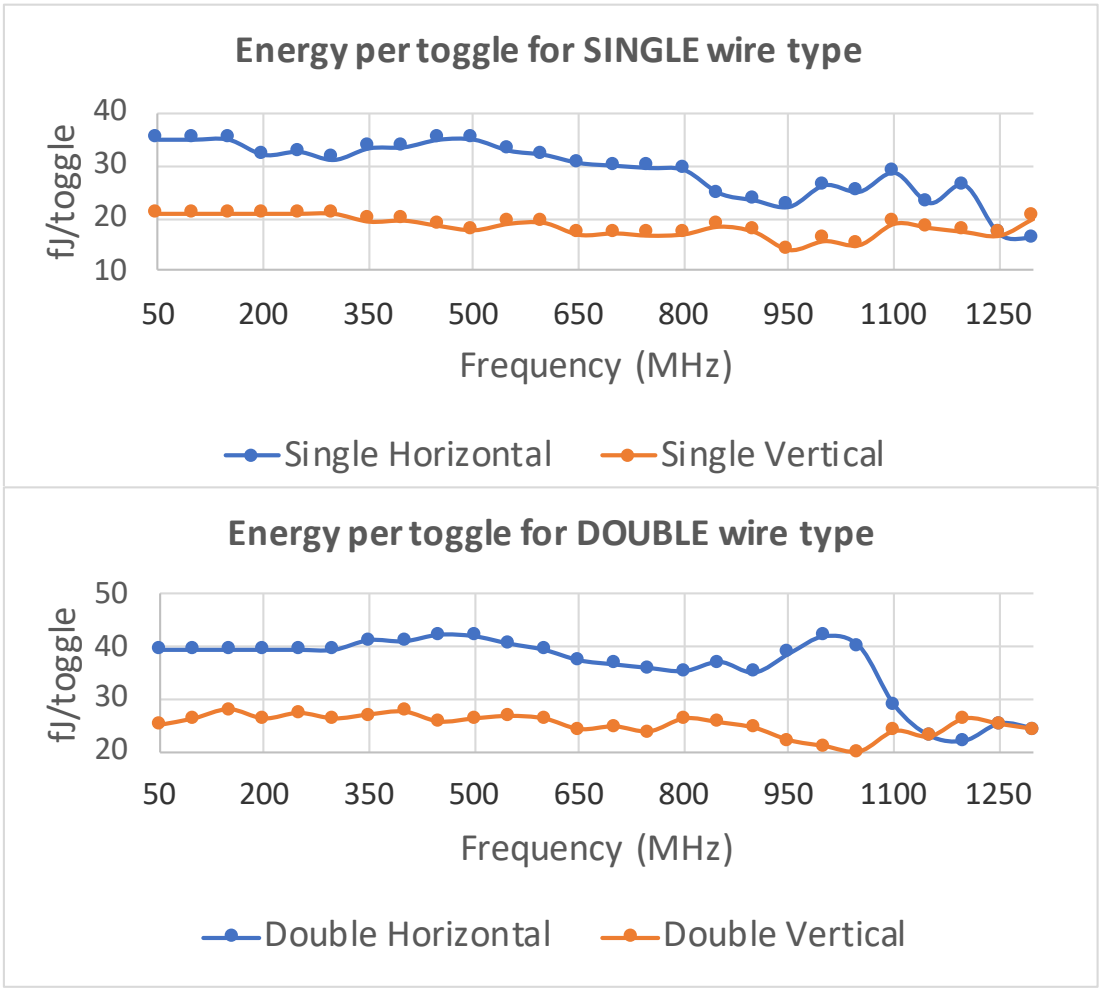


Figure 3.18: Energy per toggle for characterised wire lengths (SINGLE type and DOUBLE type).

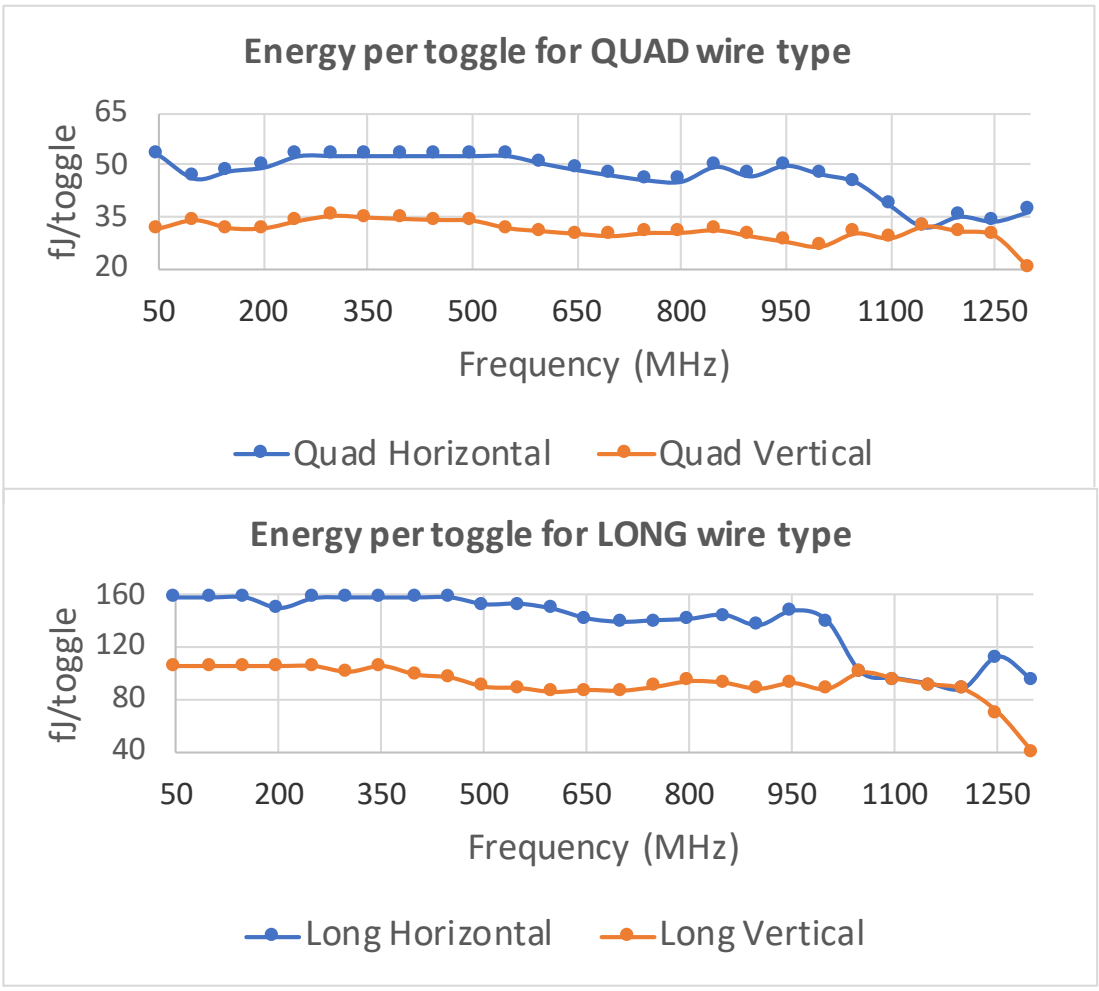


Figure 3.19: Energy per toggle for characterised wire lengths (QUAD type and LONG type).

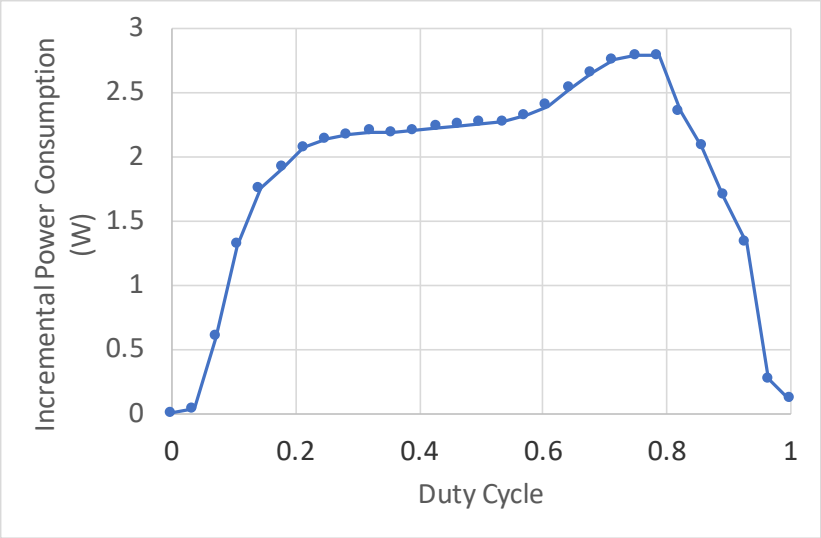


Figure 3.20: Power consumption versus duty cycle. This experiment was conducted with 1260 glitch amplifiers producing 1GHz routing to its local wires.

not publicly available. When going toward both extremes, we believe the MOSFET's strength does not allow the signal to have a full voltage swing, therefore, lowering the $V_{peak-to-peak}$ and consequently lowering the dynamic power consumption.

Vendor power estimator tool

The Vivado suite provides a power estimator tool. Using the following TCL command allows us to set the switching activity for a specific wire:

```
set_switching_activity -static_probability 0.5 -signal_rate 200
    [get_nets single_horizontal*]
```

The example would set the switching activity for all single horizontal wires to 100MHz and a duty cycle of 50%. Please note that the tool alternatively allows setting a parameter `toggle_rate`, which describes the percentage of which a signal changes related to the clock frequency. However, in this work, we use the toggle rate for denoting the frequency at which we drive our different local wires. We found that the reported power is scaling linearly with the toggle rate (which we set with `-signal_rate`). Interestingly, there is no upper bound and it is possible to set the toggle rate in the THz range and Vivado is reporting the corresponding power consumption that is in the kW range just for a small subset of the wires.

Wire Types	Number of Wires	Energy / toggle	Dynamic Power/wire	Theoretical maximum power consumption	WPP	Theoretical maximum number of wires can be used
Single Horizontal	1555200	31 fJ	31 uW	48 W	0.21	7259499
Double Horizontal	3110400	39 fJ	39 uW	122 W	0.54	5750677
Quad Horizontal	3110400	50 fJ	50 uW	155 W	0.69	4522156
Long Horizontal	1555200	150 fJ	150 uW	233 W	1.04	1500575
Single Vertical	3110400	19 fJ	19 uW	58 W	0.26	12003813
Double Vertical	3110400	26 fJ	26 uW	80 W	0.35	8790665
Quad Vertical	3110400	31 fJ	31 uW	98 W	0.44	7143214
Long Vertical	1555200	95 fJ	95 uW	148 W	0.66	2357909

Table 3.5: Estimated WPP of a legal design that only uses normal wires as the power wasting medium in Alveo U200 at 500MHz. This excludes static power and power wasting on connecting wires and other components.

Power-hammering potential

With the results we have in the last section, it is possible to estimate the portion of power wasting through wires of a malicious circuit running at a standard frequency range that does not violate any digital design rules. We selectively use 500MHz as this is relatively easily achievable without the risk of timing violation. Moreover, that speed can be distributed through a high-fanout net across the chip, meaning that a single fast toggling signal (such as an oscillator) is sufficient for excessive power-hammering potentials.

From Table 3.5, we can see that at 500MHz, power wasting on different wire types could contribute from 21% (Single Horizontal Wires) up to 104% (Long Horizontal Wires) of the total power budget. It should be noted that this does not include power wasting through other components and clock distribution networks. In reality, these wire types cannot be arbitrarily and individually used. For example, a LONG wire must be used with a QUAD wire. Therefore, it is much easier to exceed the power budget of an FPGA board by driving wires using fast toggling signals generated through standard RTL designs.

3.3 Chapter summary

In this chapter, we have provided a study about self-oscillators that could be created using FPGA resources. The study focuses on the oscillating frequency and dynamic

power aspect. We also provided a unit to quantify power-hammering potential - Waste Power Potential. It was found that the power-hammering potential of a data centre FPGA could reach up to 17x the rated power consumption if no countermeasures are taken. Additionally, we have provided an estimation of energy that is consumed for each toggle for the main wiring resources in Xilinx UltraScale+ FPGAs. This not only shows that a power-hammering attack is possible with a standard RTL design but also enables us to better detect the risk of such attacks. Based on the knowledge studied, we will investigate if we could mount an attack on AWS, a big Cloud Service Provider offering FPGA-accelerated services.

Chapter 4

Case study on attacking AWS F1 FPGA instances

In this chapter, we will reveal the general security implementation of an FPGA-based Cloud Infrastructure. This is important to assess if cloud service providers are more exposed to vulnerability when integrating FPGAs into their infrastructure. This study focuses on attacking AWS F1 because these instances are widely used and had been among the first public offerings. The CSPs Alibaba, Huawei, and Nimbix offer FPGA instances featuring the same Xilinx-VU9P device that is used for AWS F1, and these CSPs use the same Xilinx vendor DRC checks and provide an equivalent security architecture to AWS. Therefore, the here presented attacks and countermeasures are deployable in other FPGA cloud settings. Microsoft Azure uses both Xilinx and Intel FPGAs. Xilinx and Intel FPGAs are similar and share fundamentally the same vulnerabilities. For instance, power-hammering on Intel Stratix-10 was researched in [48].

We assume a scenario where an attacker (eventually using a counterfeit identity) deploys power-hammering designs on AWS F1 instances. This scenario cannot be prevented through protocols for secure remote reconfiguration of FPGAs that prevent tampering with configurations, as shown in [7]. Such approaches are useful for closed systems, but cannot be used by a CSP who wants to provide an easily usable service to a large customer base. Therefore, CSPs are using a security infrastructure that inspects user designs before FPGA deployment and monitors the FPGA board at runtime, as will be presented in the next section.

We would like to acknowledge that we asked Amazon Web Services to authorise us to use their equipment for our security related experiments before running any experiments on their instances.

	Measures	Description
Fence 1	Integrity Check	Checks for design tampering (*.dcp)
	Unrouted-Net Check	Checks for completeness of the design implementation
	AWS Shell Check	Checks compatibility with AWS Shell
	Device DNA Check	prevent user designs to access DNA_PORT (device ID)
	DRCs	Checks for unrouted nets, dangling nets, timing violations, combinatorial loops, and other design errors
Fence 2	CSP-side Bitstream Generation	FPGA bitstreams are generated by AWS (users cannot use their own bitstreams)
Fence 3	Virtual Programming	AWS restricts access to bitstreams and programming of FPGA through a custom (hypervisor) API
Fence 4	Power Warning	Power monitoring and assertion of power warnings
	Clock Gating	Gate user clock if power consumption reaches a threshold
	Over-Temperature Shutdown	Triggers shutdown sequence when temperature exceeds the critical value to prevent permanent damages

Table 4.1: Current AWS protection fences: Fence 1 – Design Inspection; Fence 2 – Bitstream Generation; Fence 3 – FPGA low-level API; Fence 4 – FPGA runtime monitoring

4.1 The AWS FPGA security architecture

AWS (like all other cloud service providers that are offering FPGA instances) has implemented multiple security fences to protect its equipment and to ensure stable operation for all users. The following sections introduce these fences in more detail.

4.1.1 Fence 1 – Design inspection

Design inspection is executed during accelerator registration, where a design is made available to be later used in the deployment phase. To implement any designs on AWS, users must follow a strict design flow (see Appendix A). All designs have to pass the DRCs, as listed in Table 4.1. The input to the registration process is a netlist in the vendor propriety design checkpoint format (DCP).[1]

First, an *integrity check* (an SHA-256 hash) confirms that the DCP file has not been corrupted. Then a *scan for unrouted* nets detects malicious designs that try tapping into other parts of the system (e.g., the cloud shell). The *PR region check* confirms that the static shell (AWS shell) will not be compromised [119]. This check is done by analysing the user DCP, which includes both the shell (provided by AWS) and the user logic. The *Device DNA check* prevents users from accessing an FPGA-specific ID. The device DNA is normally used for access control or cryptography protocols [124]. Without access to the device DNA, users have no trivial way to identify their allocated FPGAs.

The *FPGA vendor Design Rule Checks* scan for design violations, including unrouted nets, dangling nets, combinatorial loops, etc. The design tool Vivado 2019.1.3 provides more than 5000 DRCs with the severity of critical warnings and errors, which would prohibit the flow from generating bitstreams if used. The severity level (e.g., error) of many DRCs can be changed, and it is the duty of the CSP to use the right DRC receipt. Note that only 3 DRCs scan for combinatorial loops (which allows implementing fast ring-oscillators and soft logic voltage/temperature sensors). The DRCs include:

- **LUTLP-1** and **LUTLP-2** check for LUT-based combinatorial loops.
- **RPCL-1** detects ”any” combinatorial loops in the design. Here, ”any” refers to

[1] Users can provide encrypted DCP files to protect their IP. However, the CSP could break the cryptographic mechanism. For example, attackers can generate the bitstream from the DCP and annotate the logic functions back to the DCP netlist [43, 67]. Therefore, users have to trust the CSP for IP protection. This aspect is covered in [106].

Design	Side-channel Attacks	DoS Attacks	Suited for sensors & PUFs	Provisional power gain (W)	WPP
1: MUX7 ROs	✓	✓	✓	463	2.06
2: MUX8 ROs	✓	×*	✓	123	0.55
3: CARRY8 ROs	✓	×*	✓	123	0.54
4: DSP ROs	✓	×*	✓	25	0.11
5: Latch ROs	✓	✓	✓	980	4.36
6: FF Glitch Generator	✓	✓	×	519	2.31
7: LUT Glitch Gen.	✓	✓	×	1141	5.07
8: Glitch Amplification	✓	✓	×	2721	1.84
9: Enhanced CARRY8	✓	✓	✓	369	1.64

Designs 1,2,3,4,5,6,7 are taken from Table 3.2. Design 8 is taken from [71]. Design 9 is an enhancement of Design 3 (see Figure 4.2b). *: On their own, Designs 2,3,4 are not well suited for DoS attacks. However, DOS is possible using additional glitch amplification.

Table 4.2: Malicious designs that are currently deployable on AWS.

the vendor information. However, this test fails to detect several combinatorial loop designs, as listed in Table 4.2.

Additionally, the design inspection checks for timing, IO, and power violations. However, this step only provides reports and violations will not prevent the AFI generation. This allows an attacker to deploy overclocked designs for implementing side-channel attacks [109].

AWS provides a few default clocks for a user to choose from. However, using these clocks is not enforced and users can generate their own clocks with higher clock speeds than the default clocks. This is a security threat for both power-hammering and implementing voltage/temperature sensors.

4.1.2 Fence 2 – Bitstream generation

After passing the design inspection, the Amazon FPGA Image (AFI) is generated. This is a file consisting of the configuration bitstream and some metadata (that we ignore here). The FPGA vendor bitstream generation tool ensures the correct translation of the netlist (given as a DCP file) into a bitstream. It is important to understand that, due to

the low-level implementation of the actual FPGA fabric, it is possible to create short-circuit situations that can draw excessive current. While an individual short-circuit is not a concern (about a few mA per short-circuit) [22], an attacker could possibly deploy hundreds of thousands of shorts with corresponding consequences.

Because attackers cannot inject their own bitstreams, we cannot bypass this fence. Consequently, we have not been able to attack a cloud FPGA with short-circuits (as this requires bitstream manipulations). All designs passing Fence 1 will pass Fence 2 and 3.

4.1.3 Fence 3 – FPGA API

After the AFI is generated, it can be loaded onto the FPGA fabric. However, AWS blocks any direct access to the generated AFI as well as the programming and debugging processes. AWS is providing a set of command-line tools to program and debug the FPGA [14], and programming is only possible through an AWS-provided API. For programming, the command *fpga-load-local-image* is encapsulating the vendor programming mechanism [121]. For debugging, AWS provides users with virtual LEDs, virtual DIP switches, and virtual JTAG [14]. This is available through the PCIe connection.

Like with the bitstream generation, we cannot bypass Fence 3, which is active when deploying a design. Security Fence 3 prevents attackers from directly accessing the configuration bitstream (AFI) and the configuration port.

4.1.4 Fence 4 – FPGA monitoring

At runtime, AWS uses three safety mechanisms. The first one is *power monitoring and condition monitoring*. The Xilinx VU9P data centre FPGAs provide a system monitor mechanism featuring three 10-bit 200kSPS ADCs to read FPGA temperature sensors and core voltages [120]. Related work suggests that the available sampling rate is insufficient to detect quick voltage transients [70] and to generate power warnings if excessive power is ramped up rapidly. While the present mechanism is sufficient for virtually all practical benign designs, it is insufficient for the malicious circuits that we deployed in this work.

The second safety mechanism allows slowing down a user design to limit power consumption below a certain threshold ($\approx 100W$). When exceeding this power budget, the shell will *gate all user clocks* to stop switching activity and corresponding dynamic

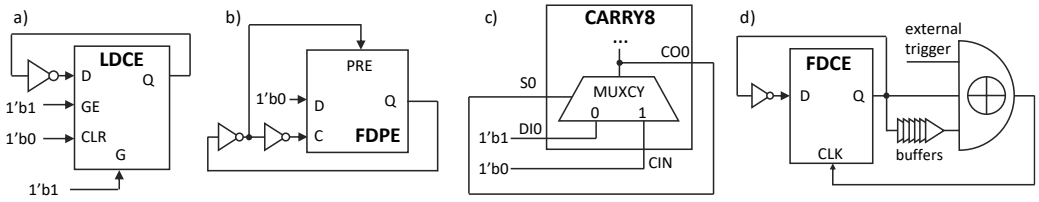


Figure 4.1: Oscillator designs deployable on AWS F1 instances: (a) transparent latch, (b) flip-flop with asynchronous preset, (c) ring-oscillators implemented through carry-chain logic, (d) a self-oscillating circuit using glitch amplification.

power.

As the last safety mechanism, AWS uses a failsafe mechanism that is built into the FPGA for protecting the device from overheating. AWS has set the critical temperature to 125°C (which is the default value). When exceeding this temperature threshold, the FPGA is *triggering the shutdown sequence*, which deactivates all drivers of the FPGA (including all drivers for clock nets and most other resources). This is activated globally and will cut off the PCIe connection between the FPGA and the host.

4.2 Power-hammering attacks on AWS EC2 F1 instances

Since AWS F1 instances are equipped with Xilinx UltraScale+ VU9P FPGAs, we ran experiments under lab conditions using an FPGA board featuring the same FPGA (we used an Alveo U200 data centre card) to measure power increments caused by different power-hammering circuits. This allows us to calibrate our power-hammering attacks without the need to measure power on AWS hardware. Furthermore, we used the AWS tool flow for tuning our designs to pass the design inspection (Fence 1).

Chip power dissipation consists of both static and dynamic power. In benign designs, the dynamic power can contribute from 20% to 70% of the total power dissipation [8]. The dynamic power consumption depends on switching activity which is expressed by the activity factor α that denotes how often a signal can toggle within one clock period. Because α is data-dependent, estimates are commonly used to model this effect. E.g., the power estimator from Xilinx sets α to 12.5% by default [122]. However, malicious circuits can switch 100$\times$ faster as investigated in Chapter 3, and the power estimator is not well suited to catch such designs.

Figure 4.2 and Table 4.2 show attacks that bypass the security Fences 1-3. The reported power gains had been derived on a small region of the FPGA ($\approx$ 10% of the total resources) and then scaled up according to the user FPGA resources available

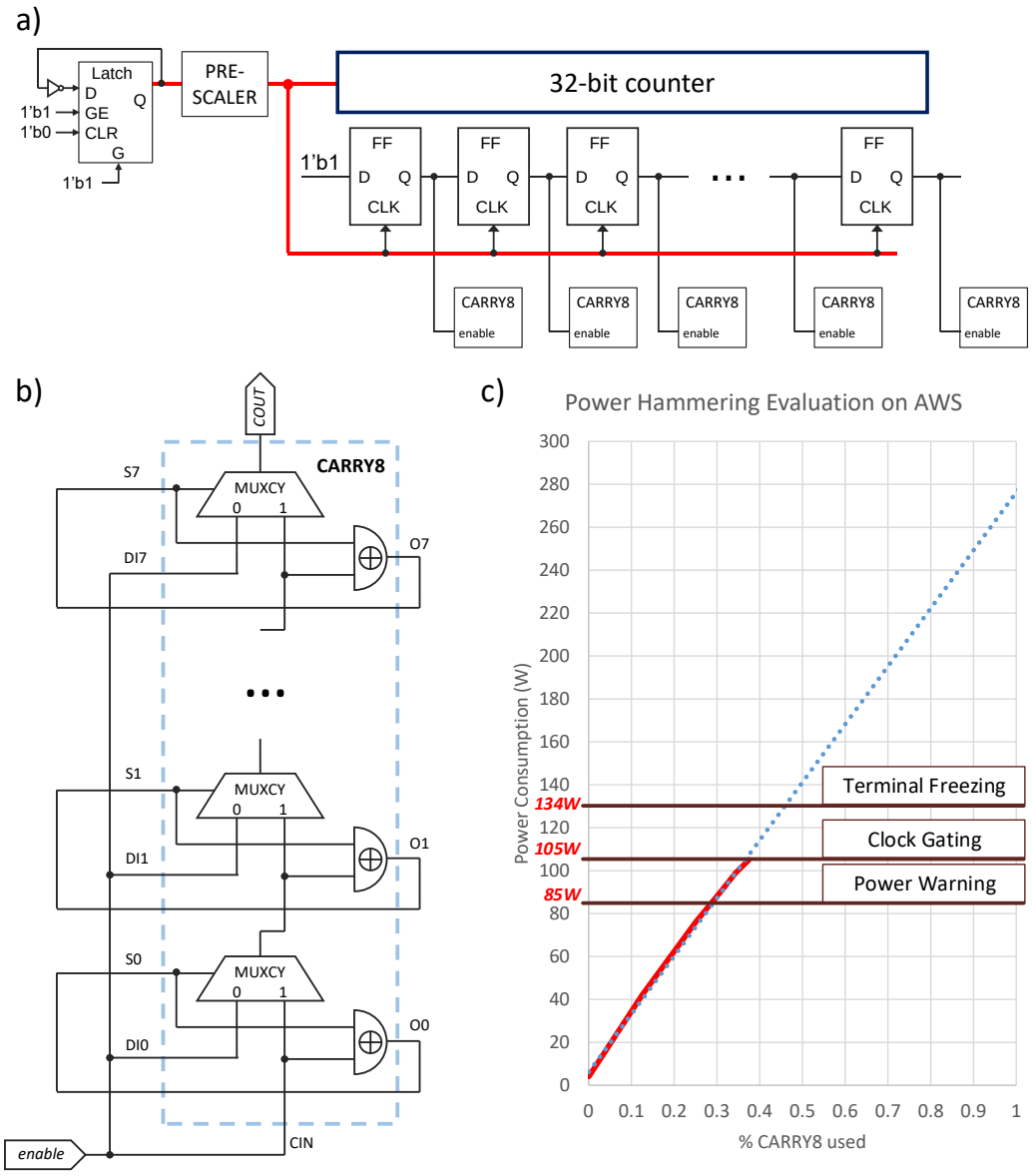


Figure 4.2: Power-hammering designs and power evaluation on AWS. a) Self-clocked design to bypass the clock gating protection. b) One carry-chain primitive forms 8 combinatorial loops. c) Evaluation using 81920 carry-chain primitives. The continuous red line is the recorded power measurement. The linear dotted blue line shows the expected power consumption when leaving the experiment running freely.

on AWS F1 instances. We divided the malicious designs into two groups: 1) designs using combinatorial loops and 2) designs creating glitches. The latter is shown in Figure 4.1d, which uses a toggle flip-flop where the output is routed to an XOR, but with different latencies in order to create glitching at double the frequency of the toggle flop itself. This amplified clock is fed back to the flop, causing a self-propelled oscillation. We used this concept to amplify the glitching of signals, which are then used to drive a large number of routing wires. This results in the 2.7 KW power gain level and WPP of 1.84 as reported for design 8 of Table 4.2. Amplifying switching activity is a pattern that allows boosting slow oscillators to much higher frequencies with corresponding power-hammering potential.

For the denial-of-service experiments deployed on AWS F1 instances, we used a variant of design 3 using the carry-chain primitive (CARRY8) and modify it to implement 8 combinatorial loops for each primitive (see Figure 4.2b). The enable signal is used to control the oscillation.

4.3 Bypassing Fence 4 – FPGA monitoring

When exceeding certain power levels, F1 instances trigger different exceptions. We observed a power warning when power consumption reached 85W. When reaching about 105W, the shell stops all user clocks. To bypass this clock gating mechanism, we implement a clock source using a ring-oscillator using a transparent latch. As shown in Figure 4.2a), this oscillator was used to sequentially activate a chain of power-hammering circuits, see Figure 4.2b). This was slowed down using a prescaler resulting in a power rising level of $0.4W/sec$. With this, we linearly increase dynamic power over time such that the time corresponds to the power consumed. This allows us to observe the FPGA monitoring behaviour even if the shell applies clock gating. For the experiment, we instantiated a chain with 81,920 CARRY8 primitives for power-hammering. We have deployed this design on AWS, and the result is shown in Figure 4.2c. We observed that when increasing power beyond the clock gating point (105W), the SSH terminal freezes at about 134W, which may have triggered an over-temperature shutdown. All power levels had been measured indirectly by measuring the time. For improving accuracy, we used a reference counter to measure the speed at which the power-hammering circuits are activated. Our experiments confirm that 1) the clock gating can be bypassed and 2) exceeding 134W can freeze an instance with a loss of the SSH connection.

4.4 FPGA fingerprinting on AWS EC2 F1

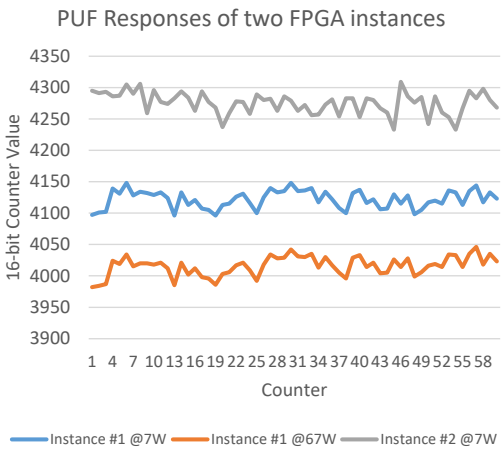


Figure 4.3: AWS F1 FPGA PUF responses.

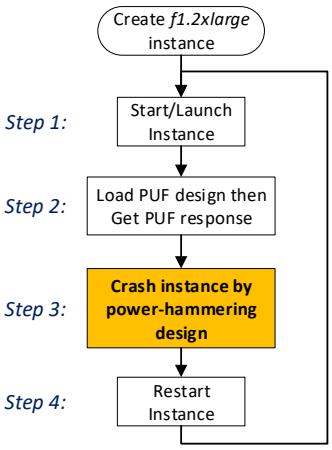


Figure 4.4: AWS attack flow.

In the last paragraph, we presented an experiment that crashed an AWS F1 FPGA instance. In order to investigate the behaviour of an instance after being crashed, we need to identify individual FPGAs (or corresponding instances) to determine their availability for measuring that a DoS attack was mounted successfully. AWS does not provide an identifier that could be used to identify a specific physical instance or FPGA (e.g., a serial number, MAC, etc.). Additionally, AWS prevents users from accessing factory-programmed keys on the FPGA, including the 96-bit factory-programmed unique DNA (Device DNA) or the 32-bit user-defined eFUSE [124]. Therefore, we implemented a PUF for fingerprinting the FPGAs. From the fingerprint database we collect, we will know exactly which FPGA fabric we are currently connected to. Note that these PUFs are not used for any security protocols, and security concerns against PUFs would not apply in our context. PUF implementations on FPGAs are well studied [62] and our contribution is providing an implementation that is deployable on AWS F1 instances where the traditional LUT-based combinatorial feedback loop cannot be used (see also 4.1.1).

Recently, a group from Yale University used DRAM decay to fingerprint AWS F1 instances [103]. That approach uses the fact that each FPGA chip has access to four dedicated DRAM modules. However, we have not considered this method because 1) DRAMs are removable; 2) the method requires three AFI loading steps, which means tripling the time for experiments; and 3) it is trivial to be mitigated by AWS. This is because the attack uses two different user designs, one with a memory controller

and one without. The second design is used to temporarily disable DRAM refreshes, with the resulting decay being the fingerprint signature. However, AWS could mitigate this in Fence 3 by i) simply not allowing to change between designs that once use and once not use memory, which would be a rather unusual case and therefore not affecting benign customers or ii) including a memory blanking phase that could be applied transparently to the user.

As an alternative, we implemented an oscillator-based PUF for fingerprinting. We implemented 60 ROs using transparent latches. We constrained all ROs with the same physical routing paths and the cycle path delay was reported to be $456ps$ each (by the Xilinx Vivado tool). For the fingerprinting, we counted the response of each RO separately over a time period of $2048ns$. Since the VU9P FPGA used in AWS has 3 super-logic regions (SLRs), which are separate dies integrated together on an interposer, we used 3 identical PUFs (one per SLR) to increase the confidence level of our fingerprinting. The unique ID of each SLR is represented through a set of 60-counter values. We identify the match of two PUF responses by calculating the Pearson Correlation Coefficient between two counter value sets. In our experiments, we defined that two sets match if the correlation coefficient is larger than 85%. The coefficient is selected after trials on the same FPGA board by placing ROs randomly on the FPGA layout to introduce local heat spots. According to Evans, the coefficient is categorised as a very strong correlation [42]. We acknowledge that the PUF will need more detailed evaluation and calibration in order to be used for security application because in some conditions such as hotspots or busy adjacent wires, the PUF may not yield accurate results. However, as the PUF circuit and therefore the proximity of wires used for the PUF implementation is fixed, the impact is estimated to be low.

Figure 4.3 shows three PUF responses where the orange and blue traces were derived from the same FPGA but at different power levels (7W and 67W). These traces show a very strong correlation (94%) in the shape of the count values and the gap between values is due to the temperature difference. Therefore, the PUF responses are robust to temperature changes. The top grey trace was derived from a different FPGA as a reference. These experiments allow 1) measuring the temperature of an FPGA (which could, for example, reveal information about a previous design) and 2) fingerprinting an FPGA to derive secrets of the cloud service provider. This can include the number of total instances or scheduling policies.

4.5 Mounting a DoS attack on AWS EC2 F1 instances

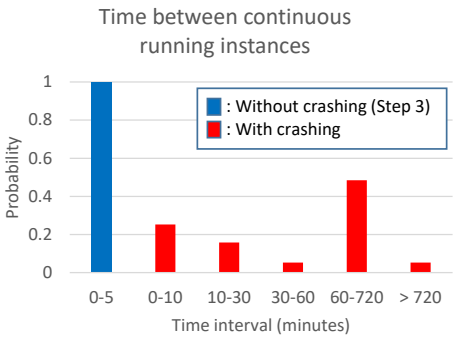


Figure 4.5: Time interval between two running instances.

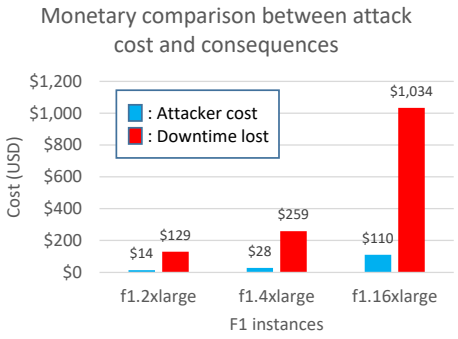


Figure 4.6: Estimated attack cost and loss after 100 attacks with an attack time of 5 minutes and downtime of 52 minutes.

We conducted experiments on the on-demand `f1.2xlarge` instances in the North Virginia `us-east-1` region. The attack was mounted from Apr 17, 2020 to Apr 28, 2020. For the power-hammering experiments, we used a moderate power level of 396W to crash the FPGAs (ramped-up rapidly) as our intention was not to damage equipment. It is worth mentioning that we applied this power constant and that by creating resonance effects in the power regulation circuit (by using a pulsed stimulus of defined frequency and duty cycle), less power is likely sufficient to crash the board or to cause potential damages [36].

The attack was performed, as shown in Figure 4.4 (see Appendix A for more details about the lifecycle of an Amazon Instance). After creating an instance, we first fingerprinted the FPGA and then crashed it using power-hammering. After this, we started the attack with the next instance. As a reference, we ran experiments without crashing the instance (without *Step 3*) to measure the time it takes under normal conditions to receive the same FPGA instance again. With this, we confirmed that an F1 instance stays on the same host computer if it is left running [16]. After five tries, we observed that without crashing, it took less than five minutes to i) establish an instance, ii) fingerprint it, and iii) shut it down, and we always got the same FPGA instance or one belonging to a small pool of instances. After this, we run another 96 experiments including the crashing of the instance. As shown in Figure 4.5, we observed that, on average, it took about an hour before we were able to start a new instance after a crash and the longest waiting time was up to 22 hours.

Interestingly, when comparing the PUF responses, we found that the *minimum time*

to get a previously crashed F1 instance re-allocated is about 52 minutes. And we found only one occasion where the same FPGA was allocated in two consecutive experiments. This value is much below the rates reported in [103] with 25% and our reference experiments. This is a strong indication that the host machine needs to re-initialize the crashed FPGA and the minimum downtime is close to one hour. Our experiments also show that the crash behaviour is not consistent. This could be an indicator that it required a human operator to bring up an attacked instance.

The here presented DoS attack can potentially be deployed in different scenarios. In one, a malicious design is placed in the AWS marketplace, and after a forced crash, users will experience extensive delays to receive new instances. Note that because we can use routing resources that are left unused in a particular design to build power-hammering circuits, the extra cost for embedding this attack would be neglectable. Alternatively, an attacker could temporarily prevent AWS from selling the service of crashed F1 instances. Given the service is billed in each second (if it runs for less than a minute, then the cost is rounded up to the next minute) [56], we can estimate the monetary loss. Figure 4.6 shows the estimated cost and possible loss when running 100 attacks. As we can see, the downtime loss is about an order of magnitude greater than the attacking cost. The ratio could potentially be higher if an attacker can prevent AWS from fulfilling quality of service agreements.

Because we do not want to cause potential damage to AWS equipment, we have not conducted experiments with greater power-hammering potential (e.g., using glitch amplification). With higher power-hammering potentials, there is a potential loss in customer confidence and possible loss of equipment, which states a greater financial risk.

4.6 Chapter summary

In this chapter, we provided a real-world example of a Denial-of-Service attack using power-hammering on an FPGA cloud provider. This demonstrates the profound effects of power-hammering on cloud computing and the necessity of having an effective tool to prevent the attack before the malicious circuit is even mounted to the FPGA board. The following chapter presents countermeasures to prevent such attacks.

Chapter 5

FPGA netlist scanner for malicious circuits

Having discussed the security issues and case study of FPGA-based infrastructure in previous chapters, we strongly believe there is a need to ensure a secure environment for using FPGAs in the cloud. Current countermeasures could be separated into the reactive group and the proactive group. Reactive approaches are based on responding to the early-stage effects of the attack. For example, a study in [70] shows a mechanism to detect voltage transients which are caused by power-hammering attacks with a resolution up to the nanosecond range. On the other hand, proactive approaches aim to prevent attacks before they actually happen. Proactive countermeasures could be further divided into intrusive and non-intrusive methods. Intrusive methods involve changing the design in order to hide or mask power signatures [97]. Non-intrusive methods require design examination instead of modification. A typical example of design examination is Design Rule Checks (DRCs) embedded in almost any Electronic Design Automation (EDA) tool. The pros and cons of each approach are summarised in Table 5.1.

We believe that non-intrusive proactive countermeasures are more effective without exposing potential risks compared to the other. In this chapter, we will present a scanning mechanism that is able to detect malicious circuits in an FPGA design. Our scanner could be embedded as DRCs for EDA tools or it could be used separately as third-party virus scanning software for FPGAs. The Python implementation of the scanning mechanism was done by the PhD Student Kaspar Matas. This thesis contributes to building an FPGA virus scanner. The specific ingredients provided through this project include:

		Advantages	Disadvantages
Reactive Approaches		IPs can be reused development flow can be unchanged	Potential hardware damages before detection, potential false positive results in power-noisy environment, potential false negative results in side-channel attacks
Proactive Approaches	**Intrusive**	Improve security robustness of designs	Potential area and performance trade-offs, legacy IPs needs re-design
	Non-intrusive	No affects on area or performance, prevent attacks before any potential damages	Checks need frequent update to keep pace with new attacks

Reactive approaches: [70]
Intrusive proactive approaches: [97]
Non-intrusive proactive approaches: [37, 125]

Table 5.1: Pros and cons of approaches to prevent power attacks for FPGA-based infrastructures

1. defining the FPGA virus signatures;
2. decoding the architecture graph for Xilinx UltraScale+ FPGAs;
3. creating a custom DRC to integrate the scanning mechanism into the FPGA design flow;
4. generate testing circuits to evaluate the FPGA virus scanner.

5.1 Malicious circuits scanning mechanism

In this section, we will introduce the malicious circuit scanning mechanism for FPGAs that could be effectively embedded into existing Xilinx DRC.

5.1.1 Hardware versus software virus scanning

In software systems, the confidentiality of data and task integrity are usually protected by different layers that may go beyond what a software virus scanner is testing. This includes protection mechanisms provided by the software operating system (OS) or runtime environments. Software binaries encode the functionality of a program primarily as sequences of instructions. Consequently, a virus scanner for software binaries will

Software	Hardware
Check software binaries using regular expressions and cryptographic hash matches	Analyse netlists and check graph properties
Check address ranges to prevent executing malicious code	Check location information of primitives and wires
Bounds checking and memory randomisation to prevent buffer overflows (eventually by an OS)	Check volume of netlist written to prevent configuring adjacent regions

Table 5.2: Contrasting protection mechanisms: software versus FPGA hardware techniques.

include a pattern-matching engine, typically searching for regular expressions, which are also known as virus signatures.

Contrarily, the functionality of a module is encoded as a netlist - whether in the form of a bitstream or a design checkpoint - which in turn is a structural representation given as configured FPGA primitives and configured switching elements (i.e. multiplexers). A netlist can be modelled as a graph, and the whole physical FPGA implementation process can be described by graph transformations, as summarised in Section 5.1.2. Consequently, *a virus scanner for FPGAs needs a checker engine for graph properties*. For example, in Chapter 3, we examined ring-oscillators in more detail that in one variant are implemented as cyclic combinatorial circuits (e.g., a LUT that has an output connected to its input without passing a flip-flop). By scanning a netlist (i.e. its corresponding graph representation), we can spot such oscillators.

A software OS and most software virus scanners commonly check memory addresses (or memory ranges) used by programs to ensure that data is not corrupted in malicious ways or that, for example, data segments are not executed as code. Similarly, a virus scanner for FPGAs has to check that a configuration bitstream is not corrupting the configuration context of other parts of the system, including other configurations or states of other parts of the system (e.g., the surrounding shell that commonly provides DDR memory and PCIe access). Consequently, a virus scanner needs a netlist parser that ensures that a module will only change the configuration context of resources allocated to that module. This requirement includes parsing addressing information that encodes locations of primitives on the FPGA fabric as well as tracking the volume of configuration data that is written to the device. The latter tracking prevents a kind of

a buffer overflow that can arise when configuring Xilinx FPGAs[1]. This attack would exploit that Xilinx FPGAs perform something similar to an auto-increment that keeps configuring an FPGA as long as it receives configuration data through a configuration port. As a consequence, this could overwrite the configuration of the fabric outside an intended module bounding box.

To some extent, the tracking of the configuration bitstream length is equivalent to buffer overflow detection and prevention techniques (e.g., bounds checking) as embedded into some compilers like the Clang frontend for LLVM [26].

We summarised the main differences between software and hardware virus scanning in Table 5.2. For full system security, it requires hardware support from the runtime system. For instance, systems commonly provide memory management units (MMUs) that can protect memory regions against malicious accesses. These units are also available in CPU-FPGA hybrids such as Xilinx Zynq UltraScale+ devices or Intel Stratix-10 SX SoC devices; and these chips include dedicated IOMMUs to protect the memory subsystem from malicious accesses initiated from the FPGA side (e.g., by an accelerator module). Alternatively, MMU functionality can be implemented in the FPGA's soft logic (commonly as part of a shell [10]). In this study, we are, in particular, focusing on FPGA vulnerabilities at the electrical level because protecting a system at the system level is very well studied and, therefore, not further covered here.

5.1.2 Modelling the FPGA virus scanning problem

Formally, any FPGA architecture can be modelled by its architecture graph $G_A = (V_A, E_A)$ which includes as its node primitives V_A^P and switches V_A^S with $V_A = V_A^P \cup V_A^S$ as well as directed edges E_A between the nodes representing wires or connections[2]. When a module is implemented for an FPGA, its specification (e.g., some RTL code) will undergo several transformation steps, including logic compilation, technology mapping, placement of primitives, routing, and ultimately the generation of the configuration bitstream. Concisely, we can say that the technology mapping is an allocation and mapping of primitive Boolean functions (the result of the logic synthesis step) to a set of connected primitives (including their internal configurations). The result of this step is a *netlist*, which is a graph $G_N = (V_N, E_N)$, where the nodes are FPGA primitives.

[1] Please note that this problem is not necessarily bound to a specific vendor but that this problem is best understood for Xilinx FPGAs which dominate the research on runtime reconfigurable systems and the commercial cloud providers.

[2] For the sake of clarity, we deliberately omit a discussion about bidirectional wires that had been available in older FPGA architectures.

During placement, the nodes get placed on the architecture graph G_A, which is a binding β of the netlist nodes $V_N \rightarrow V_A^P$. In practice, this means that we annotate for each node in G_N the location coordinate L of the corresponding primitive of the FPGA:

$$V_N \rightarrow L(V_A^P), \forall\, V_A^P \in V_N$$

The process of routing can be defined as computing a binding of the netlist edges E_N to switches V_A^S and wires E_A. In general, this is a quite complicated process, and the actual routing has, among other things, to find spanning trees (for multiple edges $e_n \in E_N$ that have the same source node in V_N). Primitive nodes commonly have multiple input and output ports $p \in P_t$, where P_t is the set of ports for a specific primitive type t. The routing information can be seen as a set of switches (a list of nodes in V_A^S) and wires (a list of edges in E_A) that are used to implement each connection in E_N. The configuration of a switch is given by none (if the switch is not used) or exactly one edge from another node (or port in the case the source is a primitive node), which in turn represents the selected routing multiplexer input (e.g., in a switch matrix).

A placed and routed netlist can be directly mapped to a bitstream and encodes the exact configuration of each element $V_A^P, V_A^S \in V_A$. It is essential to understand that this mapping is *reversible*, meaning that a bitstream can be mapped back to a placed and routed netlist. However, this mapping needs the architecture graph to rebuild the routing, which is only encoded as segments in the bitstream, rather than as complete paths. On the contrary, a netlist generated through the implementation flow still provides a substantially higher level of abstraction than the bitstream. This is because a netlist typically includes information such as hierarchies, symbolic names of nets and logic blocks, and information on signal vectors, which cannot be easily decompiled from an FPGA configuration binary. This fact is similar to software compilation into obfuscated program binaries that also do not allow to decompile symbolic names and hierarchies.

This project uses the reversible correspondence between bitstream and netlist to rebuild flat netlists that provide all primitives V_N^P and all switching multiplexers V_N^S, but that will not offer any higher level information (such as symbol names or Boolean equations). For the remainder of this thesis, we will use G_N to refer to a netlist that is rebuilt from a bitstream for detecting virus signatures.

Please note that the goal of this work is not to provide or offer a reverse-engineering tool for FPGAs but to show that configuration bitstreams are well suited to detect malicious circuit constructs in a module. Related work that focuses explicitly on reverse-engineering includes [107, 43, 114].

5.1.3 Detecting self-oscillating circuits

As mentioned in Section 2.1, it is of paramount importance to identify self-oscillating circuits in a design to be deployed. The following paragraphs are devoted to different classes of self-oscillating designs.

Ring-oscillators

Ring-Oscillators break the fundamental model of register-transfer level (RTL) descriptions where a circuit is described by

1. registers, including FPGA slice flip-flops, pipeline registers (e.g., inside DSP primitives), or memories, and

2. transforming logic that is forming *acyclic combinatorial paths*.

These paths can be described by a network of elementary Boolean functions implemented by look-up tables (LUTs), or DSP blocks[3] that are located between the registers.

It is a good design practice to follow the RTL design principle on FPGAs [115], and this is also the model commonly generated by High-Level Synthesis (HLS) tools [32]. In this study, we assume that all states are stored in flip-flops or other synchronous memory elements (which is the typical case for FPGA designs). Circuit analysis using latches (which are sometimes used in ASIC netlists) is a well-studied topic, and there is no fundamental obstacle to transfer the here presented methodology to circuits based on latches.

To perform a search for cycles, we have to refine our netlist model so that each port $p \in P_t$ of a primitive V_N can be either a register P_t^R or a combinatorial element P_t^L for routing or logic. With this, we expand $\forall p \in P_t^R$ a path search that terminates at any other register port $\in P_t^R$ or that recursively explores all paths while keeping track of duplicate ports visited in P_t^L, which would indicate a cycle.

In general, it requires an *odd* number of inverters in a cycle to form a Ring-Oscillator. The FPGA scanner is not interpreting the logic blocks for the existence of inverters, and we deliberately scan for acyclic paths only. The reason for this is that if a combinatorial block (e.g., a LUT or DSP block) implements an inverter between an input and an output can depend on other inputs and consequently on a state

[3] For a sake of clarity, we are omitting a deeper discussion on pipeline registers in DSP blocks for the remainder of this chapter, even an FPGA scanner can deal with all internal registers and pipeline stages in the DSP48 primitives which are available on Xilinx UltraScale+ FPGAs.

that is only known when running a module. The philosophy of the FPGA scanner is to flag any possible oscillator while not reporting a false positive for any design that is following RTL design principles. Moreover, the FPGA scanner is stricter than the Xilinx vendor DRC checks, which can also detect some cycles, but there are situations where the vendor DRC fails. For example, an enabled transparent latch can be part of a Ring-Oscillator, and due to the latch (which is logically a wire), this cycle would not be flagged by the Xilinx vendor DRC but by the FPGA scanner. More examples are provided in Chapter 3.

For our implementation, we used Xilinx Vivado for generating a report file containing the full architecture graph for the used Zynq UltraScale+ XCZU3 FPGA. However, that model does not explicitly distinguish between P_t^R and P_t^L, and we added this annotation through a regular expression replacement. Furthermore, the path search inside the virus scanner incorporates all combinatorial primitives, including LUTs, DSPs, carry logic, cascading multiplexers (i.e. *F7Mux* and *F8Mux* in Xilinx nomenclature), which requires an understanding of the bitstream encoding of primitive control bits. However, the actual search does not have to distinguish between different types of primitives. This scan will identify the oscillator Design 1 to 12, which are reported in Chapter 3.

It should be noted that although we want to flag all the possible combinatorial loops, some of them may be beneficial. For example, ROs can be genuinely used for security applications such as for a True Random Number Generator or for Physical Unclonable Functions [54, 35]. In these cases, we can allow to implement ROs in a designated location and use a filter to exclude the path that has the RO from the final result. By doing this, we will have control over the usage of ROs and therefore we can avoid misusing them for malicious purposes.

Self-clocking oscillators

In our oscillator evaluation section (Chapter 3), we evaluated an oscillator circuit that is based on glitches that are generated by an XOR gate with different input routing latencies and where the resulting glitches are fed back into the clock input of a toggle flip-flop for a self-propelled oscillation (see variant 15 in Table 3.2). In order to detect this kind of oscillator, we examine for each used flip-flop the corresponding clock source. If the source is a global clock network, the netlist is considered clean. If the clock source cannot be considered to be stable (e.g., a combinatorial LUT output), the netlist is rejected. In our systems [69, 10], we block configuration access to global

clock resources for any partially reconfigurable module by using BitMan [67] for preventing this kind of attack and the FPGA scanner will detect if partially reconfigurable modules try driving global clock resources.

Other oscillators

In Chapter 3, we presented further self-oscillating designs that allow bypassing the default Vivado design rule checks (DRCs); therefore, this allows an attacker to implement oscillators which can be deployed in cloud data centres. To confirm this, we ran experiments on Amazon F1 instances for all Oscillator designs listed in Table 3.2 and all designs that do not throw any warning by the vendor tools can be deployed. The remainder of this paragraph will present the corresponding detection mechanisms.

The self-oscillator detection mechanism in Section 5.1.3 scanned for the origin of a clock source, and we use a very similar mechanism to handle asynchronous reset/preset inputs of slice flip-flops which can also be used for creating self-oscillating circuits (see variant 14 in Table 3.2). To detect this, we query the asynchronous mode flag from the netlist for each used flip-flop, and in case any asynchronous reset/preset mode is used, we search from the control input (i.e. the reset/preset primitive input) backwards to find the origin of the corresponding control signal. If the origin is a combinatorial primitive pin P_t^L, the netlist is rejected while we flag a warning for registers P_t^R.

In order to prevent the Vivado tool from flagging a combinatorial feedback loop, a transparent latch can be incorporated in the loop (see variant 13 in Table 3.2). We, therefore, treat latches as combinatorial elements (essentially like a wire) and carry out a loop search as described for ring-oscillators. We also report a warning in case latches are used.

Regarding IOBUF-based oscillators, a similar approach to treat latches could be implemented. That is an IOBUF primitive which has both input and output used can be incorporated in the loop. And we could treat those IOBUF primitives as combinatorial elements. For stricter security (which has already been implemented on AWS), we could report a warning in case IOBUFs are utilised.

The here presented tests allow detecting any FPGA implemented self-oscillating circuits that have been reported in the literature, including all further variants reported in Chapter 3.

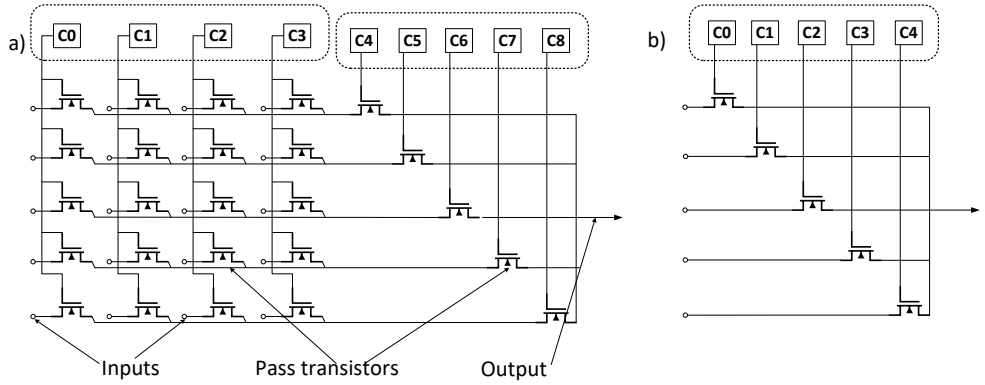


Figure 5.1: a) Switch matrix multiplexer implementation on Xilinx 7-series FPGA; b) ditto for UltraScale+ FPGAs.

5.1.4 Detecting short-circuits

In [53, 22, 92, 91], short-circuits had been reported that were implemented directly in the soft-logic on an FPGA. In older FPGA families (e.g., Xilinx Virtex-II), the fabric included some long-distance wires that could be accessed through tristate drivers at different positions, which could be used to create short-circuits inside the fabric. The deployable attack for short-circuits in modern FPGAs is based on the way switch matrix multiplexers are commonly implemented. In SRAM-based FPGAs, the multiplexers are implemented with transmission gates or pass-transistors [24] and by activating multiple inputs (i.e. switching on multiple transmission gates or pass-transistors within the same multiplexer), a short-circuit situation arises when the corresponding multiplexer inputs carry different logic levels. Therefore, by changing the input logic levels to the switch matrix multiplexers, it is possible to control the power that a shorted multiplexer is drawing precisely in time. This configuration provides the potential for DoS attacks.

As shown in Figure 5.1, 7-Series FPGAs from the vendor Xilinx implement a switch matrix multiplexer by cascading two levels of switching, each controlled through a one-hot coded configuration word (see [33, 22] for more details on FPGA switch matrix multiplexer implementations). In contrast, the multiplexers in UltraScale+ devices are smaller and use only one multiplexing level that is again one-hot encoded in the configuration bitstream. Consequently, for UltraScale+ devices, a used multiplexer input port corresponds directly to one specific configuration bit. Therefore, we reject bitstreams where a switch matrix multiplexer encoding contains more than one bit among the set of bits that control that particular multiplexer.

Please note that a vast amount of UltraScale+ switch matrix multiplexers are used

as pairs. Consequently, UltraScale+ multiplexers are similar to 7-Series multiplexers, with the main difference being that internal multiplexer details are made visible to the user. This organization simplifies the short-circuit detection for UltraScale+ devices as it is not necessary to determine the sets of configuration bits that control a specific multiplexing level, as performed in [22] using graph algorithms (e.g., the configuration bits $\mathtt{C}_0, \ldots, \mathtt{C}_3$ in Figure 5.1a) form a set of configuration bits).

5.1.5 Netlist bounding-box tests

Testing if a netlist is exceeding its allocated (partial) region during configuration was examined in several projects before. For example, the configuration manager for the Erlangen Slot Machine project evaluated start address information and scanned the length of the bitstream written to the device [77]. The REPLICA project parsed configuration bitstreams directly in hardware as part of the configuration controller that connects to the configuration port of the FPGA [49]. A full overview of partial reconfiguration techniques is provided in [31]. For the virus scanning implemented in this study, we use BitMan [67], which supports parsing of all Xilinx UltraScale+ FPGA configuration bitstreams. BitMan is used inside the FPGA scanner flow for bounding box testing and for converting FPGA bitstreams to the required netlist for further graph search, as illustrated in Figure 5.2.

5.1.6 Detecting wire-tapping

The wire-tapping test checks if a partial module is connected to ports that belong to the static system or another module outside the circuit boundary (i.e. the region allocated to a reconfigurable module). However, a static signal may have to cross a reconfigurable region, like, for example, in order to access a gigabit transceiver (as part of the shell), and a module placed into this region should not be allowed to access this crossing signal. We define therefore prohibited ports/nodes in the architecture graph $p^- \in G_A$ (i.e. a *negative filter*) that are not allowed to exist in the netlist G_N which is corresponding to the circuit encoded by the design to be examined: $p^- \notin G_N$. For convenience, we allow defining prohibited ports by regular expressions, which, for example, allows the definition of a bounding-box. This definition would be the same bounding-box as defined during system floorplanning when reconfigurable regions are defined (i.e. a P-block in Xilinx terminology). Because static routes may cross the area of a partial module differently in different systems, it is necessary to define the port list

individually for each system. The port list for a static route can be easily derived automatically using the `TCL` interface in Vivado using the `get_property` command on the specific signals to be protected.

5.1.7 Interface sanity check

As mentioned in the previous section, we defined a *negative filter* for a set of ports ($p^- \in G_A$). FPGA scanner can also search for port connections that must exist in a netlist, which implements a *positive filter* ($p^+ \in G_N$). This filter is, in particular, used for partially reconfigurable modules to check if the module connects to the foreseen wires between the static system and the module such that no interface wires are left over as antennas. Only such interface wires implement the communication between a partial module and the surrounding, while all other signals are strictly separated for both the surround (static) system and the partially reconfigurable module, as implemented in systems [78, 66]. An interface wire antenna may not necessarily indicate a malicious circuit but flags that a reconfigurable module may have an incompatible interface.

5.2 FPGA Scanner: Implementation and evaluation

This section is devoted to the implementation and evaluation of our FPGA virus scanner. Currently, there are two implementations of the FPGA scanner. The original one - which is known as FPGADEFENDER - is implemented entirely in Python by the PhD Student Kaspar Matas. The other implementation using TCL is implemented as a part of this thesis and can be embedded as additional checks in parallel to the Xilinx vendor DRCs. The mechanisms used in both implementations are mostly similar and the main difference is how we extract the netlist G_N from the different input formats (see Figure 5.2). Therefore, as a part of this thesis, we will describe how we could get the architecture graph G_A and the TCL implementation of the netlist generator. However, we can refer to the scanning mechanism in Figure 5.3 with the implementation details in a separate research article [110].

The existing circuits studied in Chapter 3 were used and tested on an Ultra96 board, but because the scanner operates on models that are automatically derived from the Xilinx Vivado tool suite, the approach is portable to all other Xilinx UltraScale+ FPGA platforms.

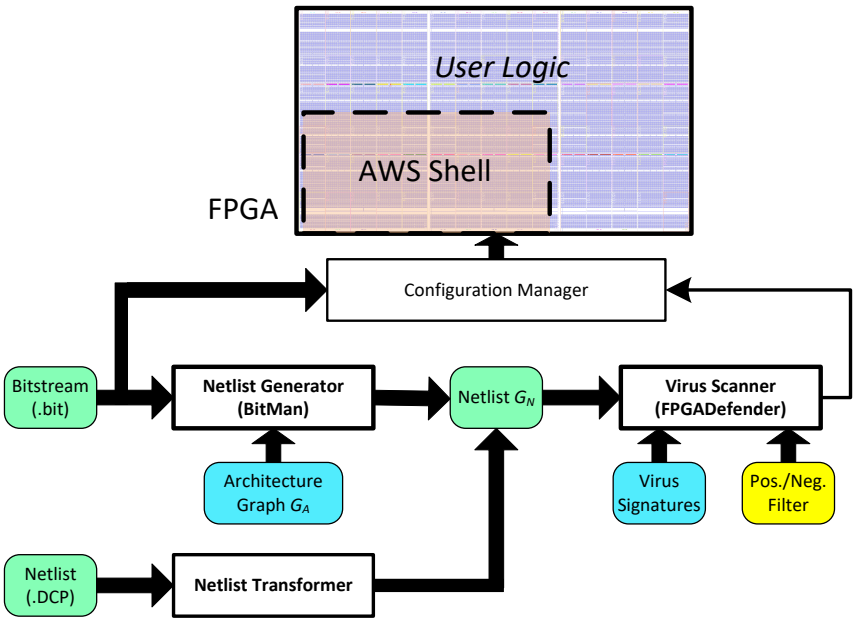


Figure 5.2: Envisioned system with a virus scanner for detecting malicious FPGA designs.

5.2.1 Architecture graph generation for FPGADefender

In order to provide the architecture graph G_A for FPGADefender, we need to decode the bitstream format using Xilinx Vivado tool and the BitMan FPGA bitstream parser (BitMan was developed by Khoa Pham in the research [67]).

The FPGA bitstream consists of the configuration commands and the configuration data [67]. The configuration data includes the actual configuration bits for every primitive and every reconfigurable routing resource. The configuration data format was studied in depth in [65]. In the study, the configuration bits are addressed by their row addresses, frame addresses, and frame offsets. Therefore, in order to find the architecture graph G_A, we need to locate the configuration bits for the FPGA resources of interest. The investigation in Chapter 3 helps us to narrow down the reverse-engineering effort so that it is not necessary to find all the configuration bits for all the FPGA resources due to the regularity of the fabric.

Firstly, we need to generate a bitstream from a blank design with the help of the Vivado tool in which we know $G_{base} \in \emptyset$. Then we need to create a bitstream with the instance of an interested resource which could be either a wire or a primitive. From these bitstreams, we can find the corresponding configuration bits which represent the directed edge $E_x \in E_A$. The configuration bits are then saved into a `JSON` dictionary. Listing 5.1 shows the snippet of the decoded configuration information for a wire in the

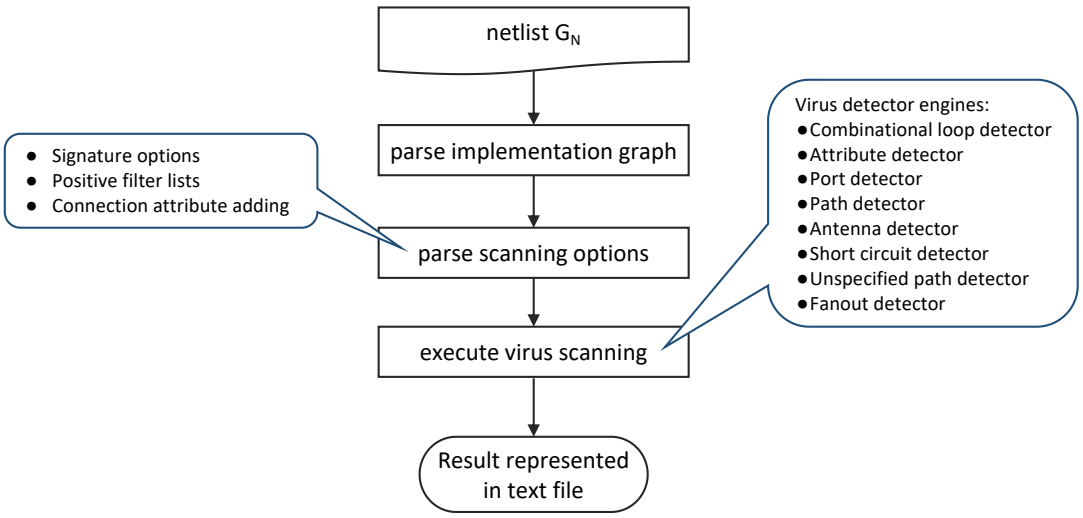


Figure 5.3: FPGA scanning flowchart.

JSON dictionary. Further, Listing 5.2 shows the snippet of the decoded configuration bit for a flip-flop which is configured as a latch. Finally, the edge information E_A can be directly extracted from *get_** TCL commands.

```
{
    "begin": "INT_NODE_SDQ_34_INT_OUT0",
    "end": "WW2_E_END6",
    "offset": 12, "frame": 54, "eo": 0, "row": 1
},
```

Listing 5.1: A snippet of the decoded configuration information for a wire.

```
 {
    "flop": "AFF",
    "mode": "LATCH",
    "offset": 18, "frame": 14, "eo": 0, "row": 0
},
```

Listing 5.2: A snippet of the decoded configuration information for a flip-flop configured as a latch.

When analysing the fracturable LUTs, we found that it is possible that a combinatorial path runs through one of the fracturable LUTs and later runs through the entire other fracturable LUT without forming a cycle or any RO (see Figure 5.4). In this example for LUT_A, the top inputs of the LUT affect only the top LUT output, and this LUT output routes through LUT_B and back to a bottom input of the same LUT_A. However, in this example the bottom output of LUT_A is only depending on the bottom

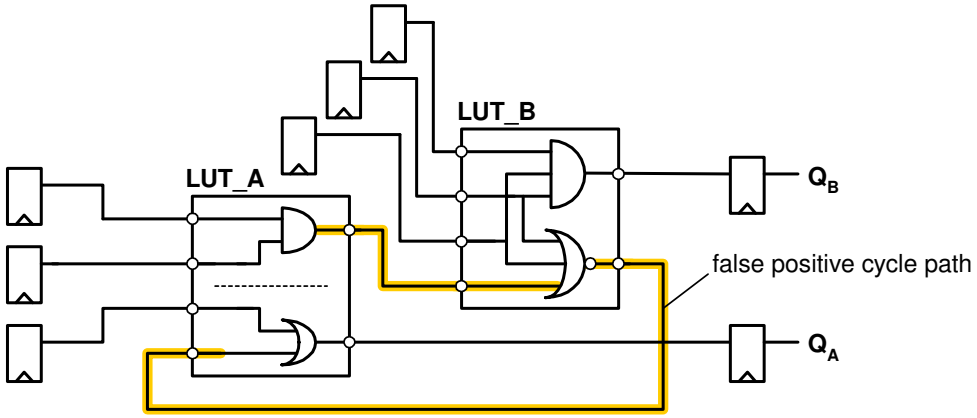


Figure 5.4: Example of a path that closes in `LUT_A` but that does not form a cycle or RO.

inputs, hence, this situation is not forming a loop. For such situations, it is not sufficient to analyse just the routing to decide if a netlist contains cycles or not. The scanner solves this problem by analysing the LUT function table (i.e. the LUT init values in the bitstream) using the Espresso logic minimiser [93] implemented in PyEDA library for Python [25]. Therefore, we also need to decode the LUT's configuration data. Similar to decoding other resources, we compare the blank bitstream to the bitstream that has a LUT initialised with different init values. The configuration for a 6-input LUT requires 64 bits of configuration data (see Listing 5.3).

```
"1": {"LUT": "A6LUT", "bit": 0, "offset": 16, "frame": 11, "eo": 0,
    "row": 0},
"2": {"LUT": "A6LUT", "bit": 1, "offset": 16, "frame": 10, "eo": 0,
    "row": 0},
"3": {"LUT": "A6LUT", "bit": 2, "offset": 16, "frame": 9, "eo": 0, "
    row": 0},
...
"64": {"LUT": "A6LUT", "bit": 63, "offset": 31, "frame": 8, "eo": 0,
     "row": 0}
```

Listing 5.3: A snippet of the decoded configuration information for a LUT init value.

With the architecture graph G_A and a bitstream input, we can generate the netlist graph accordingly. Listing 5.4 shows a snippet of the final netlist graph G_N.

```
{
  "begin": {"tile": {"name": "INT", "x": 18, "y": 19}, "name": "
    WW2_E_END6"},
  "end": {"tile": {"name": "INT", "x": 18, "y": 19}, "name": "
    INT_NODE_SDQ_34_INT_OUT0"},
  "attributes": []
},
```

Listing 5.4: A snippet of a single edge of a netlist graph.

After parsing a netlist graph G_N, scanning options are parsed to provide inputs for the virus detector engines as well as a set of positive filters $p^+ \in G_N$ and negative filters $p^- \in G_N$. Then the scanning process is executed based on a set of virus detector engines:

- **Combinatorial cycle detector:** Detect combinatorial cycles and transparent latch cycles.
- **Attribute detector:** Detect unusual synchronous design elements such as the use of latches.
- **Port detector:** Detect prohibited ports that are used in the netlist. For example, IOBUFs could be used to create combinatorial loops or listen to IO transactions.
- **Path detector:** Detect prohibited paths that are used in the netlist.
- **Antenna detector:** Detect dangling paths in the netlist (which indicate physical interface mismatches).
- **Short-circuit detector:** Detect short-circuits caused by possible bitstream manipulations (FPGADEFENDER rejects bitstreams with invalid encodings for the routing).
- **Fanout detector:** Detect and report the maximum fanout of the examined module.

5.2.2 Design evaluation

In order to test and evaluate the FPGA scanner, we developed several test cases including 1) 15 malicious designs from Table 3.2 as well as a short-circuit design; and 2) 28 reference designs including the Spector OpenCL benchmark [90], soft-core CPUs

Designs	LUT used	Comb Cycle	Latch	Short-Circuit	Fanout
Malicious Circuits					
Design 1	2000	2000	0	No	1
Design 2	2000	2000	0	No	1
Design 3	2000	2000	0	No	1
Design 4	2000	2000	0	No	1
Design 5	2000	2000	0	No	1
Design 6	2000	2000	0	No	1
Design 7	2000	4000	0	No	2
Design 8	2000	2000	0	No	10
Design 9	0	2000	0	No	1
Design 10	0	2000	0	No	1
Design 11	0	2000	0	No	1
Design 12	0	2880	0	No	1
Design 13	2000	2000	2000	No	1
Design 14 *	4000	0	0	No	3
Design 15 *	6000	0	0	No	4
Short-circuit	15974	11669	12	Yes	5223

* Designs are flagged as flip-flop clock input violation.

Table 5.3: Evaluation results for malicious designs circuits.

(MIPS and RISC-V [29]), crypto cores (AES, DES [9], and SHA3 [68]) and other peripheral circuits [85], which all do not contain any malicious circuits (see Table 5.3 for the list of malicious circuits and Table 5.4 for list of all normal test cases).

The FPGA scanner found all malicious circuits and the short-circuits in our test cases. For the experiments, each malicious circuit from *Design 1 to 15* was implemented 2000 times spread out across the FPGA. The *short-circuit* design was created directly at the bitstream level. This was implemented with the help of BitMan, which features a low-level API to access LUT values and switch matrix multiplexer configurations. To inject short-circuits, we looked for valid one-hot encoded switch matrix multiplexer configurations and randomly toggled some of the zero bits (to create randomly more hots in the multiplexer configurations). Note that the Vivado design tool does not allow to create a bitstream that contains short-circuits. This restriction means that short-circuits can be prevented if, for example, a cloud service provider generates the bitstream at the provider side rather than accepting a bitstream binary from a user.

However, this also implies that users have to share their design with the cloud

service provider (at least to some extent through design checkpoints - DCPs), which in turn is an IP protection issue. In contrast to this, FPGADEFENDER would allow a cloud service provider to directly accept FPGA configuration bitstreams while being able to detect short-circuits.

It should be noted that the FPGA vendor Xilinx does currently not provide any tool or mechanism to scan a bitstream for any malicious construct or any manipulation. The only requirement FPGADEFENDER imposes to perform its virus scanning is a plain (i.e. non-encrypted) bitstream.

So far, we tested each threat in isolation. For more rigid testing, we created designs that mix different threats and tested each time if FPGADEFENDER finds all threats correctly just by scanning the configuration bitstream. In these experiments, FPGADEFENDER correctly flagged all threats in all test cases.

We compared FPGADEFENDER combinatorial cycle detection with the Xilinx DRC checker (see Table 3.2). Here, FPGADEFENDER did not only detected correctly *Design 1 to 8* but also detected the hidden combinatorial cycles from MUX primitives in *Design 9 and 10*, an oscillator through the CLA primitive in *Design 11*, and even cycles through the DSP primitive, as in *Design 12*, where the Xilinx tool fails.

In *Design 13*, ROs are implemented through transparent latches. While Xilinx failed to flag those ROs (even if the latch enable is activated by a constant), FPGADEFENDER found all cyclic paths that run through latches.

In *Design 14 and 15*, combinatorial logic paths are used to drive the clock input of flip-flops instead of global clock sources. This will be flagged using the path detector engine in FPGADEFENDER.

As a sanity check, we used FPGADEFENDER to scan all the 28 bitstreams of the test cases that are not intentionally designed with malicious constructs (Table 5.4). FPGADEFENDER has not detected malicious constructs except for one case, the true random number generator (TRNG). The TRNG uses ring-oscillators as a source of randomness, which all got flagged by FPGADEFENDER. This case is a dilemma that could be solved by providing primitives by a system vendor (e.g., a cloud service provider) for exceptional use cases like TRNGs or PUFs (Physical Unclonable Functions). For the *True Random Number Generator* using ROs, we found exact 128 combinatorial cycles corresponding to the 128-bit random number generated.

Designs	LUT used	Comb Cycle	Latch	Short-Circuit	Fanout
Normal Circuits					
8b10b EncDec *	72	0	0	No	15
CAN Controller *	1310	0	0	No	146
BCD Adder *	68	0	0	No	6
PRNG *	237	0	0	No	107
Cordic *	1312	0	0	No	99
I2C *	307	0	0	No	87
Parallel Scrambler *	66	0	0	No	11
RS232 UART *	102	0	0	No	19
SPI *	988	0	0	No	174
Stepper Motor *	69	0	0	No	9
Breadth First Search †	604	0	0	No	204
DCT †	10085	0	16	No	418
FIR Filter †	3842	0	4	No	749
Histogram †	2409	0	0	No	217
Merge Sort †	2905	0	1	No	235
Matrix Multiplication †	8116	0	9	No	1782
Normal Estimation †	8504	0	6	No	620
Sobel Filter †	14045	0	0	No	272
SPMV †	10670	0	9	No	1552
Black-Scholes ‡	12326	0	10	No	259
RISC-V CPU ‡	3556	0	0	No	170
AES §	4520	0	0	No	162
DES §	278	0	0	No	20
Mandelbrot §	1716	0	42	No	183
MIPS CPU §	4163	0	0	No	572
SHA3 §	10662	0	0	No	262
Skin Color Detection §	2022	0	0	No	147
TRNG §	1069	128	0	No	61

* Peripheral IP designs from OpenCores [85].
† Open-source OpenCL designs from Spector benchmark [90].
‡ Other open source designs [75, 29].
§ Academic handcrafted RTL designs [9, 68].

Table 5.4: Evaluation results for benchmarking circuits.

5.2.3 TCL implementation of FPGADEFENDER

The input to FPGADEFENDER was originally a bitstream to generate the netlist graph G_N. While this provides the advantage of a standalone scanner, it requires the netlist generation from reversing the bitstream to run the scanner, which is time consuming. For that reason, it would be beneficial to have a custom DRC that reads the DCP file netlist as input and then runs the scanning engine to produce security reports in the form of Warnings and Errors [47]. From Figure 5.2, the netlist graph G_N for the scanner could be extracted from a DCP netlist using a netlist transformer which basically utilises the built-in TCL commands. The custom DRC subsequently triggers the scanning engine to give results similar to the aforementioned bitstream scanning approach. Thus, FPGADEFENDER could be integrated seamlessly into current FPGA EDA tools. However, it should be noted that the disadvantage of the current TCL implementation is that the process of extracting netlist graph G_N is considerably slower than decoding a corresponding bitstream. This is likely because the built-in TCL command interface to the vendor tools is too slow for our purpose. A TCL command normally returns an object with redundant data that consumes a large amount of memory. Without a way to flush the unused data, the tool will eventually get crashed resulting in an unfinished netlist G_N for larger problems.

5.3 Chapter summary

In this chapter, we have introduced an FPGA scanner which can perform malicious design scanning from either bitstreams or DCP netlists. The FPGA scanner can effectively detect all currently known self-oscillators which are mainly used to mount power-hammering and side-channel attacks. In addition, the scanning engine can detect short-circuits, tapping wires, the use of prohibited FPGA resources, antennas, and bounding boxes which are also common in other side-channel attacks. Moreover, the FPGA scanner could be embedded as a custom DRC into the vendor tools such as Xilinx Vivado. This provides great flexibility and practicality to be quickly adapted to current FPGA development and deployment flows on the cloud.

Chapter 6

Conclusion

This chapter summarises the contributions of this thesis. Moreover, future research directions enabled by this thesis are discussed.

6.1 Contribution summary

6.1.1 Literature review of FPGA self-oscillating circuits

An in-depth study on FPGA self-oscillators was provided in Section 3.1. The study focused on 1) finding all possible classes of self-oscillators which can be built from FPGA resources, 2) analysing self-oscillating frequencies, and 3) measuring incremental dynamic power caused by the oscillation. This work contributed to publication [110].

6.1.2 A dynamic power model for wiring resources

A dynamic power model for routing resources was introduced in Section 3.2. We have investigated and quantified the energy required for each toggle on the primary wiring resources of Xilinx UltraScale+ FPGAs. This reveals that 1) even a standard RTL design could create a fast toggling signal by creating glitches, and 2) a power-hammering attack can be deployed with a limited number of oscillating sources which fan-outs to a huge number of routing resources. The study emphasises the necessity of applying countermeasures to prevent such types of attacks on FPGA-based cloud infrastructures. Additionally, the dynamic power model can be further extended to cover other FPGA resources. This enables future studies on building a more realistic

power estimation tool that not only helps in FPGA power analysis but also detects potential power-hammering circuits.

6.1.3 Quantifying the risk of power-hammering

The Waste Power Potential (WPP) was introduced in Section 3.1 to quantify the risk of power-hammering. While power estimation indicates how much power a circuit can consume, WPP suggests if a system or an FPGA could sustainability provide enough power to a circuit. Thus, it could effectively indicate if a design is at risk of power-hammering attacks. This work was also published in [110].

6.1.4 Real-world power-hammering attack on FPGA-based infrastructure of Amazon Web Services

With the permission of Amazon Web Services, a Denial-of-Service attack was conducted successfully on their FPGA instances using the power-hammering method. The security implementation of AWS F1 instances as well as attacking techniques were studied in Chapter 4 and Appendix A. A real-world DoS attack and mitigation strategy had been published in [109].

6.1.5 A contribution to FPGADEFENDER and API for detecting malicious circuits

An FPGA architecture graph was created and described in Chapter 5. This enables the creation of FPGADEFENDER - an FPGA bitstream scanner tool that detects malicious FPGA designs. Moreover, various virus signatures had been developed for the FPGADEFENDER virus scanner engine, including the detection of self-oscillating circuits using glitch amplification, DSPs, CARRY, or cascading multiplexer primitives. This work was also published in [110].

6.1.6 A contribution to a study on a countermeasure for glitch amplification

Glitch amplification is a method to boost the switching activity for more intense power-hammering (refer to Chapter 3). However, power-hammering through glitch amplification has not been studied before. In collaboration with the PhD student Kaspar Matas,

the investigation on attacks and mitigation methods was conducted resulting in the publication [71].

6.2 Future Works

6.2.1 FPGA Power Verification

It had been proven that the Xilinx vendor tools are not reliable in providing power consumption estimation in designs which have self-oscillators [71]. Therefore, it is important to verify the power consumption of a design to make sure it is within the power budget. The method needs to take self-oscillation and glitches into account. Also, the dynamic power model for wiring resources mentioned in Section 3.2 could be used.

6.2.2 FPGA Timing Verification

The FPGA architecture model used for FPGADEFENDER could also be embedded with timing information. A timing model allows a more accurate prediction of power consumption on glitches. Also, it ensures that no flip-flop is overclocked so that it eliminates possible side-channels or PUFs.

6.3 Impact

This thesis has demonstrated that FPGAs are hugely vulnerable to power-hammering attacks and that the theoretical waste power requirement can easily reach kilowatts of waste power. Even worse, these attacks can be deployed on existing cloud infrastructures. Most importantly, this project contributed to building a virus scanner as a key defence mechanism that can circumvent the most severe effects of power-hammering to a level that other defence mechanisms (e.g., power monitoring) can actually work. We believe that such tests are essential in an FPGA-as-a-Service model and that this work is therefore enabling this model and therefore a more widespread adaptation of FPGA technology.

Bibliography

[1] A. C. Aldaya, A. J. C. Sarmiento, and S. Sanchez-Solano. AES T-Box tampering attack. *Journal of Cryptographic Engineering*, 6(1):31–48, 2016.

[2] A. L. Masle and W. Luk. Detecting power attacks on reconfigurable hardware. In *22nd International Conference on Field Programmable Logic and Applications (FPL)*, pages 14–19. IEEE, 2012.

[3] A. Moradi, A. Barenghi, T. Kasper, and C. Paar. On the vulnerability of FPGA bitstream encryption against power analysis attacks: extracting keys from xilinx Virtex-II FPGAs. In *Proceedings of the 18th ACM Conference on Computer and Communications Security*, Ccs, pages 111–124. ACM, 2011.

[4] A. Moradi and T. Schneider. Improved side-channel analysis attacks on Xilinx bitstream encryption of 5, 6, and 7 series. In *International Workshop on Constructive Side-Channel Analysis and Secure Design*, pages 71–87. Springer, 2016.

[5] A. Moradi, D. Oswald, C. Paar, and P. Swierczynski. Side-channel attacks on the bitstream encryption mechanism of Altera Stratix II: facilitating black-box analysis using software reverse-engineering. In *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, pages 91–100. ACM, 2013.

[6] A. Moradi, M. Kasper, and C. Paar. Black-box side-channel attacks highlight the importance of countermeasures: An analysis of the Xilinx Virtex-4 and Virtex-5 bitstream encryption mechanism. In *Cryptographers' Track at the RSA Conference*, volume 7178, pages 1–18, 2012.

[7] A. P. Johnson, R. Chakraborty, and D. Mukhopadhyay. A PUF-enabled secure architecture for FPGA-based IoT applications. *IEEE Transactions on Multi-Scale Computing Systems*, 1:110–122, 2015.

[8] A. Shen, A. Ghosh, S. Devadas, and K. Keutzer. On average power dissipation and random pattern testability of CMOS combinational logic networks. In *Iccad*, 1992.

[9] A. Vaishnav, J. R. G. Ordaz, and D. Koch. A security library for FPGA interlays. In *International Conference on Field Programmable Logic and Applications (FPL)*, pages 1–4, 2017.

[10] A. Vaishnav, K. D. Pham, K. Manev, and D. Koch. The FOS (FPGA Operating System) demo, 2019. https://github.com/khoapham/fos.

[11] Alibaba Cloud ECS. Deep dive into Alibaba Cloud F3 FPGA-as-a-Service instances. https://www.alibabacloud.com/blog/deep-dive-into-alibaba-cloud-f3-fpga-as-a-service-instances_594057.

[12] Amazon. AFI Power. https://github.com/aws/aws-fpga/blob/master/hdk/docs/afi_power.md.

[13] Amazon. Amazon EC2 F1 Instances. https://aws.amazon.com/ec2/instance-types/f1/.

[14] Amazon. Amazon FPGA Image (AFI) Management Tools, 2019. https://github.com/aws/aws-fpga/blob/master/sdk/userspace/fpga_mgmt_tools/README.md.

[15] Amazon Inc. AWS EC2 AFI Creation, 2020. https://github.com/aws/aws-fpga/blob/master/SDAccel/docs/Setup_AWS_CLI_and_S3_Bucket.md.

[16] Amazon Inc. AWS EC2 Instance Lifecycle, 2020. https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-instance-lifecycle.html.

[17] Amazon Inc. AWS EC2 Storage, 2020. https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/Storage.html.

[18] Avnet. Ultra96 Hardware User Guide, 2018. http://zedboard.org/sites/default/files/documentations/Ultra96-HW-User-Guide-rev-1-0-V0_9_preliminary.pdf.

[19] Avnet. Ultra96 Schematics, 2018. https://github.com/96boards/documentation/blob/master/consumer/ultra96/ultra96-v1/hardware-docs/files/ultra96-schematics.pdf.

[20] B. Ors, E. Oswald, and B. Preneel. Power-analysis attacks on an FPGA - first experimental results. In *Cryptographic Hardware and Embedded Systems - CHES*, volume 2779, pages 35–50, 2003.

[21] B. Selmke, J. Heyszl, and G. Sigl. Attack on a DFA protected AES by simultaneous laser fault injections. In *Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*, pages 36–46. IEEE, 2016.

[22] C. Beckhoff, D. Koch, and J. Torresen. Short-circuits on FPGAs caused by partial runtime reconfiguration. In *International Conference on Field Programmable Logic and Applications*, pages 596–601. IEEE, 2010.

[23] C. Beckhoff, D. Koch, and J. Torresen. Go Ahead: A partial reconfiguration framework. In *IEEE International Symposium on Field-Programmable Custom Computing Machines*, pages 37–44, 2012.

[24] C. Chiasson and V. Betz. Should FPGAs abandon the pass-gate? In *International Conference on Field programmable Logic and Applications*, pages 1–8, 2013.

[25] C. Drake. Python electronic design automation, 2018. https://pyeda.readthedocs.io/en/latest/2llm.html.

[26] C. Lattner. Clang: a C Language family frontend for LLVM, 2019. https://clang.llvm.org/.

[27] C. Paar. *Understanding cryptography: A textbook for students and practitioners*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[28] 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 *IEEE Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 45–52. IEEE, 2018.

[29] C. Wolf. PicoRV32, 2019. https://github.com/cliffordwolf/picorv32.

[30] D. Agrawal, B. Archambeault, J. R. Rao, and P. Rohatgi. The EM side-channel (s). In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 29–45. Springer, 2002.

[31] D. Koch. *Partial Reconfiguration on FPGAs: Architectures, Tools and Applications*, volume 153. Springer Science & Business Media, 2012.

[32] D. Koch, F. Hannig, and D. Ziener. *FPGAs for Software Programmers*. Springer Publishing Company, Incorporated, 1st edition, 2016.

[33] D. Lewis, E. Ahmed, G. Baeckler, V. Betz, M. Bourgeault, D. Cashman, D. Galloway, M. Hutton, C. Lane, A. Lee, P. Leventis, S. Marquardt, C. McClintock, K. Padalia, B. Pedersen, G. Powell, B. Ratchev, S. Reddy, J. Schleicher, K. Stevens, R. Yuan, R. Cliff, and J. Rose. The Stratix II logic and routing architecture. In *International Symposium on Field-Programmable Gate Arrays*, pages 14–20, New York, NY, USA, 2005. ACM.

[34] D. Meidanis, K. Georgopoulos, and I. Papaefstathiou. FPGA power consumption measurements and estimations under different implementation parameters. In *International Conference on Field-Programmable Technology*, pages 1–6, 2011.

[35] D. Merli, F. Stumpf, and C. Eckert. Improving the Quality of Ring Oscillator PUFs on FPGAs. In *Proceedings of the 5th Workshop on Embedded Systems Security*, WESS '10. Association for Computing Machinery, 2010.

[36] D. R. E. Gnad, F. Oboril, and M. B. Tahoori. Voltage drop-based fault attacks on FPGAs using valid bitstreams. In *International Conference on Field Programmable Logic and Applications (FPL)*, pages 1–7. IEEE, 2017.

[37] D. R. E. Gnad, S. Rapp, J. Krautter, and M. B. Tahoori. Checking for electrical level security threats in bitstreams for multi-tenant FPGAs. *International Conference on Field-Programmable Technology (FPT)*, 2018.

[38] D. Ziener, and J. Pirkl, and J. Teich. Configuration Tampering of BRAM-based AES Implementations on FPGAs. In *2018 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, pages 1–7, 2018.

[39] D. Ziener, S. Assmus, and J. Teich. Identifying FPGA IP-cores based on lookup table content analysis. In *International Conference on Field Programmable Logic and Applications*, pages 1–6. IEEE, 2006-08.

[40] E. Biham and A. Shamir. Differential fault analysis of secret key cryptosystems. In *Annual international cryptology conference*, pages 513–525. Springer, 1997.

[41] E. De Mulder, P. Buysschaert, S. B. Ore, P. Delmotte, B. Preneel, G. Vandenbosch, and I. Verbauwhede. Electromagnetic analysis attack on an FPGA implementation of an elliptic curve cryptosystem. In *The International Conference on Computer as a Tool*, volume 2, pages 1879–1883, 2005.

[42] Evans, J.D. *Straightforward Statistics for the Behavioral Sciences*. Brooks/Cole Publishing Company, 1996.

[43] F. Benz, A. Seffrin, and S. A. Huss. Bil: A tool-chain for bitstream reverse-engineering. In *International Conference on Field Programmable Logic and Applications (FPL)*, pages 735–738. IEEE, 2012.

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

[45] F. X. Standaert, F. Mace, E. Peeters, and J. J. Quisquater. Updates on the security of FPGAs against power analysis attacks. In *Reconfigurable Computing: Architectures and Applications*, volume 3985, pages 335–346. Springer Verlag, 2006.

[46] F. X. Standaert, S. B. Ors, J. J. Quisquater, and B. Preneel. Power analysis attacks against FPGA implementations of the DES. In Jürgen Becker, Marco Platzner, and Serge Vernalde, editors, *Field Programmable Logic and Application*, pages 84–94, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[47] FPGA Researchers at The University of Manchester. FPGADefender Github Repository, 2020. `https://github.com/FPGA-Research-Manchester`.

[48] G. Provelengios, D. Holcomb, and R. Tessier. Power distribution attacks in multitenant FPGAs. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 28(12):2685–2698, 2020.

[49] H. Kalte, G. Lee, M. Porrmann, and U. Ruckert. REPLICA: A bitstream manipulation filter for module relocation in partial reconfigurable systems. In *IEEE International Parallel and Distributed Processing Symposium*, 2005.

[50] H. Lohrke, S. Tajik, T. Krachenfels, C. Boit, and J. P. Seifert. Key extraction using thermal laser stimulation. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 573–595, 2018.

[51] I. Giechaskiel, K. B. Rasmussen, and K. Eguro. Leaky wires: Information leakage and covert communication between FPGA long wires. In *Asia Conference on Computer and Communications Security*, pages 15–27. ACM, 2018.

[52] I. Giechaskiel, K. Rasmussen, and J. Szefer. Measuring long wire leakage with ring oscillators in cloud FPGAs. In *Proceedings of the International Conference on Field-Programmable Logic and Applications*, Fpl, 2019.

[53] I. Hadzic, S. Udani, and J. M. Smith. FPGA viruses. In *International Workshop on Field Programmable Logic and Applications*, pages 291–300. Springer, 1999.

[54] I. Vasyltsov, E. Hambardzumyan, Y. S. Kim, and B. Karpinskyy. Fast Digital TRNG Based on Metastable Ring Oscillator. In *Cryptographic Hardware and Embedded Systems – CHES 2008*, pages 164–180, 2008.

[55] I. Verbauwhede, D. Karaklajic, and J. Schmidt. The fault attack jungle - a classification model to guide you. In *Workshop on Fault Diagnosis and Tolerance in Cryptography*, pages 3–8. IEEE, 2011.

[56] Intel. Stop and Start Your Instance, 2020. `https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/Stop_Start.html`.

[57] J. Acle, J. Oliver, and E. Boemo. Power estimations vs. power measurements in Spartan-6 devices. In *Southern Conference on Programmable Logic (SPL)*, 2014.

[58] J. B. Note and E. Rannaud. From the bitstream to the netlist. In *ACM/SIGDA Symposium on Field Programmable Gate Arrays*, FPGA, page 264. ACM, 2008.

[59] J. Barr. Developer Preview - EC2 Instances (F1) with Programmable Hardware, 2016. `https://aws.amazon.com/blogs/aws/developer-preview-ec2-instances-f1-with-programmable-hardware/`.

[60] J. Brouchier, T. Kean, C. Marsh, and D. Naccache. Temperature attacks. *IEEE Security & Privacy*, 7(2):79–82, 2009.

[61] J. Burgiel, D. Esguerra, I. Giechaskiel, S. Tian, and J. Szefer. Characterization of IOBUF-based ring oscillators. In *International Conference on Field-Programmable Technology (ICFPT)*, pages 1–4, 2021.

[62] J. Delvaux, R. Peeters, D. Gu, and I. Verbauwhede. A survey on lightweight entity authentication with strong PUFs. *ACM Comput. Surv.*, 48(2), 2015.

[63] J. Krautter, D. R. E. 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*, pages 44–68, 2018.

[64] J. Krautter, D. R. E. Gnad, and M. B. Tahoori. Mitigating electrical-level attacks towards secure multi-tenant FPGAs in the cloud. *ACM Trans. Reconfigurable Technol. Syst.*, 12(3):12:1–12:26, 2019.

[65] K. D. Pham. *FPGA virtualisation on heterogeneous computing systems: Model, tools, and systems*. University of Manchester, 2020.

[66] K. D. Pham, A. Vaishnav, M. Vesper, and D. Koch. ZUCL: A ZYNQ UltraScale+ framework for OpenCL HLS applications. In *International Workshop on FPGAs for Software Programmers (FSP)*, 2018.

[67] K. D. Pham, E. Horta, and D. Koch. BITMAN: A tool and API for FPGA bitstream manipulations. In *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 894–897. IEEE, 2017.

[68] K. D. Pham, E. Horta, D. Koch, A. Vaishnav, and T. Kuhn. IPRDF: An isolated partial reconfiguration design flow for Xilinx FPGAs. In *International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC)*, pages 36–43, 2018.

[69] K. Georgopoulos, K. Bakanov, I. Mavroidis, I. Papaefstathiou, A. Ioannou, P. Malakonakis, K. D. Pham, D. Koch, and L. Lavagno. *A novel framework for utilising multi-FPGAs in HPC systems*, pages 153–189. Taylor & Francis, 2019.

[70] K. M. Zick, M. Srivastav, W. Zhang, and M. French. Sensing nanosecond-scale voltage attacks and natural transients in FPGAs. In *ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, pages 101–104. ACM, 2013.

[71] K. Matas, T. M. La, K. D. Pham, and D. Koch. Power-hammering through glitch amplification â attacks and mitigation. In *International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 65–69, 2020.

[72] K. Rasmussen, I. Giechaskiel, and K. Eguro. Leakier wires: Exploiting FPGA long wires for covert and side-channel attacks. *ACM Transactions on Reconfigurable Technology and Systems*, 2019.

[73] K. Zick and J. Hayes. Low-cost sensing with ring oscillator arrays for healthier reconfigurable systems. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, 5(1):1–26, 2012.

[74] Kaspersky. Social engineering in 2021, 2021. `https://go.kaspersky.com/rs/802-IJN-240/images/NJ_Social_Engineering_KFP.pdf`.

[75] L. Ma, F. B. Muslim, and L. Lavagno. High performance and low power Monte Carlo methods to option pricing models via high level design and synthesis. In *Ems*, pages 157–162, 2016.

[76] M. Joye and M. Tunstall. *Fault analysis in cryptography*. Information Security and Cryptography. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[77] M. Majer, J. Teich, A. Ahmadinia, and C. Bobda. The Erlangen slot machine: A dynamically reconfigurable FPGA-based computer. *J. VLSI Signal Process. Syst.*, 47(1):15–31, 2007.

[78] M. Vesper, D. Koch, and K. D. Pham. PCIeHLS: an OpenCL HLS framework. In *Fourth International Workshop on FPGAs for Software Programmers*, pages 1–6, 2017.

[79] M. Yamaguchi, H. Toriduka, S. Kobayashi, T. Sugawara, N. Hommaa, A. Satoh, T. Aoki. Development of an on-chip micro shielded-loop probe to evaluate performance of magnetic film to protect a cryptographic LSI from electromagnetic analysis. In *International Symposium on Electromagnetic Compatibility*, pages 103–108, 2010.

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

[81] National Cyber Security Centre. What is cyber security? https://www.ncsc.gov.uk/section/about-ncsc/what-is-cyber-security.

[82] National Institute of Standard and Technology. What is system integrity? https://csrc.nist.gov/glossary/term/system_integrity.

[83] Nimbix Inc. Xilinx Alveo Accelerator Cards, 2020. https://www.nimbix.net/alveo.

[84] O. Kommerling and M. G. Kuhn. Design principles for tamper-resistant smartcard processors. *Smartcard*, 99:9–20, 1999.

[85] OpenCores. Free and open source gateware IP cores, 2020. https://opencores.org/.

[86] P. Kocher, J. Jaffe, and B. Jun. Differential power analysis. In *Annual International Cryptology Conference*, pages 388–397. Springer, 1999.

[87] P. Swierczynski, A. Moradi, D. Oswald, and C. Paar. Physical security evaluation of the bitstream encryption mechanism of Altera Stratix II and Stratix III FPGAs. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, 7(4):1–23, 2014.

[88] P. Swierczynski, G. T. Becker, A. Moradi, and C. Paar. Bitstream Fault Injections (BiFI)-automated fault attacks against SRAM-based FPGAs. *IEEE Transactions on Computers*, 67(3):348–360, 2018.

[89] P. Swierczynski, M. Fyrbiak, P. Koppe, and C. Paar. FPGA trojans through detecting and weakening of cryptographic primitives. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 34(8):1236–1249, 2015.

[90] Q. Gautier, A. Althoff, P. Meng, and R. Kastner. Spector: An OpenCL FPGA benchmark suite. In *International Conference on Field-Programmable Technology (FPT)*, 2016.

[91] R. Amerson, R. Carter, W. Culbertson, P. Kuekes, G. Snider, and L. Albertson. Plasma: An FPGA for million gate systems. In *International ACM Symposium on Field-Programmable Gate Arrays*, pages 10–16, 1996.

[92] R. Amerson, R. J. Carter, W. B. Culbertson, P. Kuekes, and G. Snider. Teramac-Configurable custom computing. In *Proceedings IEEE Symposium on FPGAs for Custom Computing Machines*, pages 32–38, 1995.

[93] R. K. Brayton, G. D. Hachtel, C. McMullen, and A. Sangiovanni-Vincentelli. *Logic minimization algorithms for VLSI synthesis*, volume 2. Springer Science & Business Media, 1984.

[94] R. Watanabe, S. Ura, Q. Zhao, and T. Yoshida. Implementation of FPGA building platform as a cloud service. In *HEART*, 2019.

[95] S. Drimer. Volatile FPGA design security–a survey. *IEEE Computer Society Annual Volume*, pages 292–297, 2008.

[96] S. M. Trimberger and J. J. Moore. FPGA Security: Motivations, features, and applications. *Proceedings of the IEEE*, 102(8):1248–1265, 2014-08.

[97] S. Mangard. *Power analysis attacks : Revealing the secrets of smart cards*. Springer US, Boston, MA, 2007.

[98] S. McNeil. Solving today's design security concerns. *Xilinx Corporation*, 2010.

[99] S. S. Mirzargar and M. Stojilovic. Physical side-channel attacks and covert communication on FPGAs: A survey. In *International Conference on Field-Programmable Logic and Applications*, FPL, 2019.

[100] S. Tajik, F. Ganji, J. P. Seifert, H. Lohrke, and C. Boit. Laser fault attack on Physically Unclonable Functions. In *Fault Diagnosis and Tolerance in Cryptography (FDTC)*, pages 85–96. IEEE, 2015.

[101] S. Tajik, H. Lohrke, J. P. Seifert, and C. Boit. On the power of optical contactless probing: Attacking bitstream encryption of FPGAs. In *Conference on Computer and Communications Security*, pages 1661–1674. Association for Computing Machinery, 2017.

[102] S. Tian and J. Szefer. Temporal thermal covert channels in cloud FPGAs. In *International Symposium on Field-Programmable Gate Arrays*, pages 298–303. ACM, 2019.

[103] S. Tian, W. Xiong, I. Giechaskiel, K. Rasmussen, and J. Szefer. Fingerprinting cloud FPGA infrastructures. In *International Symposium on Field-Programmable Gate Arrays*, Fpga '20, pages 58–64, New York, NY, USA, 2020. Association for Computing Machinery.

[104] S. Trimberger and S. McNeil. Security of FPGAs in data centers. In *International Verification and Security Workshop (IVSW)*, pages 117–122. IEEE, 2017.

[105] S. Wilton. A crosstalk-aware timing-driven router for FPGAs. In *International Symposium on Field Programmable Gate Arrays*, pages 21–28. ACM, 2001.

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

[107] SymbiFlow. Project X-Ray, 2019. `https://github.com/SymbiFlow/prjxray`.

[108] T. Iakymchuk, M. Nikodem, and K. Kepa. Temperature-based covert channel in FPGA systems. In *Reconfigurable Communication-Centric Systems-on-Chip (ReCoSoC)*, pages 1–7. IEEE, 2011.

[109] T. M. La, K. D. Pham, J. Powell, and D. Koch. Denial-of-Service on FPGA-based cloud infrastructures – attack and defense. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2021(3):441–464, 2021.

[110] T. M. La, K. Matas, N. Grunchevski, K. D. Pham, and D. Koch. FPGADefender: Malicious self-oscillator scanning for Xilinx UltraScale+ FPGAs. *ACM TRETS*, 2020.

[111] T. S. Messerges, E. A. Dabbish, and R. H. Sloan. Power analysis attacks of modular exponentiation in smartcards. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 144–157. Springer, 1999.

[112] 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*, 55(11):640–642, 2019.

[113] T. Wollinger, J. Guajardo, and C. Paar. Security on FPGAs: State-of-the-art implementations and attacks. *ACM Transactions on Embedded Computing Systems (TECS)*, 3(3):534–574, 2004.

[114] T. Zhang, J. Wang, S. Guo, and Z. Chen. A comprehensive FPGA reverse engineering tool-chain: From bitstream to RTL code. *IEEE Access*, 7:38379–38389, 2019.

[115] V. Taraate. *Advanced HDL synthesis and SOC prototyping*. Springer US, 2019.

[116] Xilinx. Configuration Issues: Power-up, Volatility, Security, Battery Back-up, 1997. https://www.xilinx.com/support/documentation/application_notes/xapp092.pdf.

[117] Xilinx. Extended Spartan-3A Family Overview, 2011. https://docs.xilinx.com/v/u/en-US/ds706.

[118] Xilinx. The Xilinx SDAccel Development Environment - Bringing The Best Performance/Watt to the Data Center, 2014. https://www.xilinx.com/support/documentation/backgrounders/sdaccel-backgrounder.pdf.

[119] Xilinx. Partial Reconfiguration, 2019. https://www.xilinx.com/support/documentation/sw_manuals/xilinx2019_1/ug947-vivado-partial-reconfiguration-tutorial.pdf.

[120] Xilinx. UltraScale Architecture System Monitor, 2019. https://www.xilinx.com/support/documentation/user_guides/ug580-ultrascale-sysmon.pdf.

[121] Xilinx. Vivado User Guide: Programming and Debugging, 2019. https://www.xilinx.com/support/documentation/sw_manuals/xilinx2019_1/ug908-vivado-programming-debugging.pdf.

[122] Xilinx. Xilinx Power Analysis and Optimization, 2019. https://www.xilinx.com/support/documentation/sw_manuals/xilinx2019_2/ug907-vivado-power-analysis-optimization.pdf.

[123] Xilinx. UltraScale Architecture Clocking Resources, 2020. https://docs.xilinx.com/v/u/en-US/ug572-ultrascale-clocking.

[124] Xilinx. UltraScale Architecture Configuration, 2020. https://www.xilinx.com/support/documentation/user_guides/ug570-ultrascale-configuration.pdf.

[125] Xilinx. Vivado Design Suite Tcl Command Reference Guide (UG835), 2021. https://docs.xilinx.com/r/2021.2-English/ug835-vivado-tcl-commands/report_drc.

[126] Xilinx. UltraScale Architecture and Product Data Sheet: Overview, 2022. https://docs.xilinx.com/v/u/en-US/ds890-ultrascale-overview.

[127] Xilinx Inc. Vivado 2018.02, 2018. https://www.xilinx.com/products/design-tools/vivado.html.

[128] Xilinx Inc. Delivering a Generation Ahead at 20nm and 16nm, 2019. https://www.xilinx.com/about/generation-ahead-16nm.html.

[129] Xilinx Inc. UltraScale Architecture DSP Slice, 2019. https://www.xilinx.com/support/documentation/user_guides/ug579-ultrascale-dsp.pdf.

[130] Xilinx Inc. UltraScale Architecture Memory Resources, 2019. https://www.xilinx.com/support/documentation/user_guides/ug573-ultrascale-memory-resources.pdf.

[131] Xilinx Inc. Zynq UltraScale+ MPSoC Data Sheet: DC and AC Switching Characteristics, 2019. https://www.xilinx.com/support/documentation/data_sheets/ds925-zynq-ultrascale-plus.pdf.

[132] Y. Kim, R. Daly, J. Kim, C. Fallin, J. H. Lee, D. Lee, C. Wilkerson, K. Lai, and O. Mutlu. Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors. In *International Symposium on Computer Architecture (ISCA)*, pages 361–372. Institute of Electrical and Electronics Engineers Inc., 2014.

[133] Y. Nasser and J. Lorandel and J. C. Prevotet and M. Helard. RTL to transistor level power modeling and estimation techniques for FPGA and ASIC: A survey. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 40(3):479–493, 2021.

Appendix A

FPGA Technology and Implementation of FPGA designs

This section provides backgrounds on FPGA technology, the tool flow to implement user circuits on FPGAs, and the basics required to register and run user FPGA applications on AWS with respect to FPGA security vulnerabilities related to AWS F1 instances.

A.1 FPGA technology

As illustrated in Figure A.1, an FPGA consists of a regular fabric with configurable logic blocks (CLBs) connected by routing channels. The CLBs provide a switch matrix with programmable multiplexers for setting connections (for implementing the routing of a user circuit running on an FPGA). Inside a CLB, there are one or more look-up

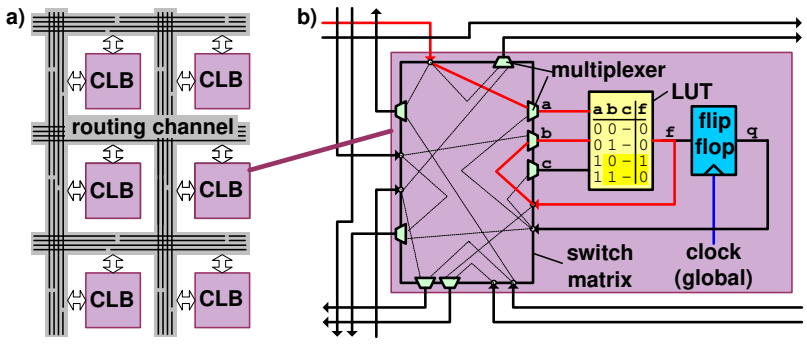


Figure A.1: a) Illustration of an FPGA fabric with configurable logic blocks (CLBs) and routing channels, b) CLB details. The red path in b) shows a controllable ring-oscillator.

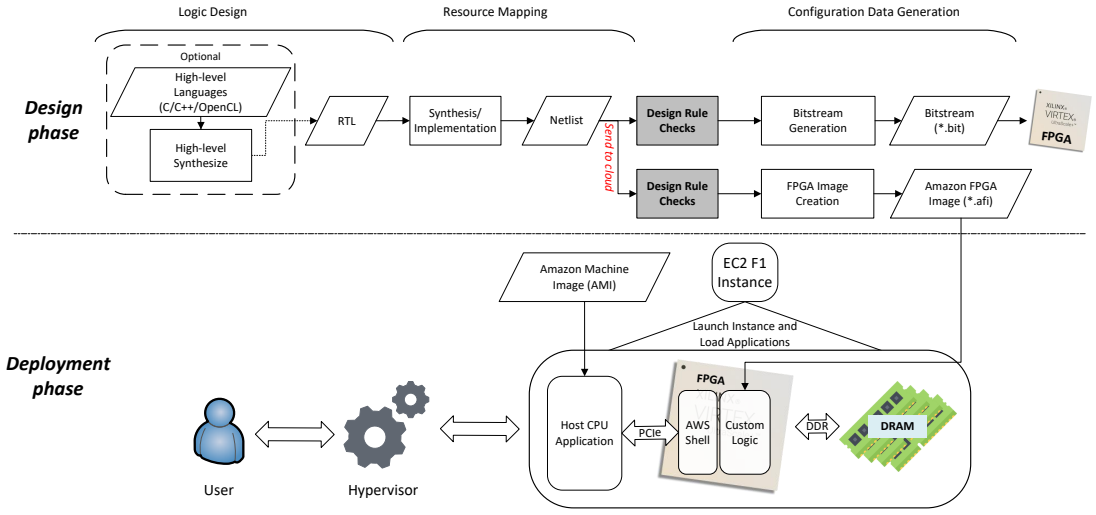


Figure A.2: FPGA development for AWS F1 instances.

tables (LUTs) to implement Boolean functions as a truth table. The LUTs are small memories that are written when configuring the FPGA (along with the multiplexer configuration information) and read when using the LUT as a Boolean function generator. Therefore, a LUT can implement any Boolean function limited only by the size of the LUT. Data centre FPGAs commonly use 6-input LUTs that can alternatively be used as two independent 5-input LUTs with shared inputs. Typically, each LUT output can be passed through a flip-flop, which stores the states of a digital circuit.

The red path in Figure A.1b) shows a ring-oscillator. If input `a` of the LUT is `'1'` then output `f` = *NOT* `b`, otherwise the output is `'0'` (here input `c` is unused `'-'`). Such oscillators can run at a few GHz with a corresponding power footprint. Because of the programmable routing, FPGAs are slower than dedicated ASICs where routing is carried out through direct metal wire connections without any switching on the paths. Therefore, FPGA designs usually run at a few hundred MHz and are certainly much slower than the previously described ring-oscillator. Because data centre FPGAs provide over a million LUTs, and by driving the fast switching oscillator signal to the routing wires, dynamic power demand could scale to over a kilowatt. However, this demand is a theoretical value because no system could deliver or sustain such power levels.

We use the term power-hammering potential P to express this theoretical value. User designs with a large P may break down the power supply and can leave the safe operational supply voltage margins. Power-hammering creates voltage drops in the FPGA itself but also in neighbouring pieces of equipment. This is a possible risk if an

FPGA board under attack shares some hardware infrastructure, like power supplies or cooling facilities.

Because the speed of a ring-oscillator depends on the supply voltage and device temperature, such circuit can measure voltage fluctuations (e.g., for power analysis attacks) or the device's temperature. Such sensors can be implemented in cloud FPGAs. Moreover, ring-oscillator frequencies may vary at different positions inside an FPGA and across different FPGAs due to process variations, which we used for FPGA fingerprinting.

FPGAs provide further blocks, including I/O blocks (for the communication to the outside world), memory blocks (to provide small on-chip caches/memories), and arithmetic blocks (called DSPs). Typically, a cloud FPGA configuration provides some basic infrastructure (commonly called a *shell*) that is in charge of all off-chip communication. User logic will only connect to ports provided by the shell, but never directly to I/O (such as PCIe). A cloud user cannot access to any configuration port. Therefore, users will only use logic cells, memory blocks, and DSPs that are not occupied by the shell. This is enforced and verified by the FPGA design tools.

The CLBs of data centre FPGAs commonly provide dedicated carry logic to implement fast adders and counters. Other features in CLBs include multiplexers to build larger function generators from a set of adjacent LUTs. Consequently, there are many possibilities to implement ring-oscillators and circuits that can draw excessive power.

A.2 Implementation of FPGA designs

This paragraph describes the design process for FPGA designs all the way to a configuration bitstream that can be loaded onto an FPGA for acceleration. The basic flow is shown in Figure A.2 and includes three major stages:

1. **Logic Design**: Here, users specify the hardware functionality of the FPGA (using Hardware Description Languages (HDL) like Verilog or VHDL or a high-level programming language such as C, C++, OpenCL).

2. **Resource Mapping**: At this step, the logic design is synthesized into Boolean logic functions, which are mapped into the available FPGA primitives (e.g., the LUTs and DSP blocks). The result forms a graph called *netlist* where the nodes represent the primitives and the edges model connections. This graph is then mapped onto the physical FPGA by placing the primitives and computing the

routing (i.e. the multiplexers settings, as shown in Figure A.1). The result of this process is again a netlist that includes the placement and routing information.

3. **Configuration Data (Bitstream) Generation**: This step generates the configuration binary to be used for programming the FPGA. There exists a one-to-one correspondence between the netlist and the bitstream.

The top row in Figure A.2 shows a path where the entire tool-chain is executed at the user-side. Alternatively, AWS provides cloud instances for running the flow. For executing a design on AWS F1 instances, users have to provide a netlist to AWS for the final configuration data generation.

A.3 Registering of user designs on AWS

AWS does not allow clients to upload their own bitstreams and instead requires a netlist. During a *registering* phase, a user netlist is translated into a configuration bitstream. As can be seen in Figure A.2, registering includes the steps Design Rule Checking (DRC) and FPGA Image Generation. The latter process generates a configuration bitstream and some metadata (e.g., the shell version), which is packaged into an *Amazon FPGA Image (AFI)* [15]. Note that the netlist provides user design details, which force clients to share their IP with AWS. Users can upload their own netlist in an encrypted format, but that is only obfuscating a netlist, as shown in Section 4.1.

All user bitstreams are generated by AWS to guarantee the compatibility with the shell, and, most importantly, to perform checks (see Section 4.1) to ensure that the bitstream is not damaging the FPGA hardware (e.g., through short-circuits [22, 92, 91, 53]). A more comprehensive overview of FPGA security aspects is discussed in Section 4.1. Once an AFI-image is created, it will be stored by AWS.

A.4 Deployment of user designs on AWS

Before running an AFI on a cloud FPGA, a user needs to create an F1 instance and accesses it through a hypervisor to program the FPGA and load the software application stack. AWS F1 instances use Elastic Block Store (EBS) instances which mount external disks for storage [17]. Therefore, an EBS-backed instance can preserve user data after instance termination or stop. We use this feature to log system states even in the case an attack may result in a connection drop to the client.

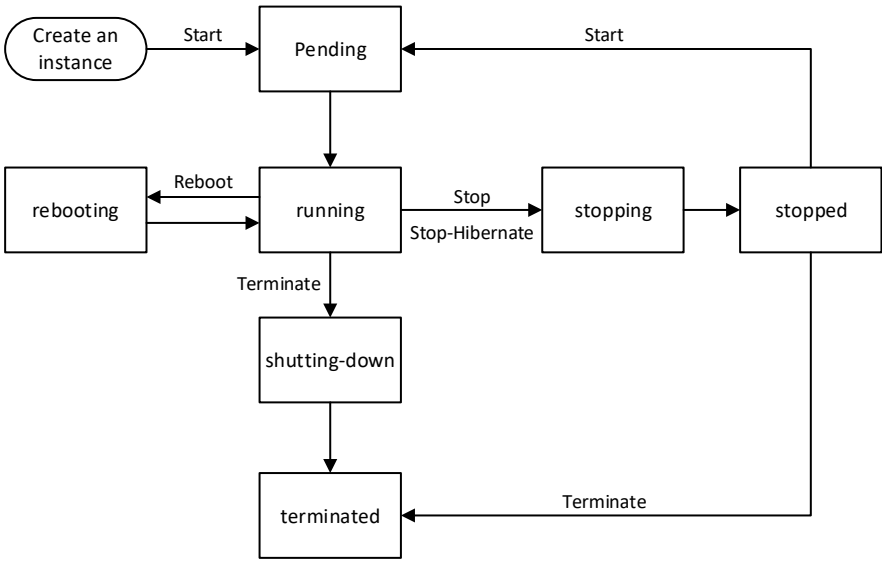


Figure A.3: Lifecycle of an Amazon EBS-backed EC2 instance. This figure is adopted from [16].

Throughout its lifecycle, an AWS instance will go through several states, as shown in Figure A.3. In the *pending* state, the hypervisor finds an instance and boots the machine. After the instance is ready, it enters the state *running* where a user can load the AFI file with the user configuration.

Users are only billed when the instance is in the running state or preparing to hibernate. Other states are not billed. This is of interest for mounting denial-of-service attacks where the goal is to use minimum cost to maximize the time an instance is occupied without billing.