Denial-of-Service on FPGA-based Cloud Infrastructure - Attack and Defense

DOI:
10.46586/tches.v2021.i3.441-464

Document Version
Accepted author manuscript

Link to publication record in Manchester Research Explorer

Citation for published version (APA):
La, T., Pham, K., Powell, J., & Koch, D. (2021). Denial-of-Service on FPGA-based Cloud Infrastructure - Attack and Defense. *IACR Transactions on Cryptographic Hardware and Embedded Systems (TCHES)*, *2021*(3), 441-464. https://doi.org/10.46586/tches.v2021.i3.441-464

Published in:
IACR Transactions on Cryptographic Hardware and Embedded Systems (TCHES)

Denial-of-Service on FPGA-based Cloud Infrastructures
—
Attack and Defense

Tuan La, Khoa Pham, Joseph Powell and Dirk Koch

The University of Manchester, UK
tuan.la@postgrad.manchester.ac.uk
[khoa.pham,joseph.powell,dirk.koch]@manchester.ac.uk

Abstract. This paper presents attacks targeting the FPGAs of AWS F1 instances at the electrical level through power-hammering, where excessive dynamic power is used to crash FPGA instances. We demonstrate different power-hammering attacks that pass all AWS security fences implemented on F1 instances, including the FPGA vendor design rule checks. In addition, we fingerprint the FPGA instances to observe the responsiveness of the instances, which indicates a successful denial-of-service attack. Most importantly, we provide an FPGA virus scanner framework, which was improved to support large datacenter FPGAs for preventing such attacks, including virtually all currently demonstrated side-channel attacks. Our experiments showed that an AWS F1 instance crashes immediately by starting an FPGA design demanding 369W. By using FPGA-fingerprinting, we found that crashed instances are unavailable for about one to over 200 hours.

Keywords: IACR Transactions · TCHES · FPGAs · Power-Hammering · DoS attack

1 Introduction

The flexibility and power efficiency of FPGAs is increasingly used for cloud computing. For instance, [Inc14] reports that FPGA acceleration can achieve 25× better performance per watt and $50-70\times$ latency improvement compared to CPU/GPU implementations. Major shifts towards FPGA-enabled Cloud Services include the Microsoft project Catapult for accelerating their Bing search engine [PCC+14], and the introduction of Amazon Web Service F1 FPGA instances [Ama19b] in 2012 and 2016, respectively.

However, opposed to using FPGAs as components in systems where only one party defines the configuration of the FPGA, integrating FPGAs into a cloud uses two designs: 1) a shell (i.e. the infrastructure design) provided by the cloud service provider (CSP) or FPGA vendor and 2) the user design providing the acceleration functionality. The latter allows users to mount attacks in their design to gather sensitive data of potentially both the CSP and other tenants [Jak20]. This is possible due to the low-level programmability of FPGAs. This includes 1) designs containing short-circuits or thermal hotspots, which can induce faults or damage a chip [HUS99, BKT08] and 2) soft-logic sensors to enable remote side-channel attacks [MS19b].

To prevent this, CSPs implemented several fences to protect the FPGA equipment and to ensure stable and secure operation across all clients. As will be discussed in detail in

Section 4.2, these fences include: 1) inspecting FPGA designs for potentially malicious circuits, 2) generation of configuration bitstreams by the CSP, 3) a low-level API that blocks direct access to the FPGA configuration port, and 4) runtime monitoring of the FPGA.

We will show that fence 1) is insufficient to detect different classes of malicious circuits and that fence 4), can be bypassed. By bypassing fence 1), we remove fences 2) and 3) because we can generate and ultimately deploy malicious bitstreams on (AWS) cloud FPGAs. By bypassing fence 4), harm caused by malicious designs may propagate to other parts of a system (e.g., through voltage fluctuations on a shared power supply).

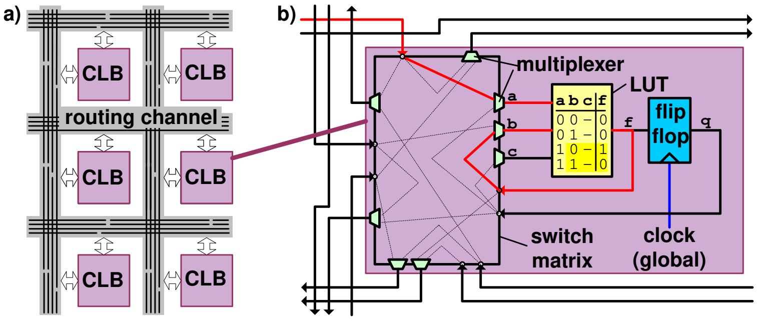
We will discuss and demonstrate multiple scenarios of denial-of-service attacks (DoS) that had been deployed on FPGA instances. To verify whether the launched attacks were successful and to measure the impact on the availability of AWS F1 cloud instances, we implemented a fingerprinting technique (in Section 8). The proposed fingerprinting technique can serve as a convert channel to leak data across different FPGA boards, which we demonstrated by measuring device temperatures. Additionally, user designs can measure power fluctuations with high accuracy, which allows further side-channel analysis. This could be used to reverse engineer CSP scheduling policies and physical information about the cloud infrastructure. To mitigate these attacks, we proposed fixes to the existing fences and added a fence that analyzes the FPGA bitstream before it is used to configure an FPGA. For the latter, we reimplemented the FPGADefender Virus scanner [LMG+20] to scale to large datacenter FPGAs. The main contributions of this paper include:

1. An evaluation of different (malicious) power-hammering designs with respect to an attack on FPGA-based cloud computing instances (in Section 4.3);
2. Using of the FPGA-fabric as a sensor for temperature monitoring and for fingerprinting (through physical unclonable functions, PUFs) on AWS F1 instances (in Section 4.5);
3. A denial-of-service (DoS) demonstration on AWS F1 FPGA cloud instances (in Section 4.6);
4. Mitigation strategies, including custom design rule checks (DRCs) for the vendor tool flow and a reimplementation if the FPGADefender FPGA virus scanner [LMG+20] (in Section 5).

Furthermore, we provide backgrounds of FPGA technology, FPGA development and deployment on AWS in Section 2, a literature review on current attacks and countermeasures on FPGA-based systems in Section 3, a discussion in Section 6, and a conclusion in Section 7.

2 Backgrounds

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. This paper focuses on attacking AWS F1 because these instances are widely used and had been among the first public offerings. The CSPs Alibaba, Huawei, 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 than 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 [PHT20].



a	b	c	f
0	0	–	0
0	1	–	0
1	0	–	1
1	1	–	0

Figure 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.

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 of configurations, as shown in [JCM15]. Such approaches are useful for closed systems, but cannot be used by a CSP who wants to provide an easy usable service to a large customer base. Therefore, CSPs are using a security infrastructure that inspects user designs before FPGA deployment and that monitor the FPGA at runtime, as presented in Section 4.

2.1 FPGA Technology

As illustrated in Figure 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 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. Datacenter 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 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 `'-'`). As we will show in Section 4.3, 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, FPGAs designs usually run at a few hundred MHz and certainly much slower than the previously described ring-oscillator. Because datacenter 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 neighboring 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

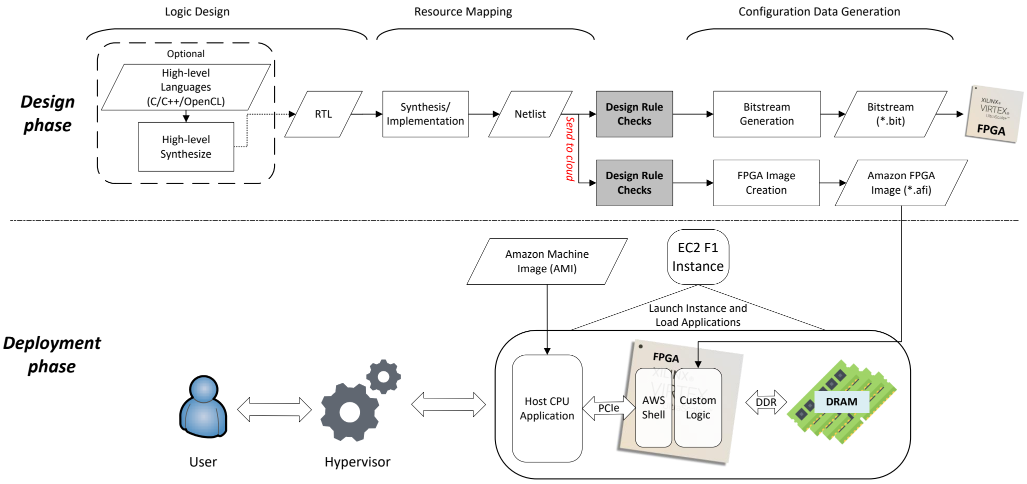


Figure 2: FPGA development for AWS F1 instances.

different FPGAs due to process variations, which we will use 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. Threrefore, 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 datacenter 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.

2.2 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 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 1). The result of this process is again a netlist that includes the placement and routing information. For the rest of this paper, we use the term *netlist* to refer to this fully implemented netlist.

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 2 shows a path where the entire tool-chain is executed at user-side. Alternatively, AWS provides cloud instances for running the flow. For executing a design

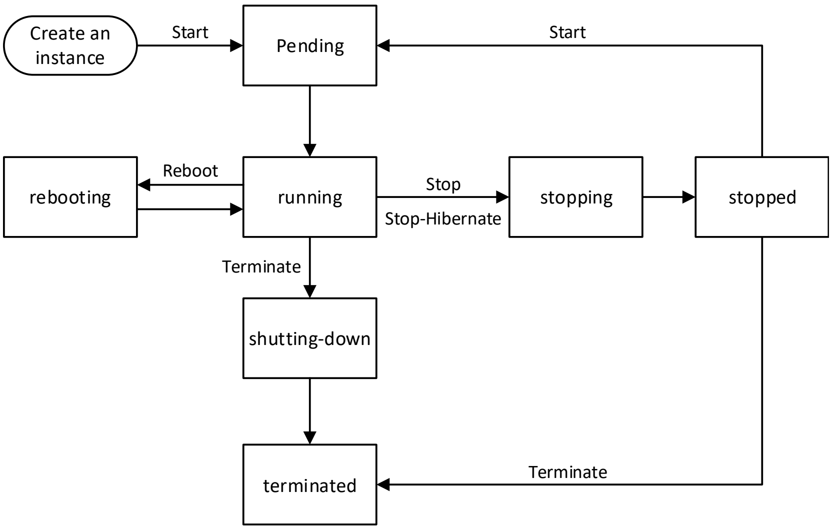


Figure 3: Lifecycle of an Amazon EBS-backed EC2 instance. This figure is adopted from [Ama20b].

on AWS F1 instances, users have to provide a netlist to AWS for the final configuration data generation.

2.3 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 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)* [Ama20a]. 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 4.2.

All user bitstreams are generated by AWS to guarantee the compatibility with the shell, and, most importantly, to perform checks (see Section 4.2) to ensure that the bitstream is not damaging the FPGA hardware (e.g., through short-circuits [BKT08, ACC$^+$95, ACC$^+$96, HUS99]). A more comprehensive overview of FPGA security aspects will be discussed in Section 4.2. Once an AFI-image is created, it will be stored by AWS.

2.4 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 [Ama20c]. 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.

Throughout its lifecycle, an AWS instance will go through several states, as shown in Figure 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.

3 Attacks and Countermeasures on FPGA-based Systems

The number of involved parties and the physical proximity between an attacker and the victim FPGAs are different between client-based FPGA usage and cloud-based usage, resulting in different attack scenarios, as discussed in the following paragraphs.

3.1 Client-based FPGA Usage

In client-based usage, FPGA accelerators (consisting of user, vendor, or third-party IPs) are integrated and deployed into an FPGA system. In this model, clients access own data and no other party can compromise the user data. However, an attacker with physical access can extract information stored in the system by using a wide range of techniques. Attack scenarios can vary from cheap modifications such as bitstream manipulation for causing Malicious Electrical Level Threats (MELT) [HUS99, BKT08] to complicated setups such as electromagnetic analysis [DMB05]. For example, an attacker can use an oscilloscopes to extract power traces to deploy power analysis attacks [SÖQP04, SMPQ06, MDS99, OOP03]; or it is possible to use inductive probes to sense electromagnetic emission from the chip to deploy electromagnetic analysis attacks [DMB05].

In the client-based model, stealing IPs is the attacker's primary motivation. Consequently, most related attacks target IP theft and only a few studies address the availability aspect [GOT17, BKT08]. To mitigate such threats, FPGAs can be configured with encrypted biststreams [McN10, TM08, TM17].

3.2 Cloud-based FPGA Usage

The Cloud computing model involves the CSP, the FPGA vendor, the client, and other cloud users. Cloud-based FPGA usage allows data leakage, fault injection, and denial-of-service attacks. See [Che20] for a comprehensive overview of security issues related to FPGA usage in the cloud.

3.2.1 Information Leakage

Cloud-based information leakage attacks require remote sensors. For instance, we can measure the speed of a circuit running on an FPGA, to measure temperature and voltage. To measure the speed of a design, we have to distinguish between synchronous and arbitrary designs running on the FPGA.

Synchronous designs are the norm for accelerator designs. Here, synchronous refers to register-transfer level designs (RTL), where all states are stored in flip-flops (i.e. registers) and where logic circuits form acyclic paths between flip-flops. In order to measure the speed of a path between flip-flops in a synchronous design, we have to overclock a circuit. For instance, we can set all bits on one input of an overclocked adder to `'1'` (i.e. int -1) and adding a 1 on the other input, will propagate a carry through the adder and successively changing the sum result bits from `'0'` to `'1'`. Therefore, by overclocking the adder such that we sample the sum result faster than what the critical path delay is allowing, we can measure the speed of the adder and consequently frequency and temperature of the FPGA.

A CSP has to decide for security (possible information leakage) or user demands. While in most practical applications overclocking is not acceptable, it was shown in [SOY$^+$20] to be beneficial for machine learning inference where a design was overclocked by up to 40% without a substantial loss in accuracy.

However, overclocked synchronous circuits are less suited for sensing than oscillators, and virtually all demonstrated remote attacks on leaking information from FPGAs use oscillators. In [LMG+20], it was reported that ring-oscillators on FPGAs could run at 6GHz, which allows measuring supply voltage fluctuations with high accuracy and RO-based sensors have been used for remote power side-channel attacks [SGMT18, ZS18]; crosstalk attacks [GRS19, RGE19, RPD+18]; and thermal-covert channels [TS19, Sze20].

Basic ring-oscillators are prohibited on AWS instances and will be detected by the vendor DRCs (see Section 4.2). Table 2 provides a list of alternative oscillator circuits that bypass the DRC, and that had been deployed on AWS instances. Note that the oscillators are free-running and do not depend on an external clock.

3.2.2 Fault-Injection Attacks

The dynamic power consumption from fast oscillation can cause voltage drops inside an FPGA, on the FPGA board power distribution network, and the power supply feeding an FPGA board. This can trigger timing violations and consecutive errors and uses the same effect that was used in the previous paragraph for information leakage. But instead of sensing voltage fluctuations, they are used to slow-down circuits for injecting faults [KGT18, MS19a], and for DoS attacks. DoS attacks are a core issue examined in this paper. Given the situation that none of the major cloud vendors offer multi-tenant FPGA sharing in their cloud infrastructures [Ama19b, Ali19, Hua19, Bai19, Ten19, Nim19, Mic19, OVH19], denial-of-service attacks are posing a severe and direct threat to most established cloud FPGA vendors. However, even multi-tenancy is not used by today, there are infrastructure components that are shared among multiple tenants. For example, excessive heat production may impact neighboring instances or power supply rails are typically shared such that a voltage drop from one FPGA instance may potentially impact another instance. Finally, consuming excessive power may damage equipment.

3.2.3 Countermeasures

Systems can be designed to be robust against fault injection attacks (e.g., through incorporating extra timing slack and randomizing clocks) [Man07, KK99]. Another strategy is to monitor a system to detect malicious behavior [ML08, ZSZF13]. Other studies advocated the necessity of detecting malicious designs prior to its execution [KGT19, LMG+20]. For preventing denial-of-service attacks, scanning of designs is currently the most important protection. Other countermeasures, like power and temperature monitoring, are still relevant but can only act as a secondary measure after scanning ensures that a system stays in well-defined operational conditions. The here presented paper-hammering attacks can ramp up power demand from a few watts to (theoretically) over a kilowatt in just a few nanoseconds, which is much faster than any monitoring system could report or react on.

A design scan should be robust enough to confidently prevent *all critical attacks* while preventing false positives. It is therefore vital to understand the exact surface of attack in order to tune scanners and design rule checks.

4 Denial-of-Service Attacks on Cloud FPGAs

In this section, we will present a denial-of-service (DoS) attack on AWS F1 instances and we will investigate the security fences implemented by AWS (in Section 4.2). As mentioned in Section 2, our attacks should work directly for other CSPs that offer Xilinx FPGAs because all CSPs rely on the vendor DRCs, which our attacks bypass. The attacks used in this paper use the low-level programmability of FPGAs and some related attacks had been shown for other vendors (e.g., Intel [PHT20, PRP+19, RPD+18] and Lattice [KGT19]).

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

	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

We crashed AWS F1 instances through power-hammering (i.e. drawing excessive power) such that the instance disappears temporarily from the pool of available instances. Without proof, our power-hammering will operate the FPGA outside its safe operational power envelope, which can impact the stable operation of other parts in the cloud datacenter due to shared infrastructure (e.g., through shared power-supply rails). Moreover, an attacker may deploy an attack not isolated, as being done in this paper, but coordinated over a larger number on instances simultaneously (e.g., to blow a fuse in a rack). Also, the attacks presented here could be used to gather information about the cloud datacenter (e.g., the physical neighborhood of specific nodes). In our examined scenario, an attacker aims at temporarily reducing the number of ready-to-use F1 instances at minimum cost.

4.1 Attack Model

We consider a CSP (here AWS) hosting multiple users reliably. The attacker can be a cloud service user (e.g., registered using a counterfeit identify and credit card to hide identity) or an IP core provider in the AWS marketplace (or a similar IP distribution system for clouds). In the latter case, the attacker is using other users to run the attack. For instance, an attacker may provide a popular bitcoin-mining accelerator with an embedded power-hammering Trojan to crash a large number of instances simultaneously.

The attack itself uses FPGA configurations that create excessive waste power through dynamic FPGA power consumption for crashing or damaging F1 instances (i.e. a DoS attack). The effects of power wasting maybe amplified through dynamic power consumption patterns (e.g., for triggering resonance effects in power regulator circuits [GOT17]).

4.2 The AWS FPGA Security Architecture

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

4.2.1 Fence 1 – Design Inspection

This fence 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 Section 2.2). All designs have to pass the DRCs, as listed in Table 1. The input to the registration process is a netlist in the vendor propriety design checkpoint format (DCP).[1]

First, an *integrity check* (a 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 [Xil19a]. This check is done by analyzing 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 [Xil20]. 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 softlogic 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 the vendor information. However, this test fails to detect several combinatorial loop designs, as listed in Table 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, as mentioned in Section 3.2.1.

AWS provides a few default clocks a user can choose of. However, using these clocks is not enforced (as demonstrated in Section 4.4) 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.

4.2.2 Fence 2 – Bitstream Generation

After passing the design inspection, the 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 netslist (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) [BKT08], 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 require bitstream manipulations). All designs passing Fence 1 will pass Fence 2 and 3.

[1] Users can provide encrypted DCP files to protect their IP. However, the CSP could brake the cryptographic mechanism. For example, attackers can generate the bitstream from the DCP and annotate the logic functions back to the DCP netlist [BSH08, PHK17]. Therefore, users have to trust the CSP for IP protection. However, this aspect is outside of the scope of this paper and is covered in [SVF$^+$21].

Table 2: Malicious designs that are currently deployable on AWS.

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

Designs 1,2,3,4,5,6,7 are taken from [LMG+20]. Design 8 is taken from [MLPK20]. Design 9 is an enhancement of Design 3 (See Figure 5b). *: On their own, Designs 2,3,4 are not well suited for DoS attacks. However, DOS is possible using additional glitch amplification.

4.2.3 Fence 3 – FPGA API

After the AFI is generated, it can be loaded to 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 [Ama19a], and programming is only possible through an AWS-provided API. For programming, the command *fpga-load-local-image* is encapsulating the vendor programming mechanism [Xil19c]. For debugging, AWS provides users with virtual LEDs, virtual DIP switches, and virtual JTAG [Ama19a]. 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.

4.2.4 Fence 4 – FPGA Monitoring

At runtime, AWS uses three safety mechanisms. The first one is *power monitoring*. The Xilinx VU9P datacenter FPGAs provide a system monitor mechanism featuring three 10-bit 200kSPS ADCs to read FPGA temperature sensors and core voltages [Xil19b]. Related work suggests that the available sampling rate is insufficient to detect quick voltage transients [ZSZF13] 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 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 ind will cut off the PCIe connection between the FPGA and the host.

4.3 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

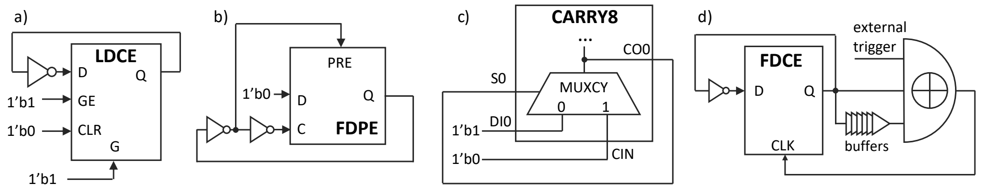


Figure 4: Oscillator designs deployable on AWS F1 instances: (a) transparent latch, (b) flip-flop with asynchronous preset, (c) ring oscillators implemented through carry-chain logic, (d) a self-oscillating circuit using glitch amplification.

an Alveo U200 datacenter card) to measure power increments caused by different power-hammering circuits. This allows us calibrating 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, dynamic power can contribute to 20% to 70% of the total power dissipation [SGDK92]. 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 [Xil19d]. However, malicious circuits can switch 100× faster [LMG$^+$20], and the power estimator is not well suited to catch such designs.

Figure 5 and Table 2 show attacks that bypass the security Fences 1-3. The reported power gains had been derived on a small region of the FPGA (≈ 10% of the total resources) and than scaled up according to the user FPGA resources available 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 4d), 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 than 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 reported for design 8. 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 5b). The enable signal is used to control the oscillation.

4.4 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 5a), this oscillator was used to sequentially activate a chain of power-hammering circuits, see Figure 5b). 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 behavior 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 5c. We observed that when increasing power beyond the clock gating point (105W), the SSH terminal freezes at about 134W, which

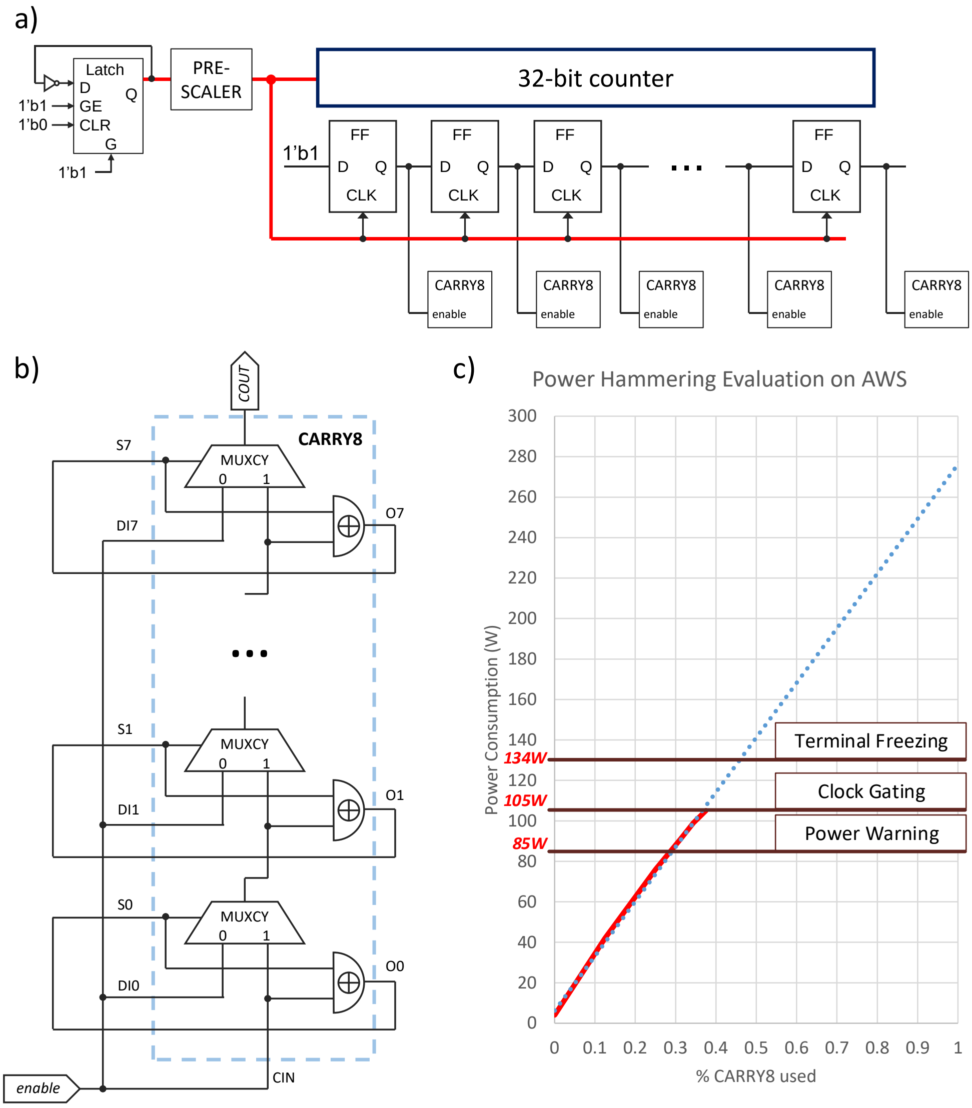


Figure 5: Power-hammering designs and power evaluation on AWS. a) Self-clocked design to bypass the clock gating protection. b) One carry-chain primitive forms 8 combinatorial loops. c) Evaluation using 81920 carry-chain primitives.

may have triggered an over-temperature shutdown. All power levels had been measured indirectly through 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 cock gating can be bypassed and 2) exceeding 134W can freeze an instance with a loss of the SSH connection.

4.5 FPGA Fingerprinting on AWS EC2 F1

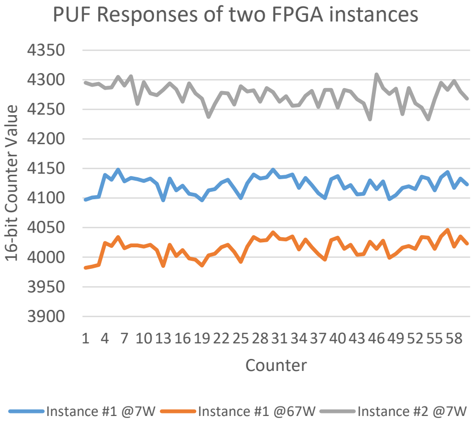


Figure 6: AWS F1 FPGA PUF responses.

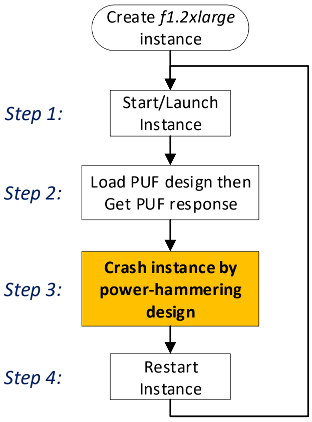

Create *f1.2xlarge* instance
Step 1:
Start/Launch Instance
Step 2:
Load PUF design then Get PUF response
Step 3:
Crash instance by power-hammering design
Step 4:
Restart Instance

Figure 7: AWS attack flow.

In the last paragraph, we presented an experiment that crashed an AWS F1 FPGA instance. In order to investigate the behavior 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 [Xil20]. Therefore, we implemented a PUF for fingerprinting the FPGAs. 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 [DPGV15] and our contribution is providing an implementation that is deployable on AWS F1 instances where combinatorial feedback loops cannot be used (see also 4.2.1).

Recently, a group from Yale University used DRAM decay to fingerprint AWS F1 instances [TXG$^+$20]. 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 be 456*ps* 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 count values. We identify the match of two PUF responses by calculating the Pearson Correlation Coefficient between 2 counter value sets. In our experiments, we defined that two sets match if the correlation coefficient is larger than 85%.

Figure 6 shows 3 PUF responses where the first and last traces were derived from the same FPGA but at different power levels (7W and 67W). These traces show a strong correlation (94%) in the shape of the count values but with a temperature-related offset. Therefore, the PUF responses are robust to temperature changes. The trace in the middle was derived from a different FPGA as a reference. These experiments allow 1) measuring temperature of an FPGA (which could, for example, reveal information of 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.

4.6 DoS on AWS – Deployment & Attack

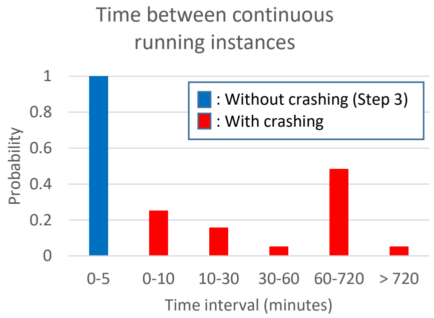


Figure 8: Time interval between two running instances.

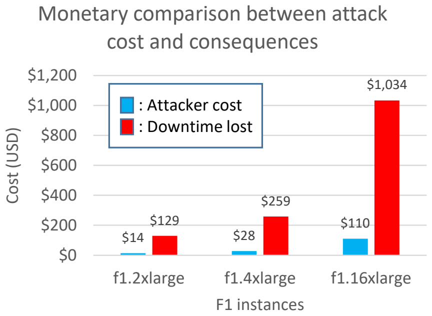


Figure 9: Estimated attack cost and loss after 100 attacks with an attack time of 5 minutes and downtime of 52 minutes.

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 [GOT17].

The attack was performed, as shown in Figure 7. After creating an instance, we firstly 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 [Ama20b]. 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. After this, we run another 96 experiments including crashing of the instance. As shown in Figure 8, 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*

Table 3: Comparison of current mitigation methods.

Methods	Attacks on Availability		Attacks on Confidentiality			Cost	Reuse	Using FPGA Resources
	RO-based Power-hammering	Glitch-based Power-hammering	RO-based Power-analysis	Cross-talk Leakage	Fault Injection			
Hardware	✓[1]	✓[1]	✓[1]	✓[1]	✓[1]	VERY-HIGH	✓	×
Design Guidelines	✓	✓	✓	✓	✓	HIGH	✓	×
Runtime Monitoring	✓[1]	✓[1]	✓[1]	×	✓[1]	MID	✓	✓
Design Inspection	✓	×[2]	✓	✓	×[2]	LOW	✓	×

[1]: Works robust only for small power-hammering levels. [2]: Could be done, but not implemented so far.

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 [TXG^{+}20] 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 behavior 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 market place, and after a forced crash, users will experience extensive delays for receiving 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 of selling the service of crashed F1 instances. Given the service is billed in second (if it runs for less than a minute, then cost is rounded up to the next minute) [Int20], we can estimate the monetary loss. Figure 9 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.

5 Mitigation Strategies

Oscillators can be used for leaking information, as shown in Section 2 and (most importantly) for DoS attacks, as demonstrated in the last section. By using glitch amplification and high fanout nets, *any oscillator circuit could be used for power-hammering with a potential exceeding well above 1KW*. Therefore, it is important to prevent any oscillator to be deployed. Without this, any FPGA monitoring could easily be overwhelmed, and the risk of power-hammering would be virtually unmanageable.

The following lists categorizes the most prominent mitigation strategies (see also Table 3):

1. **Hardware:** By designing FPGA boards with greater capabilities for supplying power and cooling and by over-provisioning power supplies, systems can be built more robust. This is the most costly mitigation strategy but has the advantage that it can mitigate almost any type of attack transparently to users. However, this approach is limited to moderate power-hammering levels and cannot be thought of as the only method because it is unfeasible to entirely mask power-hammering potentials that can reach kilowatts.

2. **Design Guidelines:** This requires designers to follow strict coding guidelines such that potentially malicious designs (e.g., asynchronous elements) are prohibited. This either requires a certified supply chain or design inspection to enforce the design guidelines. Moreover, this approach may result in poor customer satisfaction due to the imposed guidelines.

3. **Runtime Monitoring:** Runtime monitoring of power and temperature, as done by AWS in Fence 4, is useful for managing the benign operation of the FPGA hardware. However, similar to designing the hardware more robust, monitoring can only serve as a mitigation strategy for attacks at low to moderate power-hammering levels. Considering a possible power-hammering potential in the kilowatt domain would not allow us to monitor or react fast enough to prevent unstable operation or equipment damage.

4. **Design Inspection:** This is the key protection applied in Fence 1 by AWS. This, in particular, includes checking for design errors (DRCs). However, the present DRCs offered by the Xilinx vendor tools had been created mainly to identify design errors and not for spotting malicious circuits. This is the main reason why we found multiple design that bypass Fence 1 (see Table 2). Nevertheless, design inspection is virtually for free and protects against the most severe power-hammering threats, which makes security manageable for the previous mitigation strategies. However, this implies that the inspection is robust to *catch all critical designs*. In Section 5.1, we will demonstrate that the most severe hardware security issues for FPGA cloud computing (short circuits and self-oscillating circuits) can be reliably detected through design inspection. However, even design inspection is cheap to implement, this process needs continuous maintenance. Design inspection could be performed at the netlist level or bitstream level. The latter provides users with better IP protection.

Each approach has its pros and cons in terms of cost and effectiveness. To mitigate attacks on availability, we believe that design inspection should be the preferred method. In the present AWS security ecosystem, this would add more tests to Fence 2, which is the bitstream and AFI generation. With this, we can limit the power-hammering potential that can be mounted on an FPGA. Opposed to this, hardware robustness and monitoring are strategies that mitigate attacks that are running on the FPGA.

In practice, similar to the present deployment strategy that is in place at AWS, we believe that it needs a hybrid solution where design inspection detects all severe threats such that hardware robustness and runtime monitoring operate in their acceptable operational conditions. This is a non-trivial task and will require continuously maintaining the design inspection tools to ensure that 1) possible future threats are detected and 2) benign designs are not rejected (false positives). We believe that this model is feasible as the basic design principles that can be used for attacks are rather limited (due to the limited number of FPGA primitive types available). This potentially allows implementing a holistic scanner solution for catching FPGA hardware security issues such as oscillators for side channels and power-hammering. An example of an FPGA virus scanner for datacenter FPGAs was proposed in [LMG+20], and an extension of this scanner is presented in Section 5.1. The scanner is supposed to complement the vendor DRC checks.

5.1 Netlist/Bitstream Scanning Mechanisms

In [LMG+20] and [MLPK20], the open-source FPGA virus-scanner framework FPGADefender for Xilinx UltraScale+ devices was introduced. The framework retrieves the netlist from a bitstream (via an academic tool BitMan [PHK17]) and scans for a set of pre-defined virus signatures (i.e. properties of the design provided). For this work, we provide two contributions to FPGADefender: 1) a reimplementation of the scanner that can handle

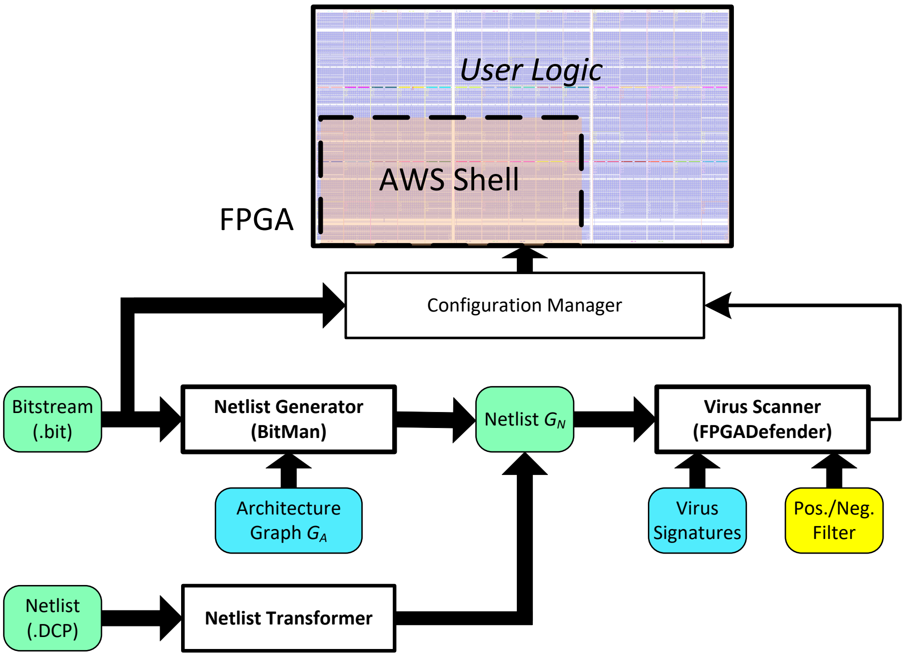


Figure 10: The FPGADefender Framework. The input is a bitstream (alternatively a netlist) that is analyzed by a virus scanner engine that is parameterized through virus signature files. The test result is passed to a configuration manager that decides if a configuration is allowed to be loaded onto the FPGA or not.

large datacenter FPGA designs (see Section 5.2) and 2) we provide an input path to scan for malicious circuits directly from the Xilinx vendor-proprietary DCP netlist file format. This is shown in Figure 10 through the block Netlist Transformer. The latter allows both end users and cloud providers to embed the FPGADefender scans as an add-on to existing DRCs. The checks performed by FPGADefender include:

- A *Combinatorial Feedback Loop Scanner* is used to scan designs for combinatorial loops in a similar way to the vendor DRCs. This scanner covers all possible combinatorial feedback loop paths, including paths through carry-chains, transparent latches, DSP blocks, and cascading multiplexers, which are all not detected by the Xilinx vendor DRC (see also Table 2).

- *Fanout Detection* is used to detect drivers with a large fanout, which can indicate malicious designs that may use high fanout nets for controlling the attack and for using routing resources for power-hammering.

- *Disallowed Port Detection* allows detecting usage of specific ports on the FPGA (e.g., for disallowing the design from accessing any of the shell resources). This can be used to disallow access to specific FPGA primitives or to whole areas of the FPGA, without limiting routing through these areas.

- *Disallowed Path Detection* detects the usage of specific wires, to prevent designs from using unsafe wires in the FPGA. Disallowing specific wires can prevent malicious designs from being able to extract data from nearby wires belonging to other parts of the design (e.g., for exploiting crosstalk effects [GRS19, RGE19]).

- *Short Circuit Detection* is used to ensure that the encoding of the switchboxes is valid, and will not cause a short circuit. This situation should never occur from bitstreams generated by the vendor tools, but can be created trivially by bitstream manipulations. Therefore, only a cloud service provider that allows customers to

Table 4: Comparison of Virus Scanner Implementations (C++ refers to the new implementation and Python refers to the implementation presented in [LMG+20]).

Design	C++ Memory	Python Memory	C++ Runtime	Python Runtime
[1)]ring_osci	2997 Mb	10369 Mb	40s	15m
[2)]full_vadd	3162 Mb	10524 Mb	44s	2h 36m 38s
[3)]4xeuro	5183 Mb	> 11080 Mb	1m 30s	> 30h

[1)] ring_osci is a single ring oscillator on an mostly empty F1 FPGA.
[2)] full_vadd is a medium size HLS generated design (≈ 1/3rd of the FPGA capacity).
[3)] 4xeuro is a large HLS generated design, which fills the chip. The Python scan for this design was stopped after 30 hours uncompleted.

upload their own bitstreams is vulnerable; providers who enforce cloud-side synthesis are not affected. Therefore, this check is only used for bitstream scanning and not for analyzing netlists (i.e. design checkpoints).

- *Glitch Detection* is used to calculate the worst-case dynamic power consumption of a design. Glitches in an FPGA design can be created through carefully designed combinatorial logic and its physical implementation. For example, a 6-input XOR can create 6 glitches in a clock cycle leading to a corresponding high dynamic power consumption [MLPK20]. The work in [MLPK20] showed that the vendor power estimator cannot reliably detect power-hammering created through glitch amplification.

The different detectors generate numerical scores that indicate the maliciousness of a design (with a low score indicating a relatively benign design, and a high score indicating problematic designs). Cloud vendors can use these scores to decide if a design will be accepted or not (e.g., for generating an AWS AFI).

5.2 FPGADefender for Datacenter FPGAs

When testing the original FPGADefender implementation [LMG+20], we found that scanning large designs, as usually deployed on datacenter FPGAs, took over a day to complete. To enable virus scanning on large FPGAs, a C++ implementation was developed, with a focus on runtime.

Several techniques are utilized to improve the speed of the scanning engine. The python data structures and algorithms are re-written in C++, using the C++ STL data structures. The C++ scanners graph implementation generates internal pointer arrays, for faster netlist traversal. The C++ scanner uses efficient algorithms for scanning, such as Tarjans algorithm for finding cycles. The LUT netlist calculation is simplified and external dependencies are removed. For instance, [LMG+20] uses the Espresso logic minimization library [BHMSV84] to scan for unused LUT imputs and we replaced this part with own functions. Additionally, the FPGA chip description data structures have been optimized for the access patterns of the netlist generator.

A comparison of the C++ and Python implementations in Table 4 shows the difference in scanning speed and our reimplementation reduces scanner speed from over a day to minutes.

6 Discussion

When we realized that power-hammering could be deployed on AWS instances with theoretically over 1KW power-hammering potential, we contacted AWS. We got permission to run tests on their production system. However, the here presented attacks do not exploit the full potential of the attack, both in terms of the power wasted and in terms of power-hammering pattern. We limited our attack as we only want to point out the problem and not cause serious harm to AWS. We would also like to stress that most of the attack circuits listed in this paper are published and rather trivial to implement. Our main contribution in this paper is to explore the AWS FPGA security ecosystem in a more systematic way.

To enable CSPs to protect their equipment, we developed a DRC receipt that uses the vendor DRC and static timing analysis to perform more robust design inspection in Fence 2. This can be seen as a patch and is integrated through a single TCL script. The patch will detect all oscillator circuits that we tested on AWS instances. The custom DRC is available under:
`https://github.com/FPGA-Research-Manchester/XilinxTCL_utilities/tree/master/01_CustomDRCs_CombinatorialLoopCheck`

We want to stress that our findings are not bound to AWS or a specific FPGA vendor. The attacks presented in this paper would work very similarly on FPGAs from other vendors (e.g., [RPD+18] examined power-hammering and ring-oscillator designs on Intel FPGAs). The reason why we (and most researchers) base research on Xilinx is due to being the market leader and because their tools provide powerful APIs. While this may allow attackers to use those interfaces, it is providing the means to implement independent scanners, and most importantly, providing overall trust in tools and the hardware. We believe that FPGA hardware security in cloud settings is manageable in similar ways to managing security threats in other pieces of IT equipment. Moreover, the relative openness of Xilinx (and other vendors like Lattice Semiconductor) in terms of documentation and tool APIs allows implementing more trustworthy security solutions.

7 Conclusion

In conclusion, this paper presents the first demonstrated DoS attack deployed on AWS F1 FPGA cloud instances. Our attack is based on FPGA power-hammering, which is drawing an excessive amount of dynamic power. By using FPGA fingerprinting through PUFs, we found that crashed instances are typically unavailable for at least about an hour, which is a strong indication that our attack was successful and that it may even require the intervention of a human operator.

Most important, we have proposed scanning mechanisms that can mitigate all attacks presented in this paper and that can be seamlessly embedded into the AWS security infrastructure for AWS FPGA instances. This includes an FPGA virus scanner and a new design rule check receipt to be used with the Xilinx vendor tools. This scanner can distinguish between user accelerators and malicious designs that can waste one to two orders of magnitude more power than benign accelerators would usually do.

With this work, we want to contribute to the currently very active research on FPGA hardware security in order to understand possible threat scenarios better and for providing mitigation mechanisms. While there is more research needed, we strongly believe that multi-tenancy as well as FPGA-as-a-Service can be accepted from a security point of view, and that design inspection at bitstream and/or netlist level provides the technology foundation to enable this movement.

Acknowledgment

This work is kindly supported by the UK National Cyber Security Centre through the project *rFAS* (grant agreement 4212204/RFA 15971) and by the European Commission through the project *EuroEXA* (grants 754337). We also thank the Xilinx University Program for providing tools and boards.

References

[ACC$^+$95] 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, April 1995.

[ACC$^+$96] R. Amerson, R. Carter, W. Culbertson, P. Kuekes, G. Snider, and L. Albertson. Plasma: An FPGA for Million Gate Systems. In *Fourth International ACM Symposium on Field-Programmable Gate Arrays*, pages 10–16, Feb 1996.

[Ali19] Alibaba Inc. Deep Dive into Alibaba Cloud F3 FPGA as a Service Instances, 2019. online: https://www.alibabacloud.com/blog/deep-dive-into-alibaba-cloud-f3-fpga-as-a-service-instances_594057.

[Ama19a] Amazon. Amazon FPGA Image (AFI) Management Tools, 2019. online: https://github.com/aws/aws-fpga/blob/master/sdk/userspace/-fpga_mgmt_tools/README.md.

[Ama19b] Amazon Inc. Amazon EC2 F1 Instances, 2019. online: https://aws.amazon.com/ec2/instance-types/f1/.

[Ama20a] Amazon Inc. AWS EC2 AFI Creation, 2020. online: https://github.com/aws/aws-fpga/blob/master/SDAccel/docs/Setup_AWS_CLI_and_S3_Bucket.md.

[Ama20b] Amazon Inc. AWS EC2 Instance Lifecycle, 2020. online: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-instance-lifecycle.html.

[Ama20c] Amazon Inc. AWS EC2 Storage, 2020. online: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/Storage.html.

[Bai19] Baidu. Baidu Cloud, 2019. online: https://cloud.baidu.com/product/fpga.html.

[BHMSV84] R. Brayton, G. Hatchel, C. McMullen, and A. Sangiovanni-Vincentelli. *Logic Minimization Algorithms for VLSI Synthesis.* Kluwer Academic Publishers, 1st edition, 1984.

[BKT08] C. Beckhoff, D. Koch, and J. Torresen. Short-Circuits on FPGAs Caused by Partial Runtime Reconfiguration. In *2010 International Conference on Field Programmable Logic and Applications*, pages 596,601. IEEE, 2010-08.

[BSH08] F Benz, A Seffrin, and S. A Huss. Bil: A Tool-chain for Bitstream Reverse-engineering. In *22nd International Conference on Field Programmable Logic and Applications (FPL)*, pages 735,738. IEEE, 2012-08.

[Che20] Chenglu Jin, Vasudev Gohil, Ramesh Karri, Jeyavijayan Rajendran. Security of Cloud FPGAs: A Survey. *ACM Computing Surveys*, 2020.

[DMB05] E. De Mulder, P. Buysschaert, S.B. Öre, P. Delmotte, B. Preneel, G. Vandenbosch, and I. Verbauwhede. Electromagnetic analysis attack on an FPGA implementation of an elliptic curve cryptosystem. In *EUROCON 2005 - The International Conference on Computer as a Tool*, volume II, pages 1879,1883, 2005.

[DPGV15] Jeroen Delvaux, Roel Peeters, Dawu Gu, and Ingrid Verbauwhede. A Survey on Lightweight Entity Authentication with Strong PUFs. *ACM Comput. Surv.*, 48(2), October 2015.

[GOT17] D. Gnad, F. Oboril, and M. Tahoori. Voltage Drop-based Fault Attacks on FPGAs Using Valid Bitstreams. In *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*, pages 1–7. IEEE, 2017.

[GRS19] 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, September 2019.

[Hua19] Huawei. Huawei FPGA Accelerated Cloud Server, 2019. online https://www.huaweicloud.com/en-us/product/fcs.html.

[HUS99] I. Hadžić, S. Udani, and J. Smith. FPGA Viruses. In *International Workshop on Field Programmable Logic and Applications*, pages 291–300. Springer, 1999.

[Inc14] Xilinx Inc. The Xilinx SDAccel Development Environment - Bringing The Best Performance/Watt to the Data Center, 2014.

[Int20] Intel. Stop and Start Your Instance, 2020. online: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/Stop_Start.html.

[Jak20] Jakub Szefer. Single-Tenant Cloud FPGA Security, 2020. online: http://www.fccm.org/proceedings/2020/Workshops/Future_of_FPGA_-Workshop/Single-Tenant%20Cloud%20FPGA%20Security.pdf.

[JCM15] Anju 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.

[KGT18] J. Krautter, D. Gnad, and M. 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.

[KGT19] J. Krautter, D. Gnad, and M. Tahoori. Mitigating Electrical-level Attacks Towards Secure Multi-Tenant FPGAs in the Cloud. *ACM Trans. Reconfigurable Technol. Syst.*, 12(3):12:1–12:26, aug 2019.

[KK99] Oliver Kömmerling and Markus G Kuhn. Design Principles for Tamper-Resistant Smartcard Processors. *Smartcard*, 99:9–20, 1999.

[LMG$^+$20] 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.

[Man07] Stefan. Mangard. *Power Analysis Attacks : Revealing the Secrets of Smart Cards*. Springer US, Boston, MA, 2007.

[McN10] Steven McNeil. Solving Today's Design Security Concerns. *Xilinx Corporation*, 2010.

[MDS99] Thomas S Messerges, Ezzy A Dabbish, and Robert H Sloan. Power Analysis Attacks of Modular Exponentiation in Smartcards. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 144–157. Springer, 1999.

[Mic19] Microsoft. Microsoft Azure, 2019. online:
https://azure.microsoft.com/en-gb/.

[ML08] 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, 2012-08.

[MLPK20] K. Matas, T. M. La, K. D. Pham, and D. Koch. Power-hammering through Glitch Amplification – Attacks and Mitigation. In *2020 IEEE 28th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 65–69, 2020.

[MS19a] Dina Mahmoud and Mirjana Stojilovic. Timing Violation Induced Faults in Multi-Tenant FPGAs. *2019 Design, Automation amp; Test In Europe Conference amp; Exhibition (DATE)*, pages 1745–1750, 2019.

[MS19b] S. S. Mirzargar and M. Stojilovic. Physical Side-Channel Attacks and Covert Communication on FPGAs: A Survey. In *Proceedings of the 29th International Conference on Field-Programmable Logic and Applications*, FPL, September 2019.

[Nim19] Nimbix. FPGAs, What's Old is New Again, 2019. online:
https://www.nimbix.net/fpgas-cloud.

[OOP03] Berna Ors, Elisabeth Oswald, and Bart Preneel. Analysis Attacks on an FPGA – First Experimental Results. volume 2779, pages 35–50, 09 2003.

[OVH19] OVH. OVH Cloud, 2019. online:
https://us.ovhcloud.com/press/press-releases/2017/ovh-and-accelize-partner-deliver-fpga-acceleration-service-through-ovh.

[PCC$^+$14] A. Putnam, A. Caulfield, E. Chung, D. Chiou, K. Constantinides, J. Demme, H. Esmaeilzadeh, J. Fowers, G. Gopal, J. Gray, M. Haselman, S. Hauck, S. Heil, A. Hormati, J. Kim, S. Lanka, J. Larus, E. Peterson, S. Pope, A. Smith, J. Thong, P. Xiao, and D. Burger. A Reconfigurable Fabric for Accelerating Large-scale Datacenter Services. In *Proceeding of the 41st Annual International Symposium on Computer Architecuture*, ISCA '14, pages 13–24, Piscataway, NJ, USA, 2014. IEEE Press.

[PHK17] 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), 2017*, pages 894–897. IEEE, 2017.

[PHT20] 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.

[PRP$^+$19] G. Provelengios, C. Ramesh, SB. Patil, K. Eguro, R. Tessier, and D. Holcomb. Characterization of Long Wire Data Leakage in Deep Submicron FPGAs. In *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA)*, page 292–297. ACM/SIGDA, 2019.

[RGE19] 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.

[RPD+18] C. Ramesh, S. Patil, S. Dhanuskodi, G. Provelengios, S. Pillement, D. Holcomb, and R. Tessier. FPGA Side Channel Attacks without Physical Access. In *2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 45–52. IEEE, 2018.

[SGDK92] Amelia Shen, Abhijit Ghosh, Srinivas Devadas, and Kurt Keutzer. On Average Power Dissipation and Random Pattern Testability of CMOS Combinational Logic Networks. In *ICCAD*, 1992.

[SGMT18] F. Schellenberg, D. Gnad, A. Moradi, and M. Tahoori. An Inside Job: Remote Power analysis Attacks on FPGAs. In *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1111–1116. IEEE, 2018.

[SMPQ06] F.-X. Standaert, F. Mace, E. Peeters, and J.-J. Quisquater. Updates on the Security of FPGAs against Power Analysis Attacks. volume 3985, pages 335,346. Springer Verlag, 2006.

[SÖQP04] François-Xavier Standaert, Sıddıka Berna Örs, Jean-Jacques Quisquater, and Bart 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.

[SOY+20] Behzad Salami, Erhan Baturay Onural, Ismail Emir Yuksel, Fahrettin Koc, Oguz Ergin, Adrian Cristal Kestelman, Osman S. Unsal, Hamid Sarbazi-Azad, and Onur Mutlu. An Experimental Study of Reduced-Voltage Operation in Modern FPGAs for Neural Network Acceleration, 2020.

[SVF+21] Shaza Zeitouni, Jo Vliegen, Tommaso Frassetto, Dirk Koch, Ahmad-Reza Sadeghi, and Nele Mentens. Trusted Configuration in Cloud FPGAs. In *29th Annual IEEE International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2021.

[Sze20] Jakub Szefer. Thermal and Voltage Side and Covert Channels and Attacks in Cloud FPGAs. In *The 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, FPGA '20, page 22, New York, NY, USA, 2020. Association for Computing Machinery.

[Ten19] Tencent. Tencent Cloud, 2019. online:
https://intl.cloud.tencent.com/document/product/213/11518.

[TM17] Steve Trimberger and Steve McNeil. Security of FPGAs in Data Centers. In *2017 IEEE 2nd International Verification and Security Workshop (IVSW)*, pages 117–122. IEEE, 2017.

[TM08] S. Trimberger and J. Moore. FPGA Security: Motivations, Features, and Applications. *Proceedings of the IEEE*, 102(8):1248,1265, 2014-08.

[TS19] Shanquan Tian and Jakub Szefer. Temporal Thermal Covert Channels in Cloud FPGAs. In *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 298–303. ACM, 2019.

[TXG+20] Shanquan Tian, Wenjie Xiong, Ilias Giechaskiel, Kasper Rasmussen, and Jakub Szefer. Fingerprinting Cloud FPGA Infrastructures. In *The 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, FPGA '20, page 58–64, New York, NY, USA, 2020. Association for Computing Machinery.

[Xil19a] Xilinx. Partial Reconfiguration, 2019. online: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2019-_ 1/ ug947-vivado-partial-reconfiguration-tutorial.pdf.

[Xil19b] Xilinx. UltraScale Architecture System Monitor, 2019.

[Xil19c] Xilinx. Vivado User Guide: Programming and Debugging, 2019. online: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2019-_1/ug908-vivado-programming-debugging.pdf.

[Xil19d] Xilinx. Xilinx Power Analysis and Optimization, 2019. online: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2019-_ 2/ug907-vivado-power-analysis-optimization.pdf.

[Xil20] Xilinx. UltraScale Architecture Configuration, 2020. online: https://www.xilinx.com/support/documentation/user_guides/ug570-ultrascale-configuration.pdf.

[ZS18] M. Zhao and G. Suh. FPGA-based Remote Power Side-channel Attacks. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 229–244. IEEE, 2018.

[ZSZF13] K. Zick, M. Srivastav, W. Zhang, and M. French. Sensing Nanosecond-scale Voltage Attacks and Natural Transients in FPGAs. In *Proceedings of the ACM/SIGDA international symposium on Field programmable gate arrays (FPGA)*, pages 101–104. ACM, 2013.