FPGA-to-CPU Undervolting Attacks

Dina G. Mahmoud*, Samah Hussein*, Vincent Lenders† and Mirjana Stojilović*
*EPFL, School of Computer and Communication Sciences, Lausanne, Switzerland
†armasuisse Science and Technology, Cyber-Defence Campus, Thun, Switzerland

***Abstract*—FPGAs are proving useful and attractive for many applications, thanks to their hardware reconfigurability, low power, and high-degree of parallelism. As a result, modern embedded systems are often based on systems-on-chip (SoCs), where CPUs and FPGAs share the same die. In this paper, we demonstrate the first undervolting attack in which the FPGA acts as an aggressor while the CPU, residing on the same SoC, is the victim. We show that an adversary can use the FPGA fabric to create a significant supply voltage drop which, in turn, faults the software computation performed by the CPU. Additionally, we show that an attacker can, with an even higher success rate, execute a denial-of-service attack, without any modification of the underlying hardware or the power distribution network. Our work exposes a new electrical-level attack surface, created by tight integration of CPUs and FPGAs in modern SoCs, and incites future research on countermeasures.**

***Index Terms*—FPGA, CPU, Security, Undervolting, Fault Injection**

I. Introduction

Heterogeneous computing systems combining field programmable gate arrays (FPGAs) and central processing units (CPUs) are commonly used in modern computing. FPGAs offer hardware parallelism and reconfigurability, which makes them suitable for workloads whose requirements change frequently and for highly parallel applications such as machine learning and computer vision. CPUs offer the advantage of being general-purpose and easily programmable. As a result, we find heterogeneous systems combining FPGAs and CPUs inside the same package, embedded system, datacenter, or cloud (e.g., Xilinx and Intel offer system-on-chip (SoC) solutions combining their FPGAs with ARM processors [1], [2]).

Recently, researchers have demonstrated the use of dynamic voltage frequency scaling (DVFS) interfaces for *remote* fault injection exploits against ARM and Intel CPUs [3]–[5]. About the same time, FPGA researchers have found that programming FPGAs with so-called *power-wasting circuits*, which draw considerable current, can produce notable fluctuations of the supply voltage. The resulting voltage drop may then be leveraged by an adversary to reset the target system [6], [7], or, if carefully controlled, to inject computational faults in the host FPGA [8]–[10].

In this work, we examine the possibility of leveraging the effects of power-wasting circuits deployed on the FPGA to affect the operation of a CPU in the same heterogeneous system. We believe this to constitute a novel threat; the supply chain and the design process for FPGAs can be separate from those of the CPU and, therefore, defenses targeting only the CPU (e.g., simply limiting the level of voltage control of the DVFS interfaces [4]) will not work against voltage drops originating from the FPGA. The attacker thus possesses more flexibility, potentially leveraging the combination of *both* DVFS and FPGA power wasters. To the best of our knowledge, we demonstrate the first successful undervolting attack, in which an aggressor FPGA injects repeatable faults into the computation of a CPU in the same SoC. We investigate the possibility of, first, injecting faults into the software executing on the CPU and, second, performing a denial-of-service (DoS) attack.

II. Background and Related Work

In this section, we explain the mechanisms used for remote electrical-level fault-injection exploits on CPUs, FPGAs, and heterogeneous systems, and discuss the recent works that leveraged them.

A. CPU-based Attacks

For correct operation, electronic circuits must operate within specific frequency and voltage ranges. When these constraints are deliberately violated, faults may be injected into the operation of the circuit. Researchers have recently leveraged the possibility of manipulating the voltage and frequency for remote attacks on CPUs. Tang et al. made use of the DVFS interfaces available on ARM processors, to change the operating frequency beyond the safe limits for the processor [3]. Other researchers have also demonstrated the possibility of manipulating DVFS interfaces to change the voltage and fault Intel and ARM processors [4], [5], [11]. These exploits broke the security of encryption algorithms and of trusted execution environments (e.g., Intel SGX and ARM TrustZone). Furthermore, researchers have demonstrated how to automatically generate highly effective stressor codes, which increase the value of the minimum voltage necessary for the correct operation of a CPU [12].

B. FPGA-based Attacks

Unlike CPUs, FPGAs do not usually have DVFS interfaces. However, using the low-level programmability of FPGAs, researchers have demonstrated how to design and deploy a variety of designs to consume significant amounts of power [6], [7], [13], [14]. The high power consumption results in a voltage drop, which can inject faults, or even reset the entire board [6]. Researchers have used this voltage drop to learn AES keys, bias random number generators, and remotely activate hidden Trojans [8]–[10]. The most basic power-wasting

This research is supported by armasuisse Science and Technology.

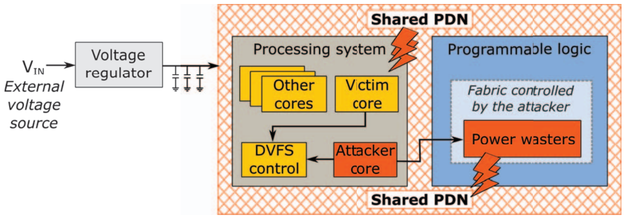

Fig. 1. Threat model, where the attacker-controlled programmable logic induces faults in the victim core, through the shared power delivery network.

design is a combinational ring oscillator (RO): an inverter in a closed loop, oscillating at a very high frequency. Usually, attacker designs turn the inverter into a NAND, where the other input is an enable signal to ensure control over the attack primitive. Alternative power-wasting designs do not draw as much power, but are suitable for attacks on the cloud FPGAs (as combinational ROs can be detected and are not allowed on, e.g., Amazon EC2 F1 instances [15]). These designs make use of CARRY primitives, registers, and long wires [7], [14].

C. *Heterogeneous Systems Exploits*

More recently, researchers have also started exploring the types of exploits possible in a heterogeneous scenario. Giechaskiel et al. demonstrated covert channels between graphics processing units (GPUs), CPUs, and FPGAs, sharing the same power supply unit in a datacenter [16]. Ring oscillators were used as both transmitters and receivers on the FPGA, and for transmitters on the CPU and the GPU, computationally heavy codes were used. Zhao et al. demonstrated the feasibility of side-channel attacks against the CPU, using sensors on an FPGA on the same chip [17]. Gnad et al. showed that the reset resulting from the activity of ROs in the FPGA fabric affects both the CPU and the FPGA on the same chip [6]. However, to the best of our knowledge, no previous work has investigated the possibility of injecting faults from the FPGA into the software executing on the CPU.

In this work, we utilize power-wasting circuits in the programmable logic (PL) of the FPGA to investigate the effects of FPGA-induced undervolting on the operation of a CPU on the same chip, which can potentially be leveraged to inject faults or carry out a denial-of-service attack. Similarly to works on CPU exploits [3], we assume that we can change the frequency of the victim processor. Unlike them, we do not assume the ability to change the processor voltage, and we increase the frequency only to values for which the processor still operates without faults. Unlike works on FPGA attacks, we do not aim to inject faults into other circuits in the PL, but instead target the CPU. Finally, similarly to previous work which has shown that the reset induced by power wasters on the FPGA can affect a CPU residing on the same chip [6], we observe DoS also due to sending the victim CPU code into an infinite loop.

III. Threat Model

Our threat model assumes an attacker targeting heterogeneous platforms which combine FPGAs with CPUs, and where the power distribution network (PDN) is shared, as illustrated in Fig. 1. While this sharing may occur at various levels (silicon die, board, datacenter rack, etc.), we consider the case where the FPGA and CPU share the same package and the on-board voltage regulator, common to a variety of SoC platforms [1], [2]. We assume an adversary with software access to the device who can run a piece of code on the processor cores and modify a part of, or the entire FPGA hardware for a limited time. Additionally, we consider CPUs equipped with frequency scaling capabilities, accessible to the victim (for performance or power efficiency) or the attacker (who has gained access to them, legitimately or not [4]). Our victim runs an application on one of the CPU cores, while the attacker aims for a DoS or a fault-injection attack. As shown in Fig. 1, the key novel component of our attack is the use of FPGA PL and, in particular, the power-wasting circuits implemented in it. This threat is different from the traditional attacks, which considered the overclocking and undervolting limits of the CPU only, ignoring the impact the PL can have on them. We do not make any assumptions on the activity or use of other cores in the system—they may be attacker-controlled, victim-controlled, or running a completely unrelated task.

IV. System Design

To investigate the possibility of power wasters in the PL injecting faults into the software running on the CPU, we chose the Genesys-ZU development board, equipped with a Zynq UltraScale+ MPSoC [18]. This system-on-chip contains a quad-core ARM A53 application processing unit (APU), a dual-core R5 real-time processing unit (RPU), a GPU, and an XCZU3EG FPGA [1]. All four SoC components are supplied by the same voltage source of 0.85 V. We made no modifications to the power distribution network, including not removing a single decoupling capacitor.

A. *Programmable Logic*

For our proof-of-concept, we consider that the adversary can control the entire FPGA for a limited time (i.e., for the duration of the attack). To maximize the power drawn from the FPGA, we used combinational ring oscillators [6]. For better control and flexibility of the attack, we divided all ROs that the attacker deploys into groups of equal size (which we refer to as *nodes*), composed of smaller *blocks* of ROs, as shown in Fig. 2. In our setup, there were 15 nodes, each containing $N_{\mathrm{BLOCK}} = 16$ blocks of $N_{\mathrm{RO}} = 500$ ring oscillators. From software, the attacker can decide how many blocks, per node, are to be enabled during the attack. Additionally, the timing of the activation of each node is controlled independently. In the PL, the control signals are converted to enable signals for each node, including the start and the end of the attack, and the choice of the duty cycle and the period of the enable signals of the nodes. To observe the effects of the various RO configurations and enable signal timing parameters on

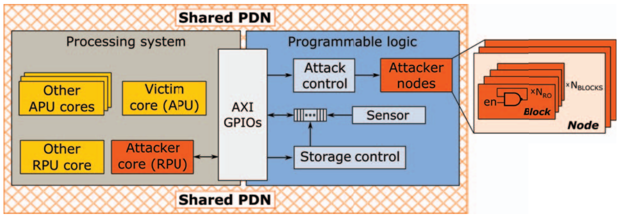

Fig. 2. Block diagram of the evaluation setup.

the voltage drop, we implemented a delay-line based voltage sensor in the PL [19]. The readings from this sensor during the attack are stored in on-chip memory, to be later sent to the processing system (PS) for the analysis. The PL operates at 100 MHz clock frequency.

B. Processing System

Given that our target platform does not offer voltage scaling (available on other platforms [20]), we only consider frequency scaling. When the frequency is changed, all APU cores are affected. Because the default frequencies of the APU and RPU cores are 1.2 GHz and 500 MHz, respectively, we chose to assign the victim to an APU core and the attacker to an RPU core (Fig. 2). In that configuration, the victim has the possibility of increasing the operating frequency (e.g., for improving the performance) as long as its core remains least affected by the frequency change. The PS and PL communicate through AXI GPIOs. We ran all codes on bare metal, leaving the study of the effects of running an operating system for future work.

We designed two attack modes: The first served to sweep all the parameters of the attack and record the corresponding sensor readings. This mode helped us find the configuration that results in the most effective (i.e., the largest and longest-lasting) voltage drop. In this mode, we ran each attack configuration once and paused for 50 ms before testing the next one. The second attack mode was more targeted as it used the best attack configuration found with the first mode. This mode was used for the fault injection. When the attack starts, the number of blocks is fixed to the maximum (in our case, 16) and the duty cycle is fixed. For each attack run, we swept the attack period within the range determined with the first mode. Between each period value and the next, we paused for 20 ms. We repeated the attack for five times, with a pause of 120 ms between each two consecutive runs (to allow the PDN to recover and reduce the likelihood of the board resetting).

V. Experimental Evaluation

With the aim of discovering whether injecting faults into the operation of the PS cores from the PL is possible, we carried out various experiments. Our evaluation aimed to find the attacker parameters that would result in both significant as well as long-lasting voltage drop, to mimic what DVFS-based exploits are capable of [4], [5]. We also identified the operating frequency limits of the CPU, to understand whether overclocking would be a realistic option. Finally, we experimented with two target victim applications. We discuss each of these experiments in the following subsections.

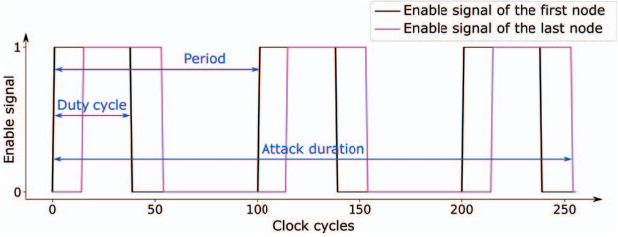

Fig. 3. Enable signals controlling the activation of attacker nodes and their corresponding parameters: period, duty cycle, and total attack duration.

A. Attacker Parameters Sweep

In our implementation, the adversary can control:

- the start of the attack (the trigger),
- the number of active blocks N_{BLOCKS},
- the total duration of the attack,
- the period of the enable signal, and
- the duty cycle (DC) of the enable signal.

The start of the attack is controlled from the PS, allowing the PS to repeat the attack. The attack duration (i.e., the time during which we activate the ROs according to the period and duty cycle parameters provided by the adversary) can be hardened into the design or controlled from the PS. As shown in Fig. 3, one attack period corresponds to the time between two subsequent activations of one attacker node. The duty cycle parameter sets the number of clock cycles, within one attack period, for which the enable signal remains active. As shown in previous work [6], [9], [21], the frequency of the activation signal matters; if it is close to the resonance frequency of the board PDN, the obtained voltage drop is more significant and, hence, fault injection is more likely. Similarly, the duty cycle affects the obtained voltage drop.

Given that previous DVFS-based attacks had precise control over either the voltage or the frequency, the undervolting or the overclocking of the victim CPU could last for as long as the adversary desired [3], [4]. To try to achieve an effect similar to persistent undervolting with the FPGA logic resources, we needed to first find the attack configuration that would not only result in a low voltage, but also ensure that the voltage remains low as long as possible. Therefore, we let the attacker implement the ROs in as many LUTs as possible. Due to the relatively small size of the programmable logic in our target device, this resulted in 60,963 look-up tables (LUTs) occupied by the ROs (or 86.4% of all the available LUTs).

Experimentally, we observed that the longer the attack duration, the more likely it was that the board would reset (DoS attack) before a fault could be injected. Therefore, we limited the attack duration to 256 clock cycles (equivalent to 2.56 µs). Then, we studied whether the activation pattern

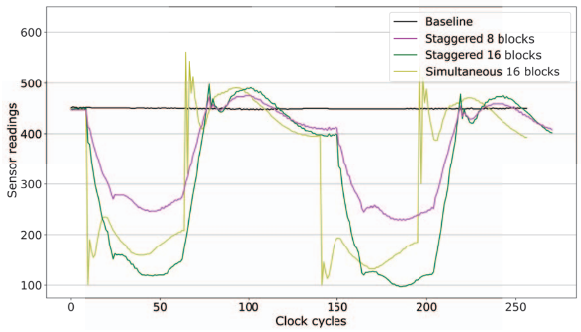

Fig. 4. Delay-line sensor readings for simultaneous and staggered activation of eight and 16 attacker nodes.

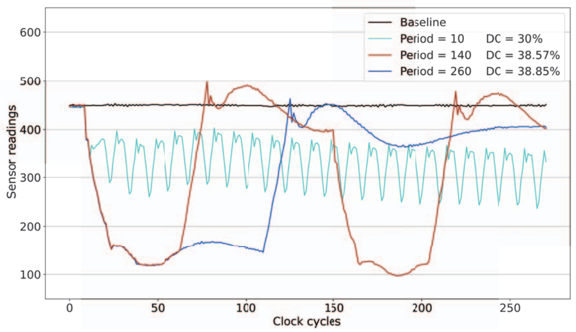

Fig. 5. Delay-line sensor readings for a selection of the attack parameters, highlighting the effects of various periods of the enable signal.

of the attacker nodes would have an impact on how long the voltage drop lasted. To that end, we experimented with the following activation patterns:

- *simultaneous* (de)activation of all the nodes and
- *staggered* (de)activation of the nodes.

We found that staggering the activation of the attacker nodes—by activating one additional node at each subsequent clock cycle—gave the most promising results. As shown in Fig. 4, the simultaneous activation resulted in an immediately large drop, which lasted for a very short time. Staggering the activation instead achieved a more persistent voltage drop, with a slightly reduced magnitude. We then swept the number of active blocks, with various values for the attack period (between 10 and 256 clock cycles), and tested various duty cycles. As shown in Fig. 4, the larger the number of blocks used, the more significant the voltage drop was and, hence, we chose the maximum number of blocks. Based on the results of the sweep, when the duty cycle was higher than 50%, the reset would be more likely to occur. Therefore, we chose to focus on duty cycles between 30% and 40%.

Fig. 5 shows a sample of the sensor readings for the maximum number of RO blocks for three different attack periods, each with a duty cycle within the 30–40% range. We can observe that, when the frequency of the enable signal was very high, the voltage drop was not as significant as when the frequency is lower, because the power draw did not last long enough. Lower frequency resulted in a more substantial voltage drop, but only up to a certain point. Past that point, an increase in the attack period meant that we lost the extra drop resulting from the repetition of the RO activation within the attack duration (period of 260 vs. period of 140 in Fig. 5). Given the results, we selected the attack parameters summarized in Table I for our next step: the fault-injection attack. These parameters resulted in a minimum observed voltage of 0.68 V for the PS (comparable to previous work [4]), and 0.642 V for the PL, as reported from the on-chip system monitor.

TABLE I
TARGETED ATTACK PARAMETERS.

	N_{BLOCKS}	Duration (clock cycles)	Period (clock cycles)	Duty Cycle	Activation
Sweep	0–16	128–16384	10–2200	10–50%	Sim. or Stag.
Chosen	**16**	**256**	**80–110**	**39%**	**Stag.**

B. CPU Operating Frequency Sweep

The default and the maximum operating frequency of the APU cores are 1.2 GHz and 1.5 GHz, respectively [18]. However, the system allows the operating frequency to be programmably increased, by reconfiguring the multipliers of the phase-locked loop (PLL) that generates the APU clock. We found that the APU operates correctly at frequencies up to 1.92 GHz; at higher frequencies, the processor stopped responding. The goal of the frequency sweep was to experimentally find the maximum operating frequency of the APUs within the frequency range 1.2–1.92 GHz for two target victim applications: The first is the multiplication code, inspired by the proof-of-concept from Plundervolt [5] (the code is given in Listing 1). The second is the AES encryption from the Wolfcrypt library [22]. Finding the maximum frequency allowed us to understand whether overclocking would be a valid option for the victim.

We ran the victims in a loop, assigning each of them to a random core, while changing the APU frequency between 1.2 GHz and 1.92 GHz, in steps of 15 MHz (the minimum value of the step supported by the PLL). In every experiment, we checked whether a fault or any other effects had occurred (e.g., loss of communication with the APU). We repeated the experiments with the remaining three APU cores idle or busy. To render the cores busy, we let them perform multiplication within a loop similar to the code in Listing 1, with the exception that we let the multiplication run during the entire experiment. We found that the maximum operating frequency for the two victim applications was above 1.9 GHz when the other cores were idle or busy, as shown in Fig. 6 and Fig. 7. Therefore, there exists a real possibility and little risk for the victim to overclock the design for better performance.

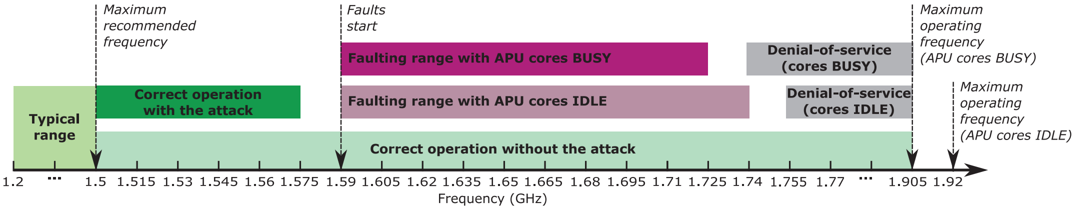

Fig. 6. Operating and faulting frequency ranges with the attack, for the multiplication victim code shown in Listing 1.

```
while (var == expected_result
        && count <= attack_duration) {
  var = initial_value;
  var *= multiplier;
  count++;
}
```

Listing 1: Multiplication victim code [5].

C. Fault Injection Attack: Multiplication

We started testing the fault injection by using the multiplication code shown in Listing 1 [5]. As the path for multiplication (implemented as a multiply-add instruction on the APU) is expected to be longer than that of simple arithmetic and logic operations, and as multiplication is used in many real applications, it constitutes a good test for our proof-of-concept. The victim code begins by initializing all required variables, including the multiplicand, the multiplier, and the attack counter. The main part of the code then takes place in a loop, from which the code should not exit unless a fault has occurred or the attack duration is over. Once the loop terminates, we read the counter and the multiplication result. If the counter value is smaller than the attack duration or the multiplication result is faulty, the program displays the faulty value and which bits are faulty.

As shown in Fig. 6, under normal operating conditions, the maximum frequency for the multiplication code was 1.92 GHz when all other APU cores were idle, and 1.905 GHz when they were busy. Therefore, we tested the attack at APU frequencies in the range 1.5–1.905 GHz, with other APU cores busy or idle. The results revealed that, regardless of the activity of other cores, the fault-injection attack was successful starting from 1.59 GHz. We observed up to six faulty bits in the multiplication result, depending on the choice of multiplicands and the APU frequency. Increasing the frequency beyond a certain point, with the attack active, resulted in loss of communication with the victim core. At the same time, the remaining parts of the system did not reset and the communication with the attacker core was maintained. This DoS from the victim APU was observed starting from 1.755 GHz when the other cores were idle, and from 1.74 GHz when the other cores were busy.

D. Fault Injection Attack: AES Encryption

The second victim we investigated was encryption. If faulted, it can lead to serious security breaches. We chose AES encryption code from the Wolfcrypt library [22], executing on top of the Xilinx FreeRTOS. Similarly to the previous experiment, the code begins by initializing all the variables, in this case, the plaintext and the encryption key. We set the key and then, within a loop, we encrypt the plaintext. After each iteration, we compare the ciphertext with the expected value (obtained under normal operating conditions), and the counter with the attack duration. The loop terminates only if a fault has been detected or if the attack duration is exceeded.

The results, presented in Fig. 7, show that, in the absence of the attack, the AES encryption proceeded correctly up until 1.92 GHz. With the attack active, we started observing faults at 1.65 GHz, when all other cores were idle, and at 1.635 GHz, when all other cores were busy. However, unlike with the multiplication code, where we had a large window of frequencies where fault injection succeeded, in the case of the AES, fault injection was successful at only two frequency points for each configuration (1.65–1.665 GHz and 1.635–1.65 GHz). At higher frequencies, we observed a DoS due to the loss of communication with the victim core. If the timing of the fault injection changed, for the obtained faulting frequency range, we found that the injected fault sent the code into an infinite loop, resulting in a DoS attack.

Tracing the location of the injected faults, which were not exploitable for breaking the encryption, we found that they occurred in XMEMCPY (the old value was copied instead of the new value) or in ByteReverseWord32 (the entire data word was reset to zero). These functions are called twice in each encryption, once with the input plaintext (to copy it into the function's local variables and to reverse the bytes order for endianness) and once with the ciphertext. This suggests that our attack, instead of faulting the AES computation, worked directly on the plaintext or the ciphertext. Such effects are useful for a DoS attack, and provide insight into which instructions are more vulnerable to undervolting. Moreover, targeting AES within an operation scheme such as cipher block chaining, the ability to manipulate the inputs (plaintext and initialization vector) can lead to compromised security [23].

VI. Discussion and Future Work

The results presented in Section V show that power wasters in the FPGA programmable logic are capable of affecting the CPU software execution, when the CPU and the FPGA share the PDN. In this section, we discuss some limitations of the discovered exploit and future work.

In this work, we have explored many attack parameters. However, for the fault injection exploration against AES and

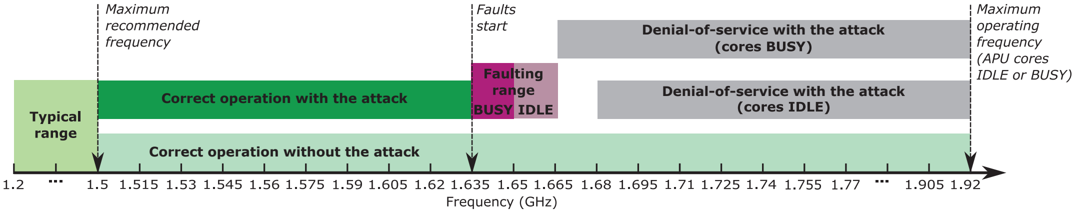

Fig. 7. Operating and faulting frequency ranges with the attack, for the AES encryption from wolfCrypt library [22].

multiplication, we fixed the size of the attacker (in our case, the maximum occupancy of the LUTs in the PL). A reduced size would result in shifting the fault injection interval to higher frequencies. Therefore, we believe that interesting future work could be to automate the selection of the attack parameters. At higher frequencies, the attacker can opt for a smaller number of power wasters to reduce the likelihood of a DoS attack. Furthermore, a characterization of various instruction types and their vulnerability to the undervolting can allow for a more targeted attack, which may result in exploitable faults for the AES. This understanding can then be combined with side-channel analysis to carefully control the timing of the fault injection [4]. The adversary can use cache side channels or, given the availability of the FPGA, implement voltage sensors in the PL [17].

Optimizing the attack and fully understanding its capabilities and limitations will then allow for designing comprehensive countermeasures. For instance, on-board sensors could be tested for detecting voltage drops and preventing the fault injection (e.g., by resetting the board, clearing the PL, or changing the voltage or the frequency). Decreasing side-channel leakage and countermeasures against cache-based side-channels and FPGA voltage sensors can also prove useful for reducing the attacker capabilities. Software defenses in critical parts of the code would also be a possibility [5]. Finally, investigating the attack on multiple platforms, potentially with different power wasters [13], [14], or even with stressor codes on the CPU [12] are interesting future research directions.

VII. Conclusion

In this work, we presented the first proof-of-concept for undervolting-based fault injection from the programmable logic of an FPGA to the software executing on a processing system in the same SoC. We have investigated the operating limits of the tested platform and shown that overclocking is a possibility. We have also explored the effects of various attack parameters on the voltage drop and duration. Using the best attack parameters found, our attack resulted in the successful injection of faults into a multiplication code and AES encryption. We managed to inject up to six single-bit faults in a single multiplication, for frequencies in the range 1.59–1.74 GHz. Our exploit also injected faults into the inputs of the AES encryption. Future work will focus on automating the attack and combining it with side-channels for a better controlled fault injection.

References

[1] "Zynq UltraScale+ MPSoC," Xilinx. [Online]. Available: xilinx.com

[2] "Cyclone V hard processor system technical reference manual," Intel, 2020. [Online]. Available: intel.com

[3] A. Tang, S. Sethumadhavan, and S. Stolfo, "CLKSCREW: Exposing the perils of security-oblivious energy management," in *USENIX*, 2017.

[4] P. Qiu, D. Wang, Y. Lyu, and G. Qu, "VoltJockey: Breaching TrustZone by software-controlled voltage manipulation over multi-core frequencies," in *CCS*, 2019.

[5] K. Murdock, D. Oswald, F. D. Garcia, J. Van Bulck, D. Gruss, and F. Piessens, "Plundervolt: Software-based fault injection attacks against Intel SGX," in *S&P*, 2020.

[6] D. R. E. Gnad, F. Oboril, and M. B. Tahoori, "Voltage drop-based fault attacks on FPGAs using valid bitstreams," in *FPL*, 2017.

[7] K. Matas, T. M. La, K. D. Pham, and D. Koch, "Power-hammering through glitch amplification – attacks and mitigation," in *FCCM*, 2020.

[8] J. Krautter, D. R. E. Gnad, and M. B. Tahoori, "FPGAhammer: Remote voltage fault attacks on shared FPGAs, suitable for DFA on AES," *TCHES*, vol. 2018, no. 3, 2018.

[9] D. Mahmoud and M. Stojilović, "Timing violation induced faults in multi-tenant FPGAs," in *DATE*, 2019.

[10] D. G. Mahmoud, W. Hu, and M. Stojilović, "X-attack: Remote activation of satisfiability don't-care hardware Trojans on shared FPGAs," in *FPL*, 2020.

[11] Z. Kenjar, T. Frassetto, D. Gens, M. Franz, and A.-R. Sadeghi, "V0LTpwn: Attacking x86 processor integrity from software," in *USENIX*, 2020.

[12] Y. Kim and L. K. John, "Automated di/dt stressmark generation for microprocessor power delivery networks," in *ISLPED*, 2011.

[13] M. M. Alam, S. Tajik, F. Ganji, M. Tehranipoor, and D. Forte, "RAM-Jam: Remote temperature and voltage fault attack on FPGAs using memory collisions," in *FDTC*, 2019.

[14] 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.

[15] "FPGA-based Amazon EC2 F1 computing instances," Amazon AWS. [Online]. Available: aws.amazon.com/ec2/instance-types/f1/

[16] I. Giechaskiel, K. B. Rasmussen, and J. Szefer, "C3APSULe: Cross-FPGA covert-channel attacks through power supply unit leakage," in *S&P*, 2020.

[17] M. Zhao and G. E. Suh, "FPGA-based remote power side-channel attacks," in *S&P*, 2018.

[18] "Genesys ZU: Zynq UltraScale+ MPSoC development board," Digilent. [Online]. Available: digilent.com

[19] K. M. Zick, M. Srivastav, W. Zhang, and M. French, "Sensing nanosecond-scale voltage attacks and natural transients in FPGAs," in *FPGA*, 2013.

[20] B. Salami, E. B. Onural, I. E. Yuksel, F. Koc, O. Ergin, A. C. Kestelman, O. Unsal, H. Sarbazi-Azad, and O. Mutlu, "An experimental study of reduced-voltage operation in modern FPGAs for neural network acceleration," in *DSN*, 2020.

[21] H. Zhu, X. Guo, Y. Jin, and X. Zhang, "PowerScout: A security-oriented power delivery network modeling framework for cross-domain side-channel analysis," in *AsianHOST*, 2020.

[22] "wolfCrypt embedded crypto engine," WolfSSL. [Online]. Available: https://www.wolfssl.com/products/wolfcrypt-2/

[23] S. Vaudenay and D. Vizár, "Under pressure: Security of Caesar candidates beyond their guarantees," Cryptology ePrint Archive, Report 2017/1147, 2017.