Voltage Drop-based Fault Attacks on FPGAs using Valid Bitstreams

Dennis R.E. Gnad, Fabian Oboril and Mehdi B. Tahoori
Institute of Computer Engineering, Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany
Email: {FirstName.LastName}@kit.edu

***Abstract*—Due to the widespread use of FPGAs in many critical application domains, their security is of high concern. In recent systems, such as FPGAs in the Cloud or in Systems-on-Chip (SoCs), users can gain access, even remotely, to the reconfigurable fabric to implement custom accelerators. This access can expose new security vulnerabilities in the entire system through malicious use of the FPGA fabric. In the past, attacks on the power supply level required local access to the hardware. In this paper, we reveal a security vulnerability in FPGAs that allows a valid configuration to generate severe voltage fluctuations, which crashes the FPGA within a few microseconds. Moreover, the extent of this crash is so severe, that manual power-cycling is required to be able to access and use the system again. This vulnerability has been systematically exploited in two different generations of FPGAs, and a SoC containing an FPGA. Because this vulnerability can lead to severe security attacks in systems using FPGA-based accelerators, we also analyze its underlying mechanism, and discuss possibilities for mitigation.**

I. Introduction

FPGAs are becoming increasingly attractive for user-defined accelerators in various systems, for instance through the OpenCL programming API [1], and many SoCs already contain FPGA fabric. They promise increased computing efficiency in the Internet of Things (IoT) for both ends: connected low-cost devices and high-end servers [2]. On the other hand, FPGAs are already widely employed in network appliances as well as many other critical domains such as military [3] or aerospace [4]. From these trends, new security vulnerabilities on FPGAs emerge and need to be analyzed, otherwise the entire system security may get jeopardized.

Previous attacks specific to FPGAs required local system access. These attacks were carried out either on the *physical level* [5–7], e.g. sensing of electrical current and power consumption, or on the *logical level*, for instance injecting trojan logic into existing bitstreams [8]. New and emerging usage scenarios, such as FPGAs in the Cloud (e.g. Amazon EC2 F1 [9]) or SoC-integrated FPGAs with user-accelerators require logic to be programmed in an untrusted way (i.e. from the user or application-side). This means that the usage of FPGA fabric by different users, trusted or untrusted, can be *remote* or *partial*, i.e., using only a fraction of FPGA fabric for each user application. This can open new security vulnerabilities that did not exist in previous application scenarios, where only trusted IP core providers could gain access through encrypted bitstreams supplied by the vendor.

Xilinx offers an *Isolation Design Flow* to secure less trustful IP cores [10]. On the logical level it provides isolation schemes to guarantee only trusted routing among different cores. However, isolation on the electrical level is not considered in this design flow. This way there can still be vulnerabilities from interactions on the electrical level, introduced by access using valid bitstreams.

In this paper, we expose such types of security vulnerabilities on FPGAs at the electrical level. We show a denial-of-service (DoS) threat that only requires partial access to the configuration of the FPGA, which can even be gained remotely. A vulnerability exists due to the possibility of generating excessive voltage fluctuations using valid bitstreams, and can cause a crash of the device within a short time. In the worst-case, the device will not reset itself, being a permanent DoS, until the FPGA or SoC is manually power cycled (power turned off and on). This can cause severe consequences, if the FPGA is used in servers.

So far, attacks on the electrical level have been carried out locally with access to the chip itself, for instance by manipulating its power supply or clock [7, 11]. Initial experiments showed that voltage drops can be generated from inside the system [12], but it has not been further evaluated. Older generations of FPGAs could be destroyed by overheating [13]. For modern FPGAs, overheat protection exists. The security vulnerability we expose in this paper is more effective and malicious than simple overheating. Our proposed attack uses a relatively small number of ring oscillators (ROs) to crash the FPGA (about 12% of total available LUTs used). Moreover, it just requires a few microseconds to crash the system, and thereby can escape various monitoring schemes. Additionally, it is not detectable by a thermal sensor, and finally, it keeps the FPGA inaccessible until it is fully power-cycled. This has been shown successfully in two different generations of FPGAs and could crash a full SoC, containing two processor cores with FPGA fabric, only using the vulnerability of this fabric.

The revealed vulnerability can be exploited for attacks with the goal to cause a DoS, affecting other users and applications. The focus of this paper is to fully analyze this vulnerability and its overall impact. We analyze the exact conditions under which the system crashes, and how systematically it can be reproduced. Furthermore, we provide a first analysis and guideline for possible mitigation of this vulnerability.

Threat Model:

The threat model we follow in this paper considers an adversary with complete or partial access to reconfigurable fabric used in two different systems and usage scenarios. The adversary goals are to cause a denial of service in the system.

One of the systems is a cloud-based environment, in which FPGA accelerators can be used by the adversary, another is a SoC with an embedded FPGA in which the programmable logic can be used (e.g. third party applications).

In a cloud-based environment, a DoS can make the FPGA unusable until manually power cycled, which is unlikely automated. In the case of a SoC with integrated FPGA, it has a similar effect, however if the system is battery-powered, the battery might need to be disconnected from the circuit. If it can not, the system stays in a permanent DoS.

Outline:

The remaining part of this paper is structured as follows: Section II introduces required background knowledge and related work. Section III explains the design of the required circuits for sensing and causing voltage drop and their experimental setup. Section IV discusses the results and how these attacks could potentially be mitigated. Finally, we conclude the paper in Section V.

II. Background

The security vulnerability analyzed in this paper intersects between three areas, which are *voltage fluctuations* on the electrical level, *security threats*, specifically on FPGAs and the *detection and protection* against power-supply related issues.

A. Nature of Voltage Fluctuations and Emergencies

FPGA chips use power distribution networks (PDNs) similar to other integrated circuits. A PDN includes the power delivery from the external power regulator all the way down to the transistors in the circuit. Sometimes PDN is used to only describe part of this network, like the internal grid of power and ground wires and decoupling capacitors. These PDNs are usually modeled as RLC networks. The voltage in these networks is in one part dependent on the current and inductance by Ldi/dt, whereas a steady-state load effect can be observed from resistive parts of the PDN and average current as an IR drop. This results in $V_{drop} = IR + Ldi/dt$. The IR drop has a lower effect in recent technology generations [14, 15]. Therefore, the main concern has become the voltage drop from a change in current (di/dt), which is harder to estimate due to the additional component of time, and might vary within the chip. Even more, it can lead to a resonant behavior between inductive and capacitive components, dependent on which frequency it is stressed.

Voltage drop will increase transistor delays $t_d \propto \frac{1}{V_{dd} - V_{drop}}$ [14, 16]. If an overall path is affected by a voltage drop for a long enough time (min. threshold time T_t) and high enough amplitude (V_{drop}), timing faults occur. Such a voltage drop is then called a *voltage emergency* [17]. In addition to timing faults, SRAM bit cells could lose data when their static noise margin voltage is violated [18].

The differences in the electrical current is influenced by both spatial and temporal circuit switching activity, which in turn is dependent on workload characteristics and system behavior [15, 19–21]. Subsequently, a maliciously crafted configuration and respective switching activity could lead to corner cases with insufficient voltage stability and jeopardize security. Rising system complexity can lead to more corner cases, which are left from time-consuming and complex electrical-level design validation, and elevate these risks.

B. FPGA and Hardware Security Threats

The triad of information security goals consist of *Confidentiality*, *Integrity* and *Availability*. In the following, a short overview of hardware security threats is given that affect these goals, specifically with the focus on FPGAs.

Confidentiality demands to protect access to secret information. In FPGAs, secret data can be deduced by monitoring and analyzing power, voltage, temperature or other emanations that can leak cryptographic keys, found through cryptanalysis [5, 22, 23]. For ASICs, hardware trojans can be inserted by a contracted manufacturer [24], where on FPGAs they can additionally be injected into existing bitstreams later [8]. *Integrity* describes maintaining data such that it can only be modified in an authorized way. It can be compromised among other techniques with electromagnetic radiation or power supply manipulation to cause bit flips from timing faults or soft errors [25, 7]. *Availability* is the goal to keep systems at service when required, and is threatened by DoS attacks that crash or destroy FPGAs. A way to overload an FPGA can be by excessive heat, generated by synthesizing ROs in great numbers. Overheating could destroy early-generation FPGAs [13]. In modern FPGAs, overheat protection is commonly available, where after a forced cooldown, the usage of the chip can be resumed.

All these attacks require a physical access to the system. An exception is overheating which is on electrical level but triggered through bitstream access or partial reconfiguration. To investigate these threats, FPGA bitstream encryption and cracking are explored widely [6, 26]. Unfortunately, guarding bitstreams is only a solution for a trusted software and hardware development approach, where bitstreams need to be signed or encrypted before the system accepts them. As reconfigurable systems and user-defined custom accelerators are becoming more widely adopted in FPGA systems, this fully trusted development approach becomes infeasible.

C. Detecting and Protecting against Voltage Fluctuations

Multiple works have addressed detecting process, voltage, and temperature (PVT) fluctuations which come from operating conditions, manufacturing variance or deliberate attacks. All of these fluctuations will affect path delays, where voltage variation can change as fast as within nanoseconds [12]. Some works concentrated on sensing errors caused by timing violations such as the 'Razor' architecture [27]. After an error is detected, a rollback or similar strategy can recover the situation to guarantee further valid operation. Other approaches monitor delays of surrogate structures that correlate with critical path delay, such as distributed ROs or delay lines. For ASIC processors, they were used to adapt the clock rate when the sensor value is approaching a critical threshold, before an error can happen [28]. For FPGAs, voltage fluctuations were monitored to detect power supply-based fault attacks in [12].

III. Circuit Design and Experimental Setup

In this section we explain the experimental setup of our components used in the analysis and the circuit we designed

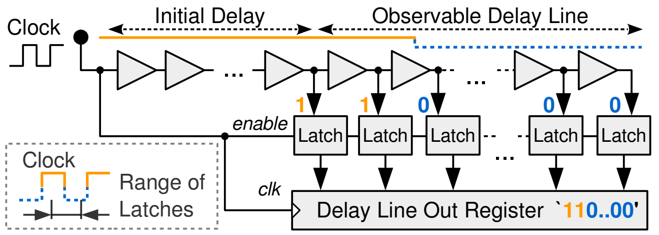


Fig. 1: Principle of a delay-chain based voltage fluctuation sensor, measuring propagation time of the clock.

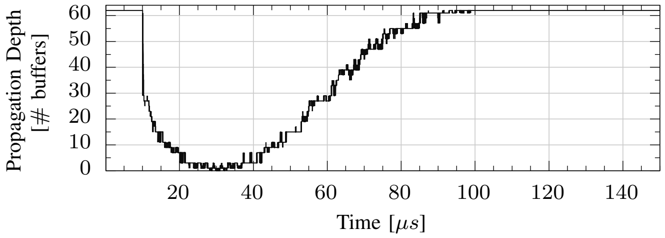


Fig. 2: Change in propagation depth into a chain of 'CARRY4' elements of a Xilinx Virtex 6 FPGA due to sudden current increase, caused by enabling 18720 ROs and keeping them enabled. Temperature 38–40 °C.

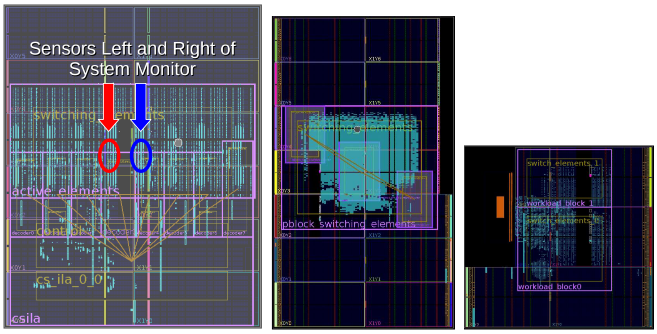


Fig. 3: **Left to Right:** Virtex 6 Floorplan with ROs and Sensors; Kintex 7 Floorplan with ROs; Zynq-7020 Floorplan with ROs.

to expose the vulnerability. First, we will explain the implementation of voltage fluctuation sensors in our FPGA in Section III-A. Then we will explain how voltage drops can be generated through ROs in Section III-B. Finally, we detail the rationale of the attack based on voltage emergencies and its experimental setup in Section III-C.

A. Delay Sensors for Voltage Fluctuation Analysis

To sense variation in delay, caused by voltage fluctuations, Zick et al. [12] show an approach using available FPGA primitives. In Fig. 1 we show the concept of this sensor. It shows a delay line, simplified as a chain of buffers. Between these buffers, latches are connected that are enabled with the same clock signal. This way, the latches measure how far the clock signal can propagate through this delay line, while they are enabled. Thus, during each 'high' period of the clock, the latches will take a 'snapshot' of how far the same clock could propagate through the buffers. An *Initial Delay* is used before the *Observable Delay Line*, that is not connected to latches in order to save area and effort to further process sensor data.

For FPGAs, other primitives are used to resemble the buffers, for which we use a 'CARRY4' primitive for each 4 buffers of the *Observable Delay Line* [12]. Thus, in an estimation of this sensor, for a Virtex 6 FPGA, one buffer equals $19.5ps$ in delay. To improve linearity of the output, we use a two-bit *bubble proof* priority encoder [29]. Due to this, the sensor used here can take on values in the range of $0-62$.

In this work, we clock the sensors at $100MHz$ (i.e. one sample every $10ns$). The sensor values are transferred from our FPGA board to the PC using the Xilinx Chipscope Integrated Logic Analyzer. It saves the sampling data within FPGA Block RAM (BRAM) and then transfers it to the PC.

We show the measurements from a sensor on the Virtex 6 FPGA, during a high voltage drop event in Fig. 2. This figure shows a trace of sensor values (y-Axis) over time (x-Axis). Such a sensor value will show the *propagation depth* into the chain of buffers, whose delay is affected by voltage fluctuations. Thus, higher values relate to faster buffers operating at a higher voltage, and as buffer delays increase from voltage drop, their propagation depth will decrease, shown as lower values on the y-Axis. Before the voltage drop within time $0-10\mu s$, the sensor is saturated at 62. At about $10\mu s$ the voltage starts to drop and the sensor shows that less bits can be propagated due to increased delay t_d of the transistors in the buffers. This voltage drop reaches its lowest point at $30\mu s$ and recovers back until $100\mu s$. It demonstrates how on-chip activity can lead to a strong increase in path delay. The delay is affected by the whole sensor range of the *Observable Delay Line*, corresponding to a change of about 12.5% of the total path (including the *Initial Delay*) according to timing analysis. In [12], 14% delay change was reported with a different method and FPGA. When sensors values are shown in the remaining work, they are based on two sensors, placed on the left or right side of the *System Monitor*, near the center of the FPGA, as shown in Fig. 3.

B. Generating Voltage Drops

We achieve the crash by causing timing faults or SRAM state retention loss through voltage emergencies. As explained in Section II, voltage drops are mainly dependent on changes in current and can therefore be caused deliberately when a spontaneous high current is required. In the setup explained here, high switching activity with proper timing is used to cause this excessive current and the resulting load on the PDN. LUTs are configured as inverters, and their input is connected to their output in a loop, forming ROs that can toggle very fast (i.e. as fast as physical transistor delays allow). By also configuring an AND in the LUT, an additional input 'I1' is used as an 'enable' signal to enable or disable the RO oscillation as shown in Fig. 4. Standard FPGA tools can be used to implement these ROs.

By suddenly enabling all ROs through the LUTs 'I1'-inputs, they cause a strong influence on the voltage stability in the whole system, resulting in delay increases, especially in nearby paths. We showed this effect of voltage fluctuations on path delay in Fig. 2, where within a few nanoseconds a steep drop can already be observed in a nearby sensor. Potential thermal influence is negligible in this observation, as with the low amount of ROs used in our experiments, temperature increases much slower and requires several seconds to minutes for a significant increase in heat. This means that such voltage

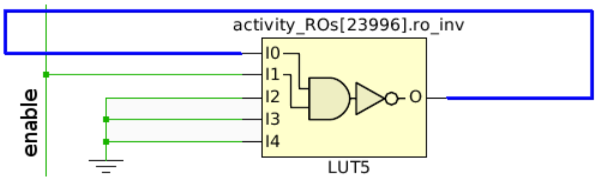


Fig. 4: Single ring oscillator made out of one LUT5 primitive

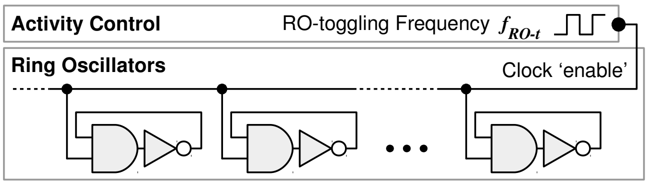


Fig. 5: Principle of connecting ROs to a clock f_{RO-t} that toggles their allowed activity (inside which they toggle themselves as fast as physically possible).

drops cannot be detected by on-chip thermal sensors or slower integrated voltage monitoring circuitry.

C. Voltage Drop-based Fault Attack

Voltage drops depend on various other design and runtime parameters, such as process variation, local temperature and generic voltage noise. To cause a voltage emergency, an isolated voltage drop might be insufficient to create the required load on the PDN. Therefore we have to consider spatial spread and plan a series of various voltage drops carefully timed with respect to each other.

Consequently, we enable and disable the ROs to create voltage drop *pulses* in rapid succession, meaning they oscillate for a certain time and are deactivated in another, which we call the frequency of RO-toggling f_{RO-t}. This principle is depicted in Fig. 5. Please note that this frequency of activation f_{RO-t} is different from the inherent switching frequencies of the ROs themselves. By varying f_{RO-t}, a reproducible crash of the FPGA can be caused. Please note too, besides this pulse frequencies, also the placement and amount of activated ROs has an influence on the attack quality. However, we observed that this influence is much lower than the pulse frequency, if a sufficient amount of ROs is available (when about >7% of the LUTs are used, tested on the ML605 board). Hence, in the following, we limited the description of our analysis to a fixed number of ROs, for the sake of brevity.

When testing different frequencies for f_{RO-t}, one needs to take into account that the worst critical situation takes some time to build up. When simply enabling the ROs, on the ML605 board, it takes about $10 - 20\mu s$ from the start of the path delay increase until it reaches the lowest point and starts to recover again, as shown in Fig. 2. That means, we should keep the ROs oscillating until they cause a high enough voltage drop, and start over again, before voltage starts to recover. Due to that, reducing f_{RO-t} will crash the FPGA more effectively, until the range of recovery ($< 25 - 50$ kHz). To reduce the thermal influence on our experiments, we keep the FPGA within 38-40 °C using the on-board fan.

We also checked if timing failures could be caused when ROs are added to a legitimate design. Our investigation showed that it either operated correctly, or crashed entirely. Follow-up work is required to do a more detailed analysis.

IV. Results and Discussion

The devices tested in this paper are a 40nm Xilinx Virtex 6 on ML605 board (37,680 slices×4 LUTs), a 28nm Kintex 7 on KC705 board (50,950 slices×4 LUTs), and a 28nm Zynq 7020 on Zedboard (Dual ARM Cortex-A9 + 53,200 LUTs).

In the following subsections we will expose the previous mentioned vulnerability that can be exploited for DoS in these devices. We will first analyze the conditions required for crashing in Section IV-A, then we try to explain the root cause in Section IV-B. In Section IV-C we then show if detecting a voltage drop with the delay line-based sensors is fast enough to prevent it, and discuss different possibilities to stop the activity that leads to the crash.

A. Crash Conditions

All three boards do not reliably crash when using a single voltage drop. But we can crash all three boards with an automated sweep through different f_{RO-t}, confirmed by a drop in measured power consumption on the board, and the stop of blinking LEDs controlled by the FPGA.

Of the three boards, the ML605 and KC705 always stop being accessible after the crash, and consume less power, which is due to a locked-up voltage regulator for at least the ML605. Thus, restarting the PC while keeping the board running does not resolve the situation. A manual power-cycle of the FPGA board is required before any access (i.e. JTAG) is possible again. On the KC705 we also tried to by-pass the on-board USB JTAG with a stand-alone JTAG dongle, which showed to be not sufficient to reactivate and access the FPGA board again.

The Zedboard has one of three conditions occur randomly. In one of them it stops being accessible to JTAG, like the other boards. In another case, it resets, which also deconfigures the FPGA part and resets the ARM cores. In the last case, it looks like a reset as well. However, when trying to reprogram the Zynq in Vivado, it locks up in the middle of reprogramming the bitstream. The software has then to be forcefully terminated. After that, the SoC stays inaccessible like in the first condition.

If the ML605 is connected to Chipscope during the crash, we get the following message:

```
WARNING: System Monitor Die Temperature has invalid
data. [..] See Answer Record 24144.
```

Where a lookup of this *Answer Record* does not lead to relevant information. For the KC705 board, if connected to Vivado Hardware Manager, the software quits this program, and shows in a popup:

```
[Labtoolstcl 44-153] HW Target shutdown. Closing
Target: localhost:[..]
```

For the Zedboard, when connected to the Vivado Hardware Manager and crashes in any way (inaccessible or not), the crash is not immediately recognized and will only be seen when trying to reprogram the board as:

```
ERROR: [Labtools 27-3165] End of startup status: LOW
ERROR: [Common 17-39] 'program_hw_devices' failed
due to earlier errors.
```

If it is connected for debugging during the crash, it will immediately show by:

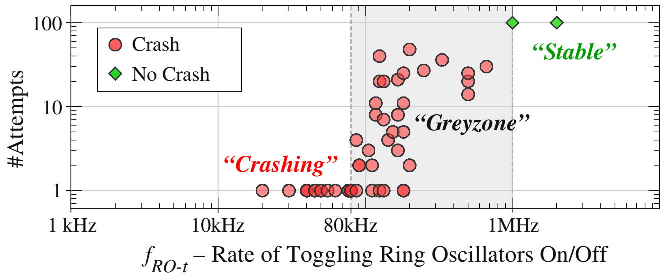


Fig. 6: Range of frequencies to toggle ROs on/off that cause the ML605 to crash after a certain number of attempts. Temperature regulated within 38-40 °C.

```
ERROR: [Xicom 50-38] xicom: Core access failed. [..]
ERROR: [Labtools 27-3176] hw_server failed [..]
Resolution: Check that the hw_server is [..]
```

If the ARM cores are on a debug connection in the Xilinx SDK, this connection is terminated.

In the following, we give more detailed results on the specific frequencies required for the crash on the ML605 board:

We activate 18720 ROs (about 12% of the LUTs in this specific Virtex 6) in a range of f_{RO-t} of 20 kHz$-$2 MHz. For each attempt to crash the FPGA, we program it and start an activity at a single f_{RO-t}. We then check how many attempts we require in order to crash the system. Since the recovery from a crash is time-consuming and not easily automated, we ran more experiments for the corner cases. The results in Fig. 6 show how many attempts are required to cause a crash, depending on the chosen f_{RO-t}. Above 1 MHz, the voltage does not drop enough, and no crash happens with at least 99% probability (based on 100 attempts). Within 80 kHz$-$1 MHz is a 'greyzone' in which the crash occurs in a non-deterministic way. For the f_{RO-t} tested below 80 kHz, the crash happens always at first try.

To check the maximum time the crash requires, we set the Xilinx Chipscope Integrated Logic Analyzer (ILA) to trigger on the start of our malicious activity. After the trigger condition, we set it to collect only 16 samples before sending them to the PC. The expected time required in total from the trigger condition until received by the PC is less than $150\mu s$, based on the JTAG frequency, data size and internal sampling rate. Thus, the crash happens in less than $150\mu s$.

We additionally experimented with the two FPGA boards being connected inside a workstation to PCIe, in which the crash also happened. By using on-board switches to power either of the two boards off and on, the ML605 can be accessed again. However, the KC705 requires a cold reboot of the workstation (i.e. even the workstation's power supply), which can be a permanent DoS in a server environment that requires manual power cycling.

Interesting for these two boards is their power consumption after the crash, which is less than in any other condition by at least 20%, because one voltage regulator stopped operating. However, for the Zedboard there is no power difference. Table I summarizes the conditions.

B. Attack Analysis

The vulnerability exposed in this work is caused through voltage emergencies. In the background section, we reviewed various failure causes due to timing faults, SRAM state retention loss, or resonance in the PDN that supplies the FPGA. For the FPGAs tested here, both internal BRAM and the configuration memory are based on SRAM. The required time T_t and amplitude V_{drop} for a voltage emergency are different for a timing fault or SRAM state retention loss, where power distribution networks can have weaknesses when stressed at the right frequency.

On the ML605 board, $0V$ can be measured for the FPGA core supply voltage *VCCINT* after the crash. This situation shows that the respective on-board voltage regulator was shut off and is causing the permanent nature of the crash until power cycling of the board. The other voltage regulators on the board still operate normally and keep some LEDs lighted up and the fan spinning.

The sensors mapped to the FPGA can show some more details and evidence why this situation occurs, which we show with some example traces in Fig. 7. These were collected when the system did not crash (therefore we can only show a minimum frequency of 83.333 kHz).

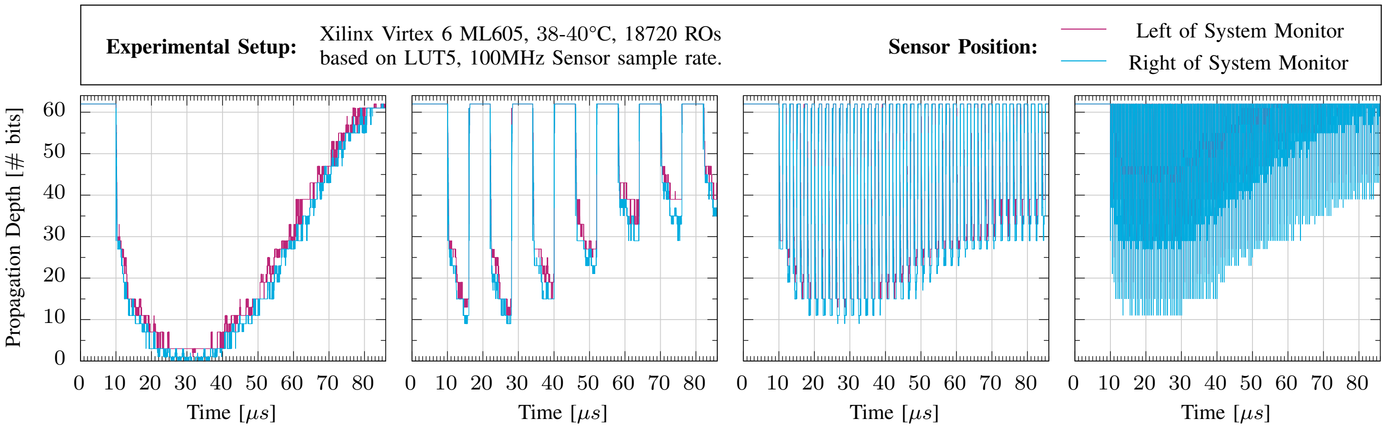
A larger V_{drop} can be seen when the ROs are activated suddenly and stay activated, as in Fig. 7a, which recover after around $80\mu s$. When we cause repetitive events however, we can see a certain amplitude of V_{drop} to repeat itself for a longer stress period, shown for $f_{RO-t} = 83.333$ kHz in Fig. 7b. When the frequency of f_{RO-t} is varied in Fig. 7b-7d the worst-case V_{drop} of each frequency monitored with the sensors is very similar. However, the 500 kHz shown in Fig. 7c does only rarely cause a crash, potentially because the sensors reach below a value of 10 only for very short times. With 2 MHz shown in Fig. 7d we have never got a crash, and in that case, none of the sensors reach below 10.

Thus, a possible conclusion is that the board crashes directly due to extreme stress from a frequency-dependent resonance in the on-chip PDN or on-board voltage regulator. However, other components, like the *System Monitor* or configuration memory, might first become faulty, and subsequently lead to the voltage regulator getting disabled. The requirements to cause voltage emergencies in these components might differ from each other, and with that also the sensitivity to different f_{RO-t} frequencies. For instance, high frequency but short-timed V_{drop} 'spikes' can be absorbed in longer T_t times. Since in the experimented boards the memory and logic subsystems of the FPGA use the same supply voltage (and likely same PDN), the excessive voltage drop in the logic part (ROs) can cause voltage emergencies in the memory subsystem as well.

To collect evidence for a retention failure of SRAM, we can check either BRAM or configuration memory. We use BRAM, as it is easier accessible in the fabric, and Chipscope actually uses it for its sample memory. Thus we can see its failure by receiving corrupted data in Chipscope. We received such corrupted data, when experimenting with f_{RO-t} within the non-deterministic greyzone. In that range, we can sometimes

TABLE I: Different conditions for the tested boards. Power consumption measured with wall plug, default power supplies.

Board	**% LUTs used for ROs**	**Standalone Power Consumption**				**Crash Recovery (Inaccesible)**	
		After Reset	After Flashing/Programming	All ROs Active	After Crash (Inaccessible)	PCIe Connected	Standalone
ML605	12.4%	14.3W	16.4W	36.5W	11.1W	Off/On Board	Off/On Board
KC705	11.8%	21.8W	13.5W	29.4W	7.0W	PC Power Off/On	Off/On Board
Zedboard	12.8%	3.3W	3.3W	6.2W	3.3W	not available	Off/On Board



(a) Suddenly activate all ROs. (b) ROs on/off at 83.333kHz. (c) ROs on/off at 500kHz. (d) ROs on/off at 2MHz.

Fig. 7: Influence on the increase in path delay when applying different activation frequencies f_{RO-t} to the ring oscillators. Case (b) almost always leads to a crash, because at 80μs it is not yet recovered as (a) and the voltage drops last longer than (c) or (d). For (d) it never crashes, probably because the value never goes below 10.

still receive partial traces, short before a crash, showing evidence of an anomalous BRAM behavior.

In Fig. 8 we show part of such a corrupted trace. In this trace until 131.66μs, we receive typical fluctuation data. After this time, all data bits received from Chipscope are '0' for exactly 150 ns, then they are '1' until the crash. In all the partial traces we recorded, the behavior is similar, and the intermittent '0's always appear for 150 ns on a systematic basis.

Please note, in the case of receiving all-'1', it is not the sensor being saturated again, as the sensor saturation value is 62 and not 63, but the reserved part in BRAM is 6 bit. We assume itself, or its output latches, got reset to '111111'= 63.

In conclusion, the permanent nature of the crash depends on the voltage regulator crashing or shutting down for safety reasons. This situation is in turn either caused directly by resonance in the PDN, or by causing other problems in the logic and configuration memory of the FPGA. More details will be analyzed in future work.

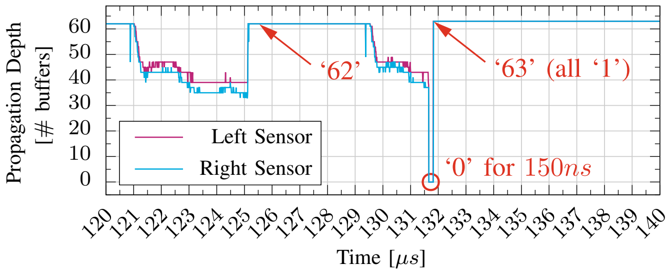


Fig. 8: Detail of path propagation depth when a crash happens, but some data was still transmitted. After the undershoot to all zero at 131.66μs, all data received by chipscope is '1' – potentially the reset state of BRAM output latches.

C. Discussion

In the previous sections, we showed the nature of deliberately caused voltage emergencies that lead to a DoS. Specifically, the attack leads to a DoS situation much quicker than by overheating, and with the addition of keeping the FPGA inaccessible, even to JTAG, until its power supply is reset.

Allowing user-configurable accelerators in such systems create security vulnerabilities that can compromise the availability of FPGA resources. A complete server might require reboot, including full power cycling. In the worst-case, a system based on SoCs with included FPGA can get into a permanent DoS, for instance when they are powered by a non-removable battery.

Threats like these could be reduced, given a scheme to detect and disable the excessive switching activity, before it escalates to voltage emergencies for the complete chip.

In an additional experiment, we show how fast voltage recovers when all ROs are disabled after a sensor value below '30' is detected. This way, we estimate the latency of the sensor and how fast the voltage drop wears off. Fig. 9 shows the sensor readings for this experiment. The sensor values show recovery after two samples (20ns). However, as the sensors are saturated at 62, the system might still require more time to fully recover.

Applying this option to a real design would require to reserve area for the sensors and one input on each LUT. The sensor thresholds would need to be chosen such that legitimate activity is not affected. Additionally, all LUTs would effectively have one input less available to the design,

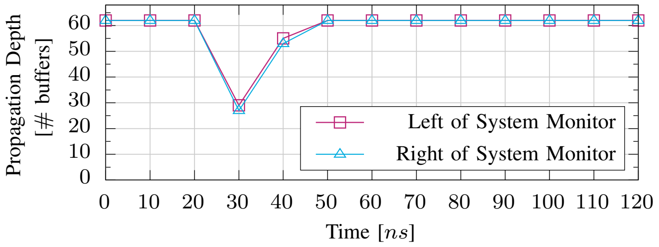


Fig. 9: Detail of path propagation depth when disabling the ring oscillators after detecting a drop in sensor value with previously idle activity.

and no option for a disable switch might exist for other FPGA primitives, making it rather infeasible and requiring other options These options could be based on power gating or a way to quickly disable the interconnects in affected areas.

To even prevent malicious bitstreams from loading, they can be sanity checked in software, with the challenge of keeping legitimate bitstreams working, but not leaving loopholes for malicious ones. One option that recent FPGA tools already use as default constraints is checking for combinational loops during bitstream generation, which can be deactivated by the user. Thus, the check would need to be done at a privileged system software level, inaccessible to the user or application developer.

For all of these possibilities, new experiments are required. To be able to deactivate arbitrary malicious circuits fast enough, new FPGAs might need to be manufactured.

V. Conclusion

FPGAs become more widely adopted as user-defined accelerators, such as in the cloud, or integrated in SoCs. In these new usage scenarios, classic ways to enforce security, like bitstream encryption, become infeasible, making them vulnerable to new security threats. In this paper, we revealed such a security vulnerability, by showing a systematic approach to crash two generations of FPGAs, and an SoC containing an FPGA, which can lead to a denial of service threat in these systems. This denial of service is caused by a specific configuration when bitstream-level access is given, and only requires about 12% of the available LUT-resources in the tested FPGAs. Such a vulnerability can allow attacks on FPGAs used in data centers, SoCs, and other application domains, where the entire system needs to reboot or even be fully disconnected from power in order to power cycle the crashed FPGAs. Additionally, we discuss how proper mitigation could be implemented to prevent denial of service.

References

[1] T. S. Czajkowski, U. Aydonat, D. Denisenko, et al. From OpenCL to high-performance hardware on FPGAs. In *FPL*, 2012.

[2] C. W. Fletcher, I. A. Lebedev, N. B. Asadi, D. R. Burke, and J. Wawrzynek. Bridging the GPGPU-FPGA Efficiency Gap. In *FPGA*, 2011.

[3] S. Skorobogatov and C. Woods. *Breakthrough Silicon Scanning Discovers Backdoor in Military Chip*. In *CHES*, 2012.

[4] B. Butka and R. Morley. Simultaneous Switching Noise and Safety Critical Airborne Hardware. In *Southeastcon*, 2009.

[5] S. B. Örs, E. Oswald, and B. Preneel. *Power-Analysis Attacks on an FPGA – First Experimental Results*. In *CHES*, 2003.

[6] A. Moradi, A. Barenghi, T. Kasper, and C. Paar. On the Vulnerability of FPGA Bitstream Encryption Against Power Analysis Attacks: Extracting Keys from Xilinx Virtex-II FPGAs. In *CCS*, 2011.

[7] L. Zussa, J. M. Dutertre, J. Clédière, and A. Tria. Power supply glitch induced faults on FPGA: An in-depth analysis of the injection mechanism. In *IOLTS*, 2013.

[8] R. S. Chakraborty, I. Saha, A. Palchaudhuri, and G. K. Naik. Hardware Trojan Insertion by Direct Modification of FPGA Configuration Bitstream. *IEEE Design & Test*, 2013.

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

[10] D. Corbett. The Xilinx Isolation Design Flow for Fault-Tolerant Systems, 2012.

[11] S. Endo, T. Sugawara, N. Homma, T. Aoki, and A. Satoh. An on-chip glitchy-clock generator for testing fault injection attacks. *J. Cryptogr. Eng.*, 2011.

[12] K. M. Zick, M. Srivastav, W. Zhang, and M. French. Sensing Nanosecond-scale Voltage Attacks and Natural Transients in FPGAs. In *FPGA*, 2013.

[13] I. Hadžić, S. Udani, and J. M. Smith. FPGA Viruses. In *FPL*, 1999.

[14] A. V. Mezhiba and E. G. Friedman. Scaling trends of on-chip power distribution noise. *Trans. VLSI Syst.*, 2004.

[15] S. Das, P. Whatmough, and D. Bull. Modeling and Characterization of the System-Level Power Delivery Network for a Dual-Core ARM Cortex-A57 Cluster in 28nm CMOS. In *ISLPED*, 2015.

[16] K. Arabi, R. Saleh, and X. Meng. Power Supply Noise in SoCs: Metrics, Management, and Measurement. *Des. Test. Comput.*, 2007.

[17] A. Muhtaroglu, G. Taylor, and T. Rahal-Arabi. On-die droop detector for analog sensing of power supply noise. *SSC*, 2004.

[18] E. Seevinck, F. J. List, and J. Lohstroh. Static-noise margin analysis of MOS SRAM cells. *SSC*, 1987.

[19] M. S. Gupta, J. L. Oatley, R. Joseph, G.-Y. Wei, and D. M. Brooks. Understanding voltage variations in chip multiprocessors using a distributed power-delivery network. In *DATE*, 2007.

[20] V. J. Reddi, S. Kanev, W. Kim, et al. Voltage Smoothing: Characterizing and Mitigating Voltage Noise in Production Processors via Software-Guided Thread Scheduling. In *MICRO-43*, 2010.

[21] D. R. E. Gnad, F. Oboril, S. Kiamehr, and M. B. Tahoori. Analysis of Transient Voltage Fluctuations in FPGAs. In *FPL*, 2016.

[22] M. Masoomi, M. Masoumi, and M. Ahmadian. A practical differential power analysis attack against an FPGA implementation of AES cryptosystem. In *Information Society (i-Society)*, 2010.

[23] T. Iakymchuk, M. Nikodem, and K. Kpa. Temperature-based covert channel in FPGA systems. In *ReCoSoC*, 2011.

[24] R. Karri, J. Rajendran, K. Rosenfeld, and M. Tehranipoor. Trustworthy Hardware: Identifying and Classifying Hardware Trojans. *Computer*, 2010.

[25] L. Sauvage, S. Guilley, and Y. Mathieu. Electromagnetic Radiations of FPGAs: High Spatial Resolution Cartography and Attack on a Cryptographic Module. *ACM Trans. Reconfigurable Technol. Syst.*, 2009.

[26] A. Bogdanov, A. Moradi, and T. Yalçin. Efficient and side-channel resistant authenticated encryption of FPGA bitstreams. In *ReConFig*, 2012.

[27] D. Ernst, N. S. Kim, S. Das, et al. Razor: a low-power pipeline based on circuit-level timing speculation. In *MICRO-36*, 2003.

[28] C. R. Lefurgy, A. J. Drake, M. S. Floyd, et al. Active Management of Timing Guardband to Save Energy in POWER7. In *MICRO-44*, 2011.

[29] J. Wu. Several Key Issues on Implementing Delay Line Based TDCs Using FPGAs. *IEEE Trans. Nucl. Sci.*, 2010.