Power-hammering through Glitch Amplification – Attacks and Mitigation

Kaspar Matas, Tuan Minh La, Khoa Dang Pham, and Dirk Koch
The University of Manchester, UK
{kaspar.matas,tuan.la,khoa.pham,dirk.koch}@manchester.ac.uk

***Abstract*—Recent work on FPGA hardware security showed a substantial potential risk through power-hammering, which uses high switching activity in order to create excessive dynamic power loads. Virtually all present power-hammering attack scenarios are based on some kind of ring oscillators for which mitigation strategies exist. In this paper, we use a different strategy to create excessive dynamic power consumption: glitch amplification. By carefully designing XOR trees, fast switching wires can be implemented that, while driving high fan-out nets, can draw enough power to crash an FPGA. In addition to the attack (which is crashing an Ultra96 board), we will present a scanner for detecting malicious glitch amplifying FPGA designs.**

I. Introduction

The rise of FPGAs used in cloud computing can be observed in recent years when major cloud vendors such as Amazon, Microsoft, Huawei, and Alibaba [1]–[4] started offering FPGA instances to customers. Meanwhile, multi-tenancy support for FPGA infrastructures in which more than one user can share the same FPGA resource is desirable and, hence, attracting research both in academia [5] and industry [6].

In FPGA cloud computing multi-tenant scenarios, system security is of paramount importance. Security concerns range from data privacy of users sharing the same FPGA resources to the availability of the system service itself [7]. Integrating FPGA resources into a cloud computing infrastructure is opening a new surface of attack at the electrical level, which is not available in the software world with CPUs and GPUs, and therefore, has not been well-studied yet. For example, a grid of ring oscillators can bring down an entire FPGA board that needs to be power-cycled to get back the normal operation [8]. Such an attack, known as *power-hammering*, is a specific threat for FPGAs due to their full low-level hardware programmability. Power-hammering is the process of creating excessive power, commonly with the intend to compromise integrity, confidentiality, or availability of a system.

Currently, all power-hammering circuits are built upon ring oscillators, which may or may not be detected by the vendor design rule checking (DRC) [9], [10]. However, the fundamental principle behind this class of attack is *to create a circuit with high switching activity that can consume as much power as possible to create voltage drops or to exceed the board power or thermal budget*. Therefore, as an alternative to ring oscillators, it is possible to use well-designed XOR trees to generate a significant amount of switching activities (a.k.a *glitches*), which currently are not flagged by DRCs.

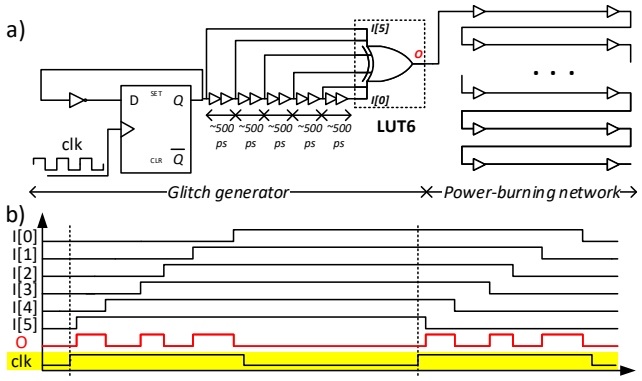

Fig. 1. Illustrations of the presented glitch amplification attack with (a) the attack circuit, and (b) the waveform got from the attack circuit shown.

With help from academic tools such as GoAhead [11] or RapidWright [12], we can fine-tune input delays inside XOR trees to achieve the desired toggling frequency. Vendor tools can do this just as well, but for this paper, we used GoAhead because it allows searching for multiple routing paths ranked by latency. While other logic functions may glitch as well, XOR is most effective as any change at any of the inputs creates a change at the XOR output (omitting possible canceling effects in real systems). In our research, the output of a *glitch generator* will be used to drive an extensive network of wires and combinatorial logic, which is acting as the *power-burning network*, as illustrated in Fig. 1. As we will show in Section IV, this allows creating a power-hammering attack at a small cost that can crash an FPGA board.

In this paper, for the first time, such malicious circuits, which can crash a Xilinx UltraScale+ FPGA board just by using a *glitch generator* and a *power-burning network*, are presented. The presented malicious circuits pass all Vivado DRCs (version 2019.1) for bitstream generation as well as all tests that are performed to deploy designs on Amazon Web Services F1 instances. It is necessary to highlight that the Power Estimation feature provided in the Vivado design tool is remarkably underestimating the potential power consumed by the proposed circuits. Hence the Xilinx vendor tools cannot currently prevent such an attack. As a mitigation for the attack, we have extended our open-source tool FPGADefender [13] by providing additional functions to detect possibly excessive switching activity. The test is performed directly on bitstreams generated by the Xilinx vendor tool.

The **threat model** considers an adversary with complete or

2576-2621/20/$31.00 ©2020 IEEE
DOI 10.1109/FCCM48280.2020.00018

partial access to FPGA fabric via full or partial reconfiguration. The adversary goals are to shut down the FPGA service of an FPGA-based system (e.g., embedded or cloud-based), and hence to cause a denial of service in the system or to manipulate system states by temporarily reducing supply voltage below safe operating conditions. The here presented attack can also serve as a template for FPGA hardware trojans in the sense that the attack requires only a little amount of logic, passes all existing DRCs but can crash a large amount of FPGA-based systems.

The contributions of this work are as follows:

- A new class of power-hammering attack based on glitch amplification that does not use oscillators (Section III);
- An extension to the tool FPGADefender to mitigate this new class of attack on FPGA boards (Section IV);
- Demonstration of the presented attack on the Ultra96 platform equipped with the Xilinx Zynq UltraScale+ MPSoC and evaluation of our mitigation strategy (Section V).

Moreover, we provide a brief literature review in Section II and conclude this work in Section VI.

II. Background and Related Work

There are two power sources in FPGAs: static and dynamic power. Dynamic power can be calculated using the following formula [14]:

$$P = \sum_{all\,nodes} \alpha \times D(y) \times C_y \times V^2 \times f \qquad (1)$$

Here, the capacitance C_y and the transition density $D(y)$ at each node y affect the dynamic power consumption with the swinging voltage V and clock frequency f. The α factor represents the static probability of a signal changing. α can be adjusted to reflect the specific use-case scenarios. When the transition density at a node is higher than one, then the node can switch its value more often than the clock signal. This is a result of *glitching*. The glitching of static CMOS circuits in ASIC is previously studied, concluding that it may contribute to 20% to 70% of power dissipation [15]. Furthermore, most of the FPGA dynamic power is consumed by the routing resources [16], [17] due to their large capacitive load.

The fact that the routing resources consume the most power and that there is a high possible power dissipation from glitches means that this can be exploited for power-hammering. The transition density is primarily affected by input states, the architecture of a circuit, and its exact physical implementation on the FPGA. This means that an implemented netlist could be designed to be malicious enough to create a temporary voltage drop that may manipulate a system (e.g., causing a state transition due to making a system timing-critical) [18] or to even crash the system entirely.

Although many power models take the switching activity into account [14], [16], [17], [19], [20], the problem is that these methods are not widely used or find the switching activity with simulation data. To find the transition density at a node without simulation, we need to find the probability of the signal changing. In this paper, we compute the probability using LUT truth table values. This computed probability propagates then further to the rest of the connecting nodes, as explained in Section IV.

III. Glitch Amplification Attacks

A. Principle/Theory

In a synchronous design, the output of a flip-flop can at maximum change its state once each clock cycle, hence, resulting in an activity factor of $\frac{1}{2}$ the clock frequency. For example, when the output of a flip-flop is fed back to the input through an inverter, then the loop will have an activity factor equal to half of the clock frequency. However, with different propagation delays combined with the evaluation of Boolean functions at combinatorial primitives on an FPGA (in this paper, we only consider LUTs, but the principle would also hold for other primitives such as DSP blocks), glitches may be generated as shown in Fig. 1. This fact is widely known, and a good designer usually reduces the glitching effect by either pipelining or avoiding flipping multiple inputs of a combinatorial logic block at the same time (e.g., using Gray codes for counters). Depending on the number of inputs N, the output activity factor can reach up to the maximum of $\frac{N}{2}$ times the input's activity factor. For example, in the implementation of a 6-input XOR gate in Fig. 1, the output of the T flip-flop has an activity of $\frac{1}{2}$, which is driven to all inputs of the XOR gate but routed with different latencies. This results in an activity factor of $6 \times \frac{1}{2} = 3$. This means that the XOR output toggles three times faster than the clock. Based on that principle, an attacker can create an acyclic circuit that meets timing constraints and appears unsuspicious, but that can generate a substantial switching activity and can draw an excessive amount of power. Using increased activity factor through glitch amplification is a way to perform power-hammering when the maximum clock frequency usable for an attack is somewhat limited (as commonly the case on FPGA Cloud instances [21]).

B. Attack Implementation

Glitch amplification attacks are comprised of two parts: the *glitch generator* and the *power-burning network* (See Fig. 1). More resources can be used on either part to make the other part less resource-intensive and more hidden.

The *glitch generator* is a standard T flip-flop with a delay chain and a wide-input XOR. In practical (Trojan) attacks, this may be controlled by some trigger logic. In our attack, we operate a T flip-flop at $200MHz$ frequency and connect its output to a delay chain followed by a 6-input XOR. By adjusting latencies of the physical implementation, this results in an activity factor of 3 at the output signal of the XOR gate. Consequently, the output of the XOR can reach 3 times the clock frequency (in our example, $600MHz$). Please note that much faster glitch frequencies can be generated by using well-tuned networks of XOR gates.

The *power-burning network* is built of wires running all over the FPGA fabric. The *glitch generator* is not the primary

power consumption source itself. Instead, the wires and components along the routing paths of the *power-burning network* are. This leads to the fact that by using a few logic primitives and redundant routing resources, a malicious circuit can be stealthily inserted without being obviously noticed.

IV. Mitigation Strategy

A. FPGADefender flow

For glitch-based power-hammering, bitstreams should be scanned before they get loaded into the FPGA. The scan could alternatively be performed at the netlist level, but a test operating on the final configuration bitstream has the advantage that it would even catch malicious circuits implanted during bitstream generation or after.

In our assumed scenario, FPGADefender would protect some partial regions in a reconfigurable FPGA-based system. An incoming configuration bitstream is at first translated by an external tool (BitMan [22]) into a netlist graph. This graph, together with the virus signatures, allows FPGADefender to perform the scan. The signatures can be plugged into the scanner engine with different configuration options to tune the scanner for different problems and FPGA boards.

B. Detecting glitch-based power-hammering circuits

We detect malicious circuits that draw excessive power through glitches by finding the transition density discussed in Section II. For each LUT table, we compute the probability that a single input change will cause the LUT output to switch. That probability will be used while traversing through the netlist to determine the transition density of individual wire segments of paths in the netlist graph.

We compute the LUT's output change probability with the following steps. First, we toggle all of the possible input values and then record the number of times the output changed. Second, we divide the result by the theoretical maximum number of output changes C that an n-input LUT can have:

$$C = n \times 2^n \tag{2}$$

That theoretical maximum is seen when each input signal's arrival also toggles the output. The formula multiplies the number of possible input encodings by n since the LUT input value can change x times when each of the input bits arrive one by one. If all of those input state transitions result in the output value changing, then we know that the LUT has a 100% probability that the output value will be toggled once one of the input values changes. Furthermore, we can deduce that the LUT is implementing either an XOR or an XNOR function.

We use Equation (2) to incorporate that physical LUTs can be underutilized (e.g., a LUT6 may implement a 4-input logic gate). We, therefore, use the PyEDA Espresso [23] library to minimize the LUT truth-table to find the number of used LUT inputs. The minimization also considers constant values. For instance, Vivado is using a LUT configured to 0 in case a logical '0' is needed. Therefore whenever a constant is connected to a LUT, the corresponding input will be skipped for the glitch score computation.

TABLE I
Glitch Amplification power-hammering attack on Ultra96 board clocked at 200MHz. The Vivado Power Estimator is set to default mode.

Designs Description	Activity Factor	Vivado Power Estimator (W)	Measured Board Power (W)
Static Output	0	1.963	4.026
Route Through	0.5	1.964	9.394
2-input XOR	1.0	1.965	10.858
3-input XOR	1.5	1.966	11.834
4-input XOR	2.0	1.967	12.200
5-input XOR	2.5	1.968	***crashed***
6-input XOR	3.0	1.969	***crashed***

With the LUT output change probabilities found, we traverse the netlist starting from each flip-flop until reaching another flip-flop or an antenna. During this process, if a LUT node is passed, the number of times the wire can change its output is multiplied by the chance that the output would flip on an input change. The transition density is computed by summing up the probabilities of the individual wires.

The initial transition density of a wire is set to 1. For catching malicious designs, we assume a worst-case scenario where each flip-flop toggles at each clock cycle. We also accumulate the transition density values up, assuming that glitches get amplified according to LUT functions without considering effects that may cancel out glitches.

In this paper, we are not weighting wires and LUTs in the sense that the elements of the physical implemented netlist have a different power consumption when being toggled.

V. Evaluation

A. Attack on Xilinx UltraScale+ FPGA

Our experiments are conducted on an Ultra96 board equipped with a Zynq UltraScale+ MPSoC ZU3EG. We filled the bottom row of the FPGA with 47 identical *glitch generators* (with little variations in the corners of the chip) using in total 0.03% of the flip-flops and 0.8% of the available LUTs. Glitching signals are connected to long routing paths, which are anchored using transparent latches. These latches make up the majority of flip-flops reported in Table II. We implemented the attack as shown in Figure 1 with about 200 ps latency increments between the LUT inputs. For the power burning network, we use *long deep paths* instead of high fan-out nets. This hinders the Xilinx vendor tool detecting the power-hammers through reported high fan-out nets.

In order to tune the glitching, we changed the generator's output behavior by setting the LUT6 configuration value. For instance, by making the outputs constant we deactivate glitching. Alternatively, the LUTs can just route one toggle signal through, or they can form XOR gates with a varying number of inputs (2 to 6) for adjusting hammering strength.

The results of different strengths are shown in Table I. It should be noted that the Xilinx Power Estimator numbers are reported for the FPGA fabric itself, whereas the measured board power is also including the power consumption of other off-chip components (e.g., regulators, memory chips, etc.). When we increase the activity factor, the power ramps up until it eventually crashes the design (when reaching XORs with 5

or 6-inputs). Our attack design is not optimized and stronger glitching should be well possible. Nonetheless, even just 0.8% of the LUTs and 25% routing resources are sufficient to crash the FPGA. Moreover, even the designs that do not crash the FPGA can still cause malicious behavior as the power supply may have no headroom for operating other modules.

B. Evaluation of FPGADefender

We evaluated FPGADefender by scanning 19 designs from different application domains and two malicious designs. All of the benchmark designs except *FPGA Miner* [24] are constrained to a region which takes up to a sixth of the whole fabric resources. We used FPGADefender with the new virus signature to compute the overall switching activity of the design (given as a bitstream encoded netlist).

From Table II, we can see that the benchmark designs have lower overall switching activity than the malicious design. The SHA3 accelerator stands out because it uses extensively wide XOR functions (which results in huge activity factors) and bit-shuffling operations (which results in high wire utilization). Moreover, this particular SHA3 implementation is unrolled and not well-pipelined. Due to our approach considering a worst case scenario for computing the activity sum, the wide XOR functions indicate a level of switching that is larger that what is possible when running the circuit on an FPGA.

The second last entry in Table II is the same design as the glitching design, but with the LUT functions changed from XOR to routing though the first input, as described in the previous section. Thus in the *pass_through_mal* design, the non-glitch amplified signal propagation causes the scanner to evaluate its switching activity count to be four times less as expected from the design using glitch generation (XOR6).

A vital factor to emphasize with these results is that the scanner model assumes that the inputs change every rising clock edge and that the glitches always propagate through the entire combinatorial path without canceling out effects. These assumptions are not accurate in real scenarios as not all flip-flop values change every clock cycle, and some glitches cancel out if the pulse-width of the glitch is not large enough.

As a result, the switching activity values for wire segments have been capped to 20 as we anticipate that wire segments on the ZU3EG will not switch more often than a rate equivalent to $4GHz$. Thus we can spot the malicious design, which has the highest activity sum out of all of the other designs. The exact threshold mark when a design has to be considered malicious still has to be explored, and the model needs to get improved to handle routing delays such that glitch canceling-out is taken into consideration.

TABLE II
EVALUATION RESULTS FOR THE MALICIOUS DESIGN AND OTHER BENCHMARKING CIRCUITS.

Name	Activity Sum	LUT %	FF %	Wires %
FPGA Miner [24]	693,023.75	5.12	3.59	2.02
RISC-V CPU [25]	199,960.90	5.55	1.32	0.94
MIPS CPU [26]	301,952.44	6.44	1.00	0.95
I2C [27]	13,208.40	0.51	0.14	0.07
SPI [27]	69,865.90	1.63	0.23	0.21
PRNG [27]	11,490.73	0.40	0.07	0.05
BCD Adder [27]	5,431.179	0.11	0.06	0.02
Cordic [27]	120,376.75	2.14	0.67	0.28
8b10b EncDec [27]	4,348.80	0.12	0.03	0.02
RS232 UART [27]	5,107.70	0.17	0.07	0.02
Stepper Motor [27]	2,724.50	0.12	0.04	0.01
Parallel Scrambler [27]	12,432.11	0.11	0.03	0.01
CAN Controller [27]	64,168.29	2.14	0.45	0.31
AES [28]	223,778.39	6.95	0.40	0.68
DES [28]	27,046.00	0.46	0.09	0.04
TRNG [28]	71,625.96	1.76	0.11	0.16
SHA3 [29]	5,421,767.16	15.13	1.62	4.94
pass_through_mal	1,888,902.00	0.80	44.57	25.46
Malicious design (XOR6)	7,518,387.00	0.80	44.57	25.46

VI. CONCLUSION

With this paper, we contribute to a recent trend of research being undertaken in the field of FPGA hardware security. We demonstrated that glitch amplification, which has not been studied for malicious circuit designs before, can be used to draw excessive levels of power. Our experiments have proven that an Ultra96 FPGA board can be crashed by using less than one percent of the available LUTs and a quarter of the wires. In that case, we could create an increase of dynamic power consumption of about 10W (measured on the 12V board supply rail). With this, we can estimate how much more power could be drawn with an Alveo U250 datacenter card, which provides $22\times$ more capacity (in terms of LUTs) and that has a total thermal power budget of 225W (which also powers four DDR memory modules). Considering that the VU11P of the Alveo card is produced using the same processing technology as the ZU3EG of the Ultra96 is and that both FPGAs have an identical fabric architecture, the here presented power-hammering circuit could very likely crash that board. Furthermore, on Amazon AWS F1 instances (featuring a Xilinx VU9P FPGA), power is limited to below 100W, and our power-hammering circuit passes all mandatory checks to be deployed. Therefore this attack is mountable on AWS F1 instances with the corresponding impact.

This work completes other work that is focusing on self-oscillating designs only. Furthermore, with the new glitch amplification detection functions added to FPGADefender (available at: [13]), the tool is now providing a more complete solution for mitigating power-hammering (and side-channel) attacks. The tool thus would enable a cloud service provider to offer FPGA-as-a-Service models where users can even upload bitstreams (rather than netlists only, as standard practice today). For future work, the effect of tuning different parameters of this attack will be studied to make the scans more accurate.

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

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

[2] Microsoft, "What are field-programmable gate arrays (FPGA) and how to deploy," https://docs.microsoft.com/en-gb/azure/machine-learning/how-to-deploy-fpga-web-service/.

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

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

[5] A. Vaishnav, K. Pham, K. Manev, and D. Koch, "The FOS (FPGA Operating System) Demo," in *FPL*, 2019.

[6] J. Weerasinghe, F. Abel, C. Hagleitner, and A. Herkersdorf, "Enabling FPGAs in Hyperscale Data Centers," in *UIC-ATC-ScalCom*, 2015.

[7] S. S. Mirzargar and M. Stojilovic, "Physical Side-Channel Attacks and Covert Communication on FPGAs: A Survey," in *FPL*, 2019.

[8] D. Gnad, F. Oboril, and M. B. Tahoori, "Voltage Drop-based Fault Attacks on FPGAs using Valid Bitstreams," in *FPL*, 2017.

[9] I. Giechaskiel, K. Rasmussen, and J. Szefer, "Measuring Long Wire Leakage with Ring Oscillators in Cloud FPGAs," in *FPL*, 2019.

[10] K. Matas, T. La, N. Grunchevski, K. Pham, and D. Koch, "Invited Tutorial: FPGA Hardware Security for Datacenters and Beyond," in *FPGA*, 2020.

[11] C. Beckhoff, D. Koch, and J. Torresen, "GoAhead: A Partial Reconfiguration Framework," in *FCCM*, 2012.

[12] C. Lavin and A. Kaviani, "RapidWright: Enabling Custom Crafted Implementations for FPGAs," in *FCCM*, 2018.

[13] K. Matas and T. La. (2019) FPGADefender. [Online]. Available: https://github.com/KasparMatas/FPGAVirusScanner.git

[14] K. K. Poon, A. Yan, and S. J. Wilton, "A Flexible Power Model for FPGAs," in *FPL*, 2002.

[15] A. Shen, A. Ghosh, S. Devadas, and K. Keutzer, "On Average Power Dissipation and Random Pattern Testability of CMOS Combinational Logic Networks," in *ICCAD*, 1992.

[16] V. Degalahal and T. Tuan, "Methodology for High Level Estimation of FPGA Power Consumption," in *ASP-DAC*, 2005.

[17] L. Shang, A. S. Kaviani, and K. Bathala, "Dynamic Power Consumption in Virtex-II FPGA Family," in *FPGA*, 2002.

[18] J. Krautter, D. Gnad, and M. Tahoori, "FPGAhammer: Remote Voltage Fault Attacks on Shared FPGAs, suitable for DFA on AES," *IACR TCHES*, vol. 3, pp. 44–68, 2018.

[19] F. Li, D. Chen, L. He, and J. Cong, "Architecture Evaluation for Power-Efficient FPGAs," in *FPGA*, 2003.

[20] E. Todorovich, M. Gilabert, G. Sutter, S. Lopez-Buedo, and E. Boemo, "A Tool for Activity Estimation in FPGAs," in *FPL*, 2002.

[21] Amazon Web Services. (2018) Viewing and Configuring Clock Frequencies. [Online]. Available: https://github.com/aws/aws-fpga/blob/e6164cb945fe616485c044f00ce6cab47abfdedd/hdk/docs/dynamic_clock_config.md

[22] K. Pham, E. Horta, and D. Koch, "BITMAN: A Tool and API for FPGA Bitstream Manipulations," in *DATE*, 2017.

[23] C. Drake. (2019) PyEDA. [Online]. Available: https://pyeda.readthedocs.io/en/latest/index.html

[24] progranism. (2011) Open-Source FPGA Bitcoin Miner. [Online]. Available: https://github.com/progranism/Open-Source-FPGA-Bitcoin-Miner.git

[25] C. Wolf. (2019) PicoRV32. [Online]. Available: {https://github.com/cliffordwolf/picorv32}

[26] D. Koch, C. Beckhoff, and G. G. F. Lemieux, "An Efficient FPGA Overlay for Portable Custom Instruction Set Extensions," in *FPL*, 2013.

[27] OpenCores. (2020) Free and Open Source gateware IP cores. [Online]. Available: {https://opencores.org/}

[28] A. Vaishnav, J. R. G. Ordaz, and D. Koch, "A Security Library for FPGA Interlays," in *FPL*, 2017.

[29] K. D. Pham, E. L. Horta, D. Koch, A. Vaishnav, and T. Kuhn, "IPRDF: An Isolated Partial Reconfiguration Design Flow for Xilinx FPGAs," in *MCSoC*, 2018.