Powerhammering through Glitch Amplification – Attacks and Mitigation

DOI:
10.1109/FCCM48280.2020.00018

Document Version
Accepted author manuscript

Published in:
Proceedings - 28th IEEE International Symposium on Field-Programmable Custom Computing Machines, FCCM 2020

Powerhammering through Glitch Amplification – Attacks and Mitigation

***Abstract*—Recent work on FPGA hardware security showed a huge potential risk through powerhammering which uses high switching activity in order to create excessive dynamic power loads. Virtually all present powerhammering 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 together with driving high fanout 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 FPGA cloud computing can be observed in recent years when major cloud vendors such as Amazon [1], Microsoft, Huawei, and Alibaba offer FPGA resources in their infrastructures. Meanwhile, multi-tenancy support for FPGA infrastructure in which more than one user can share the same FPGA resource is desirable and hence attracting active research works in both academic [2] and industry [3].

In such multi-tenant scenarios for FPGA cloud computing, system security is of paramount importance. Security concerns range from data privacy of users sharing the same FPGA fabric or FPGAs on the same board to the availability of the system service itself [4]. Apparently, integrating FPGA resources to a cloud computing infrastructure is opening a new surface of attack in 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 implemented in FPGA resources can be used to bring down an entire FPGA board which needs to be power cycled to get back the normal operation [5]. Such an attack, known as powerhammering is a specific threat for FPGAs due to their full low level hardware programmability.

Currently, all powerhammering circuits are built up on ring oscillators which may or may not be detected by the vendor design rule checks (DRC) [6]. 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. Alternatively, and the core of this paper, it is possible to use well-designed XOR trees to generate a great number of switching activities (a.k.a *glitches*) per clock cycle. With the help from academic tools such as GoAhead [7] or RapidWright [8], we can fine-tune input delays inside XOR trees to achieve the desired toggling frequency. While other logic functions may glitch as well, XOR is most effective as any change at an input creates a change at the XOR

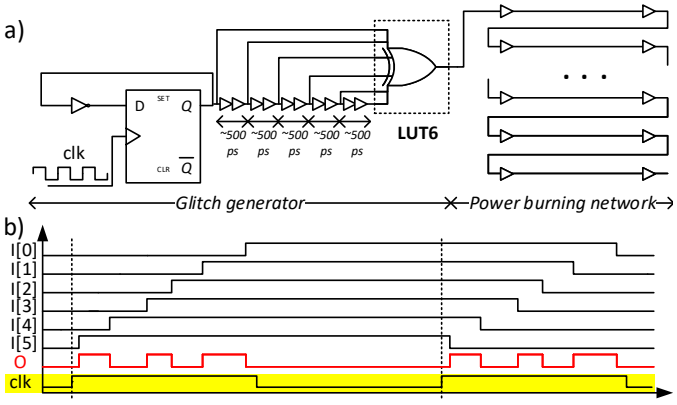

Fig. 1. Illustration of the presented glitch amplification attack. a) is the attack circuit; and b) is the waveform getting from the attack circuit.

output (omitting possible canceling effects in real systems). In our research, the output of a *glitch generator* will be used to drive a large network of wires and combinatorial logic which is acting as the *power burning network*, as illustrated in Figure 1. As we will show in Section IV, this allows to create a powerhammering attack at very low cost that can even crash an FPGA board.

In this paper, for the first time, such malicious circuits which are able to 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 v2019.1 for bitstream generation as well as all tests that are performed to deploy designs on Amazon Web Services F1 instances. It is important 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 the open-source tool FPGADefender [9] by providing additional virus signatures for this class of attack. After adding the here proposed virus signature, FPGADefender is now able to detect malicious circuit containing glitch amplification that may potentially draw excessive power. The test is performed directly on bitstreams generated by the Xilinx vendor tool.

The threat model considers an adversary with complete or 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 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 powerhammering attack based on glitch amplification that does not require ring 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 latest 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. The type which consumes most power in modern devices is dynamic power. Dynamic power can be calculated using the following formula [10]:

$$P = \sum_{allnodes} 1/2 C_y V^2 D(y) f \tag{1}$$

We can see that 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. 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*. Glitching of static CMOS circuits in ASIC is previously studied, concluding that it may contribute to 20% to 70% of power dissipation [11]. Furthermore, most of the FPGA dynamic power is consumed by the routing resources [12], [13] due to their large capacitive load.

The fact that most power is consumed by the routing resources and that there is a high possible power dissipation from glitches means that this can be exploited for powerhammering. The transition density is largely affected by the main input, the architecture of the design, 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) [14] or to even crash the system entirely.

Although there are many power models that take the switching activity into account [10], [12], [13], [15], [16], 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 that the signal changes and that will in this paper be computed using LUT truth table entries. This computed probability propagates then further to the rest of the connecting nodes.

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 $1/2$ the clock frequency. This is the situation when a Flip-Flop implements a T Flip-Flop (where the output of a Flip-Flop is fed back to the input through an inverter). However, due to different propagation delays on routing and for evaluating 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 Figure 1. This fact is widely known and a good designer usually reduces the glitching effect by either pipelining or avoid 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 Figure 1, the output of the T Flip-Flop has an activity of $1/2$ which is driven to all inputs of the XOR gate but routed with different latencies. This results in an activity factor of $6 \times 1/2 = 3$. 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 completely normal but it can generate a substantial switching activity and can draw an excessive amount of power. The increasing of the activity factor through glitch amplification is also a way to perform powerhammering when the maximum clock frequency usable for an attack is rather limited (as commonly the case on FPGA Cloud instances [17]).

B. Attack Implementation

Glitch amplification attacks are comprised of two parts: the *glitch generator* and the *power burning network* (See Figure 1).

The *glitch generator* is a normal T Flip-Flop with a delay chain and a wide-input XOR. In practical attacks, this may be controlled by some trigger logic. In our attack, we operate the T Flip-Flop at $200MHz$ frequency then 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 networks of XOR gates and a well-tuned implementation.

The *power burning network* is built of wires running all over the FPGA fabric. The main power consumption is not drawn by the *glitch generator* itself but by the wires and components along the routing paths of the *power burning network*. 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

To mitigate the presented attacks, 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. For this checking purpose, FPGADefender [9] was used for this paper.

FPGADefender would protect some partial region in a reconfigurable FPGA-based system. An incoming configuration bitstream is at first translated by an external tool (BitMan [18]) 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 powerhammering

To detect malicious circuits that draw excessive power by propagating generated glitchy signals, the transition density discussed in Section II will have to be computed from the generated netlist graph. To find that density, the scanner uses the LUT configuration values in the netlist. These values help finding the probability that a single input change will cause on a LUT's output. That probability will be used while traversing through the netlist to determine the transition density.

To find a LUT's output change probability, all of the possible input values will have all bits toggled and the number of times the output changed is recorded. Then that value is divided by the maximum number of output changes which is when each toggle also flips the output.

Physical LUTs can have more inputs than the design uses (e.g., a LUT6 may implement a 4-input logic gate). To get the number of used inputs, the LUT configuration values will be minimized using the PyEDA Espresso [19] library. We are also considering constant values. For instance, each CLB has a VCC primitive in the case a logical '1' is needed and for a logical '0', Vivado is using a LUT configured to '0'. Whenever a constant is connected to a LUT, the corresponding input will be skipped for the glitch score computation.

Next, the wires are processed from all of the nodes annotated as Flip-Flops until another Flip-Flop or an antenna is reached. During the process, if a LUT node is detected, the number of times the wire can change its output is multiplied by the chance that the output would flip on an input change. For catching malicious designs, we assume the worst-case scenario where the starting Flip-Flop values toggle every clock cycle and the number of glitches gets amplified with each LUT encountered without effects that may cancel out glitches.

TABLE I
Glitch Amplification powerhammering 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***

V. Evaluation

A. Attack on Xilinx UltraScale+ FPGA

Our experiments are conducted on a Ultra96 platform equipped with a Zynq UltraScale+ MPSoC ZU3EG. We implemented 47 *glitch generators* using only 0.03% Flip-Flops and 0.8% of the LUT resources. Glitching signals are then connected to long routing paths which are anchored using transparent latches. Here we guided the routing behavior by routing through transparent latches and pre-placed them all over the design. It should be noticed that in this attack, we do not use high fan-out but we use *long deep paths* instead. Therefore, Xilinx vendor tool[1] cannot detect the powerhammering circuits through a reported high fanout net. Moreover, we can easily change output's behavior by setting LUT6 initial value to realize static output (output always equals to 0), route through (output always equals to one input), and XOR gates with inputs varying from 2 inputs to 6 inputs. By keeping the same routing and changing LUT value, we can observe the effect of glitching on the power consumption.

The results are shown in Table I. It should be remarked that the Power Estimator is 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 using 5-input XORs and 6-input XORs). Ultimately, our attack design is not optimized, and even in this case, just 0.8% of the LUTs and 25% routing resources are sufficient to crash the FPGA. Moreover, even the other designs that do not crash the FPGA can still cause malicious behavior as the FPGA core voltage may drop and the system has no headroom for regulating user circuit demands.

B. Evaluation of FPGADefender

To evaluate FPGADefender, 18 designs were scanned with an additional malicious design. These designs are common implementations of different accelerators. All of the benchmark designs except *FPGA Miner* [20] are constrained to a partially reconfigurable region which takes up to a sixth of the whole fabric resources. We used FPGADefender with the new virus signature to measure the overall glitchiness of the design.

[1] Vivado 2019.1 was used in this experiment.

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

Name	Activity Sum	LUT %	FF %	Wires %
picoRV	166723.47	5.55	1.32	0.94
i2c	11059.16	0.51	0.14	0.07
SPI	60090.64	1.63	0.23	0.21
PRNG	9778.64	0.40	0.07	0.05
AES	189270.98	6.95	0.40	0.68
DES	24902.57	0.46	0.09	0.04
TRNG	60405.56	1.76	0.11	0.16
BCD_adder	4524.12	0.11	0.06	0.02
cordic	102021.77	2.14	0.67	0.28
8b10b_encdec	3469.10	0.12	0.03	0.02
FPGA Miner	564668.20	5.12	3.59	2.02
RS232 UART	4195.33	0.17	0.07	0.02
steppermotor	2353.30	0.12	0.04	0.01
parallel_scrambler	11772.32	0.11	0.03	0.01
CAN_controller	54043.85	2.14	0.45	0.31
SHA3	4101351.79	15.13	1.62	4.94
MIPS CPU	235668.21	6.44	1.00	0.95
pass_through_mal	1725320.00	0.80	44.57	25.46
Malicious design	10351920.00	0.80	44.57	25.46

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 XOR functions and bit-shuffling operations which result in high wire utilization. Moreover, this particular SHA3 implementation is unrolled and not well pipelined.

The second last entry in Table II shows, one of the non-malicious designs that is an exact copy of the attacking design with the only difference that the LUT is simply routing through without further glitch amplification. For the *pass_through_mal* design the non-glitchy signal propagation causes the scanner to evaluate its switching activity count to be 6 times less. This demonstrates the effect of glitch amplification in a malicious design.

Additionally, the scanner model assumes that the inputs always change and that the glitches always propagate through the entire combinatorial path and that all routing involved in the glitching. These assumptions are not true 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 wire segment switching activity values have been capped to 20 as wire segments on the ZU3EG won't switch more often than a rate equivalent to 4GHz. Additionally, wire segments in the CARRY primitives inside CLBs have been discarded as well with the multiplexers inside the logic slices (e.g., F7MUXes) which will be covered in future work.

VI. CONCLUSION

With this paper, we contribute to an increased amount of research being undertaken in the field of FPGA hardware security. We demonstrated that glitch amplification, which has not be 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 only 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). To put this into perspective, an Alveo U250 datacenter card provides 22x more LUTs and has a total thermal power budget of 225W (which also powers 4 large DDR memories). Considering that the VU11P of the Alveo card is produced in the same process node than the ZU3EG of the Ultra96 and that both FPGAs have an identical fabric architecture, the here presented powerhammering circuit could potentially already crash that board. Even worse, on Amazon AWS F1 instances, power is limited to below 100W and our powerhammering circuit passes all mandatory checks to be deployed.

This work completes other work that is focusing on self-oscillating designs only. And with the new glitch amplification virus scanner rules added to FPGADefender, the tool is now providing a mostly complete solution for mitigating power hammering attacks. This would enable a cloud service provider to offer FPGA-as-Service models where users can even upload bitstreams (rather than netlists, as common practice today).

REFERENCES

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

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

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

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

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

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

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

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

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

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

[11] 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.

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

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

[14] 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.

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

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

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

[18] K. Pham, E. Horta, and D. Koch, “BITMAN: A Tool and API for FPGA Bitstream Manipulations,” in *DATE*, 2017.
[19] C. Drake. (2019) PyEDA. [Online]. Available: https://pyeda.readthedocs.io/en/latest/index.html
[20] progranism. (2011) Open-Source FPGA Bitcoin Miner. [Online]. Available: https://github.com/progranism/Open-Source-FPGA-Bitcoin-Miner.git