Checking for Electrical Level Security Threats in Bitstreams for Multi-Tenant FPGAs

Dennis R.E. Gnad*, Sascha Rapp†, Jonas Krautter*, Mehdi B. Tahoori*
Institute of Computer Engineering, Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany
*{dennis.gnad, jonas.krautter, mehdi.tahoori}@kit.edu †sascha.rapp@student.kit.edu

***Abstract*—Multi-tenant FPGAs, in which 3rd parties have partial access to the FPGA fabric, are a rising usage trend in cloud and reconfigurable SoCs. This gives rise to new types of attacks in FPGAs, as shown in recent studies. These attacks can operate on the electrical level through the common power delivery network, making them very hard to isolate. Thus, software-controlled FPGA configuration can be exploited to insert hardware trojans, impacting the security of the entire system. The attacks can be separated into fault and side-channel attacks to either actively manipulate a system or quietly extract secret information. In this paper, we show the first attempt of countermeasures against these voltage fluctuation based attacks, by analyzing FPGA bitstreams for malicious logic, basically implementing an *FPGA antivirus*. We provide a way to check bitstreams for potentially malicious structures, by extending a combination of commercial and open-source tools.**

I. Introduction

FPGAs are increasingly incorporated into many systems from low-power IoT-devices such as smartphones, all the way to high-performance cloud servers [1–5]. Altogether, FPGAs are increasingly split among multiple users or 3rd parties, and become remotely accessible, which opens up new security threats through partial configuration bitstreams [6–11].

At the logic level, proper isolations and access restrictions must be considered across different users of the same FPGA fabric [12]. On the electrical level, fault attacks [13] and power analysis side-channel attacks (SCA) [14] can still be used to attack cryptographic implementations inside integrated circuits, to recover secret information from the chip, even in the presence of proper logic-level isolation. These types of attacks traditionally carried out through physical access with measurement and fault injection equipment, are shown to be performed remotely through software-based access [6–11].

As multi-tenant FPGAs are a foreseen operation mode for FPGAs [2, 3, 15], the elimination of associated security threats is a must for their successful adoption. On one hand, it might be possible to mitigate these threats by architectural changes in the FPGA, such as separating the FPGA fabric into blocks of individually powered voltage islands. On the other hand, a hypervisor can be made responsible for only allowing benign bitstreams to be loaded into the FPGA. This mandates checking the bitstreams before loading, similar to antivirus software checking binary executables. Existing work already mentioned partial configuration bitstreams might require checks as a safety measure to prevent dangerous short-circuits [16]. The works that showed the remote electrical attacks also mentioned to check bitstreams for malicious circuits as a potential countermeasure [6, 8]. Since architectural changes are expensive and require a new chip generation, we believe it is crucial to follow the path of bitstream checking, but without overly restricting benign designs.

In this paper, we provide a methodology to check FPGA bitstreams for patterns of malicious structures based on electrical level attacks. By extracting a technology-mapped netlist graph from existing bitstreams, we check for potential security threats, the so-called *signatures*, containing fundamental requirements for voltage-based attacks. Finally we will evaluate and discuss to which extent these design restrictions affect benign designs, and what are the possible next steps to decrease limitations without sacrificing security. For the implementation, a mix of open-source and commercial tools is used, with open-source contributions to *icestorm* [17] and *yosys* [18][1].

Outline: Section II summarizes related work. Section III explains our bitstream checking approach. Section IV shows results and evaluation. Section V concludes the paper.

II. Background and Related Work

A. Power Supplies and their Security Implications

Modern semiconductor chips are supplied by a power distribution network (PDN). In these PDNs, there is a hierarchy of active and passive electronic components that start at the board level, reaching down to individual transistors to regulate the supply voltage delivered to the individual components. Since the network is not ideal, voltage noise persists on all PDN levels.

PDNs are usually hierarchical, integrating multiple voltage regulators for all the required voltage levels in the system. They also require several resistor, capacitor and inductor (RCL) components for proper functionality, while some exist as parasitic components. The total network can then be modeled as a complex network of RCL components [19].

Voltage drop can then be simplified into $V_{drop,R} = IR$ and $V_{drop,L} = LdI/dt$ components. In modern systems, $V_{drop,L}$ is the more prominent part, originating from high changes in current I in small times t, due to simultaneous fast switching of multiple gates. This, in effect, will also affect the gate and path delays $t_{delay} \propto \frac{V_{supply}}{V_{supply} - LdI/dt - IR}$.

[1] Available: https://cdnc.itec.kit.edu/MaliciousBitstreams.php

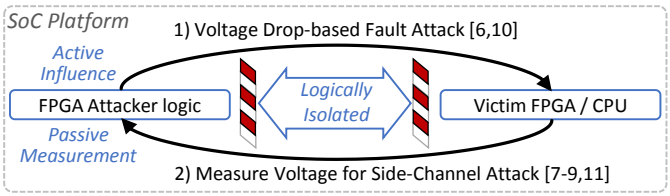

Figure 1: Simplified summary of the two threats in a system with shared FPGA logic, as an overview over the previous works [6–11]

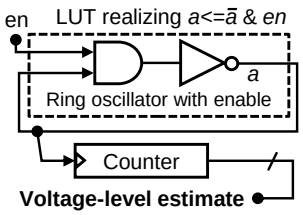

Figure 2: Ring oscillator to sense voltage fluctuations (cf. [8, 20])

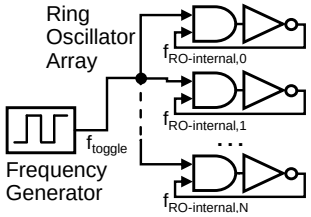

Figure 3: Ring oscillator array to cause voltage drop-based faults (cf. [6])

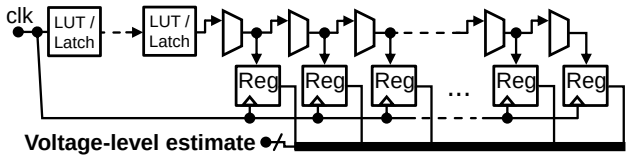

Figure 4: LUT/Latch based delay line with carry-chain based Time-to-Digital Converter to measure voltage fluctuations in FPGAs (cf. [7, 21])

Voltage fluctuations have security implications. On one hand, significant voltage drop can lead to timing violations and faults. These faults are used in *fault attacks* to recover secret keys from cryptographic circuits [13], or crash a system for denial-of-service (DoS) [6]. On the other hand, monitoring voltage fluctuations from normal circuit behavior is a well-studied *side-channel* that can leak secret information [14].

B. Threats in Systems Containing FPGA Logic

It has been shown that both power analysis SCA [7–9, 11] and fault attacks [6, 10] are possible from inside an FPGA chip, without dedicated measurement and fault injection equipment. Through diverting FPGA primitives from their intended use, standard FPGA software can be used to implement designs that cause or measure voltage fluctuations, bypassing any isolation features on the logical level, as summarized in Fig. 1. Thus, attacks are possible from inside the chip that require no logical connection between attacker and victim, and can not be covered by existing mitigation procedures.

Fault Attacks that originate from inside the FPGA are based on causing a high change in current in a small time (dI/dt), used for Differential Fault Analysis [10], or DoS attacks [6]. In these attacks, ring oscillators (ROs) are implemented in FPGA LUTs as shown in the top of Fig. 2. Enabling multiple ROs simultaneously causes a high current in a short time, and thus significant voltage drop. To cause a sufficient voltage drop that leads to timing violations or a crash, the ROs are toggled between enabled and disabled using an additional f_{toggle} frequency, with a structure as shown in Fig. 3. By that, the weak parts of a PDN are targeted for a successful attack.

Power analysis SCAs use voltage or current fluctuations [14]. Because t_{delay} depends on voltage, observing path delays can be used as a sensor. By counting how often ROs toggle in a fixed timeframe, relative voltage changes can be estimated [20], as shown in Fig. 2. A faster sensor is a Time-to-Digital Converter (TDC), shown in Fig. 4. It uses multiple cascaded FPGA primitives with registers added between them. Instead of counting, these registers can be read every clock cycle to get a relative voltage estimation [21].

III. Checking for Malicious Logic

In this work, we establish a methodology to check for malicious structures in bitstreams. Similar to an *Antivirus*, we look at *signatures* of malicious logic structures that might lead to electrical-level attacks, by defining properties that these structures fulfill. We check for both structural and behavioral properties that are uncommon in benign bitstreams, indicating malicious intentions. Please note that we do not simply check for exact matches of known attacks. Instead, we formulate the fundamental properties required for the attacks.

In Fig. 5 we present our overall flow for checking bitstreams. To be able to check existing FPGA bitstreams for malicious logic, it has to be made readable by common tools first. The bitstream is first unpacked into a vendor-specific ASCII representation, which in turn is converted into a flattened Verilog netlist using another tool. In our approach, a full representation of the bitstream is generated, that is again fully functional after re-synthesis and mapping.

After acquiring the technology mapped netlist, we split up the detection of malicious logic into the identification of combinational and sequential logic parts, across which both signatures for fault and SCA are spread. Combinational cycles can be both part of sensors, as well as a mechanism to introduce faults. This makes a structural analysis of the netlist graph necessary: Combinational cycles need to be extracted in order to evaluate their oscillation behaviour and determine, whether they can be used to cause faults. On the other hand, we use timing analysis in sequential logic, to prevent timing violations on paths that could reveal information about voltage fluctuations. Even a single correlated bit could reveal enough information when monitored over a certain timeframe.

In the following subsections we will give a more deeper understanding about the basic primitives and requirements for fault or side-channel attacks and their respective properties, which we will have to find in either combinational or sequential logic of netlist graphs to detect malicious logic.

A. Signature for Fault Attacks

As explained in Section II-B, an attacker can use ROs to cause faults or crash an FPGA. The crash or faults are achieved by additionally controlling the RO oscillation with an *enable* signal as Fig. 3 shows.

To identify a potentially malicious design, we evaluate specific characteristics in combinational and sequential logic respectively, summarized in Table I, under **Fault Attack**:

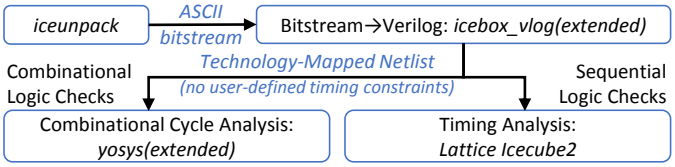

Figure 5: Overview of the methodology and implemented flow to check bitstreams for threats in combinational or sequential logic.

- From the generated netlist graph we extract combinational cycles. An experimentally evaluated threshold defines the maximum allowable (benign) number of cycles, which are allowed to be deployed on the fabric without being able to provoke crashes and fault attacks.
- Additionally, we check for a common node within the logical input cone of the combinational cycles, which would allow a synchronized activation/toggling of an RO grid.

B. Signatures for Power Side-Channel Attacks

In Section II-B we explained how an attacker can re-utilize FPGA primitives in a way to measure voltage fluctuations inside the chip. For this type of threat, two types of sensors have been shown, which can be formulated as two signatures that can be analyzed during bitstream checking.

In one case, ROs can be re-utilized as sensors, summarized in Table I, under **SCA #1**:

- Because counters based on ROs allow to sample voltage traces sufficiently for power analysis attacks [8], we must again find combinational cycles, similar to fault attacks. In this case, ROs must not have an observable output.
- These combinational cycles additionally require an output to make them suitable as a sensor. In this case, opposed to ROs for fault attacks, even a single RO can be sufficient to measure voltage, and is thus potentially malicious.

In a second case (Table I, under **SCA #2**), TDCs or any timing violations can be used as sensors. These can be checked through timing analysis:

- To prevent the use of delay lines and thus TDC sensors, we check the design for all possible timing violations. Even a single bit with unstable output, which depends on the supply voltage, can be enough to recover a secret encryption key.

Please note, since the design is extracted from a bitstream, the extracted netlist has zero timing constraints. This will strip the design from all user constraints that could otherwise hide TDC sensors. However, it will also remove any legitimate constraints, such as false path constraints for clock-domain crossing synchronizers or similar circuits. The timing information must be derived again from the basic board configuration and external clock generator or crystal, as well as internal clock generators, and also from the placement of the logic elements from the bitstream.

Successful sensor detection depends on the quality of the timing analysis, which has to include all *possible* clock configurations set at runtime. More complex designs might, for instance, use dynamic clock control or clock-domain crossings. These are general tasks for timing analysis and verification tools, and must report dynamic clock setups in which timing violations can occur. For security, only safe clock setups must be enforced, which is out of scope in this work.

Table I: Circuit properties of malicious logic signatures

Property	Fault Attack	SCA #1	SCA #2
Has combinational cycles?	×	×	
Has timing violations?			×
Comb. cycle with input?	×		
Comb. cycle with output?		×	

IV. Implementation and Results

Lattice iCE40 FPGAs are used for our evaluation, since their bitstreams are already reverse engineered and an incomplete tool existed that can create verilog from some elements in a bitstream [17], which we extended. Please note that the basic concept of our work will still hold true for other vendors. Since the existing attacks were not shown on Lattice FPGAs yet, we reproduce some of them.

A. Reproduction of Attacks in Lattice FPGAs

We reproduce a fault attack with ROs and side-channel attack with TDCs on a Lattice iCE40-HX8K Breakout Board, combined in a single design utilizing 62% FPGA resources.

Fault attack logic was implemented with LUT-based ROs, following the concept in Fig. 3 like previous attacks. The board crashes on a frequency sweep on f_{toggle} with 1920 ROs, using about 25% of available FPGA logic. For instance, $f_{toggle} = 300\,\text{kHz}$ is suitable. The FPGA resets and loses its configuration bitstream, but it can then be reprogrammed again from the host PC.

For SCA, we reproduce the TDC-based (cf. Fig. 4) attack, since RO-based sensors are not much different to ROs for faults. We use a 24 MHz AES-128 module with a 32bit datapath. In the other half of the FPGA, we put TDC-based voltage sensors at 96 MHz, and use a synchronization helper signal. We use the same leakage model as in [7], and achieve a sucessful key recovery after about 40000 traces.

B. Toolchain Extensions

To implement our flow from Section III, we base the bitstream reversal on a tool from the *icestorm* tools [17], and extend it. The unmodified *icebox_vlog* tool allows reverse engineering LUT components from Lattice iCE40 FPGA bitstreams into verilog. However, only logic assignments are used, carry chain primitives are unsupported, and placement constraints are missing. Thus, we extended icebox_vlog to include placement constraints and carry elements, as well as LUT instantiations of the respective iCE40 primitives, matching the original bitstream. A limitation is the lack of routing constraints, which do not exist in either commercial or open-source software. Please note that for a completely secure bitstream checker, routing would still need to be included.

To analyze combinational cycles, we extend the *yosys* open-source synthesis/analysis tools [18]. Yosys allows to search for combinational cycles by finding *strongly connected components* (SCCs) in the netlist graph, while ignoring registers during the search. By not revisiting nodes that it previously

processed, it will show cycles at arbitrary starting points per default, but not the loopback signal, which we implemented in our extension. At the same time, we use yosys to find a potential outgoing or incoming signal into the SCC, to evaluate the required properties shown in Table I.

Although the described tools are specific for Lattice iCE40 FPGAs, the general approach of reversing bitstreams into HDL code and checking the design for threat signatures can be deployed by all FPGA manufacturers in a similar and more extensive way, when detailed vendor knowledge is available.

C. Evaluation on Reproduced Attack Bitstreams

Fault Attack: Fig. 6 shows an extract of a graphviz diagram produced by our modified *yosys* and *icebox_vlog* software. It shows a successful isolation of SCC subgraphs for each RO, and includes their common input register instance (n53_inst), which can enable or disable the ROs through the common enable wire n53. The *BUF* blocks are interconnect points.

SCA: To identify timing-violation based TDC sensors, we use the timing analysis tool within the Lattice iCEcube2 software. We first implemented the original design of an attacker circuit, and then reversed the resulting bitstream into a verilog file using our modified icestorm tools. We compared the critical paths of both original and reversed implementations. Both lead to the last bit endpoint of the 64bit delay line TDC as the most critical path. However, as expected, the reported slack value differs slightly due to the lack of routing constraints in the iCEcube2 toolchain. The result can be verified in the floorplan. Figure 7 shows the critical path end node in the original floorplan, which is identical with *n19* in the reversed floorplan in Figure 8.

Overhead: With the reproduced design and both attacks, executing all checks requires about 2 minutes on an Intel Xeon E5-1630v4 system.

D. Discussion

We have shown a first approach to detect the known voltage-based attacks at the electrical level in which FPGA logic is re-used maliciously. However, there still exist some limitations. When checking for fault attacks, only combinational cycles were examined. However in principle high-power sequential logic could also cause faults. For SCA, the restrictions are too high and can impact legitimate designs in the form of false-positives, for instance with clock-domain crossings. These issues will be addressed in our future work.

V. Conclusion

FPGAs offer a high degree of design freedom to allow implementation of arbitrary designs in hardware, which can be exploited by any attacker with access to FPGA bitstreams. This freedom needs to be restricted to prevent a user from programming potentially malicious (partial) bitstreams. In this paper we show a first analysis on required fundamental restrictions, and propose a flow how they can be enforced with bitstream checking. Three attack signatures are formulated and checked on technology mapped netlists, and we evaluate two of them on known attacker designs.

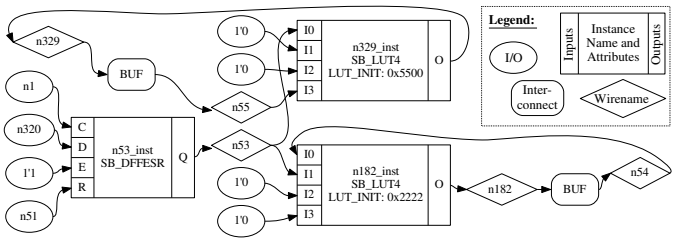

Figure 6: Combinational cycle extraction generated from modified yosys on an example design with 2×ROs. The *SB_DFFESR* register is responsible for enabling the ROs, and the shown *SB_LUT4* implement inversions in two variants of INIT configurations (0x5500, 0x2222).

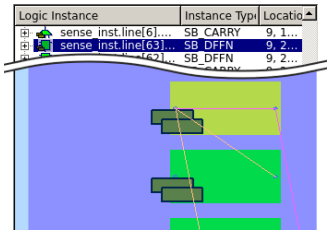

Figure 7: End node of most critical path in the original design

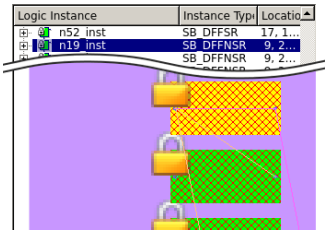

Figure 8: End node of most critical path in the reversed design

References

[1] "Inside the Samsung Galaxy S5," Chipworks, 2014. [Online]. Available: https://www.chipworks.com/ko/node/126

[2] K. Eguro and R. Venkatesan, "FPGAs for trusted cloud computing," in *FPL*, 2012.

[3] S. A. Fahmy, K. Vipin, and S. Shreejith, "Virtualized FPGA Accelerators for Efficient Cloud Computing," in *CloudCom*, 2015.

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

[5] Instance type families – Alibaba Cloud Documentation Center. [Online]. Available: https://www.alibabacloud.com/help/doc-detail/25378.html

[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] F. Schellenberg, D. R. E. Gnad, A. Moradi, and M. B. Tahoori, "An Inside Job: Remote Power Analysis Attacks on FPGAs," in *DATE*, 2018.

[8] M. Zhao and G. E. Suh, "FPGA-Based Remote Power Side-Channel Attacks," in *S&P*, 2018.

[9] C. Ramesh *et al.*, "FPGA side channel attacks without physical access," in *FCCM*, 2018.

[10] J. Krautter, D. R. E. Gnad, and M. B. Tahoori, "FPGAhammer: Remote Voltage Fault Attacks on Shared FPGAs, suitable for DFA on AES," *TCHES*, 2018.

[11] F. Schellenberg, D. Gnad, A. Moradi, and M. Tahoori, "Remote Inter-Chip Power Analysis Side-Channel Attacks at Board-Level," in *ICCAD*, 2018.

[12] T. Huffmire *et al.*, "Moats and Drawbridges: An Isolation Primitive for Reconfigurable Hardware Based Systems," in *S&P*, 2007.

[13] D. Boneh, R. A. DeMillo, and R. J. Lipton, "On the Importance of Checking Cryptographic Protocols for Faults," in *EUROCRYPT*, 1997.

[14] P. Kocher, J. Jaffe, and B. Jun, "Differential Power Analysis," in *CRYPTO*, 1999.

[15] "FPGA Device Feature List (DFL) Device Drivers," Linux Weekly News. [Online]. Available: https://lwn.net/Articles/757283/

[16] C. Beckhoff, D. Koch, and J. Torresen, "Short-Circuits on FPGAs Caused by Partial Runtime Reconfiguration," in *FPL*, 2010.

[17] C. Wolf, "Project IceStorm." [Online]. Available: http://www.clifford.at/icestorm/

[18] ——, "Yosys Open SYnthesis Suite." [Online]. Available: http://www.clifford.at/yosys/

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

[20] K. M. Zick and J. P. Hayes, "Low-cost Sensing with Ring Oscillator Arrays for Healthier Reconfigurable Systems," *TRETS*, 2012.

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