Fault Recovery from Multi-Tenant FPGA Voltage Attacks

Shayan Moini, Dhruv Kansagara, Daniel Holcomb, Russell Tessier
Department of Electrical and Computer Engineering
University of Massachusetts Amherst
Amherst, MA, USA
{smoini,dkansagara,dholcomb,tessier}@umass.edu

ABSTRACT

As multi-tenant FPGA applications continue to scale in size and complexity, their need for resilience against environmental effects and malicious actions continues to grow. To ensure continuously correct computation, faults in the compute fabric must be identified, isolated, and suppressed in the nanosecond to microsecond range. In this paper, we detail a circuit and system-level methodology to detect compute failure conditions due to on-FPGA voltage attacks. Our approach rapidly suppresses incorrect results and regenerates potentially-tainted results before they propagate, allowing time for an attacker to be suppressed. Instrumentation includes voltage sensors to detect error conditions induced by attackers. This analysis is paired with focused remediation approaches involving data buffering, fault suppression, results recalculation, and computation restart. Our approach has been demonstrated using an RSA encryption circuit implemented on a Stratix 10 FPGA. We show that a voltage attack using on-FPGA power wasters can be effectively detected and computation halted in 15 ns, preventing the injection of timing faults. Potentially tainted results are successfully regenerated, allowing for fault-free circuit operation. A full characterization of the latency and resource overheads of fault detection and recovery is provided.

CCS CONCEPTS

• **Security and privacy → Intrusion detection systems**; **Side-channel analysis and countermeasures**; **Hardware security implementation**; **Hardware attacks and countermeasures**; • **Hardware → Hardware accelerators**; **Fault tolerance**.

KEYWORDS

Hardware Security, Fault Detection and Recovery, On-chip Voltage Sensor

ACM Reference Format:
Shayan Moini, Dhruv Kansagara, Daniel Holcomb, Russell Tessier. 2023. Fault Recovery from Multi-Tenant FPGA Voltage Attacks. In *Proceedings of the Great Lakes Symposium on VLSI 2023 (GLSVLSI '23), June 5–7, 2023, Knoxville, TN, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3583781.3590246

GLSVLSI '23, June 5–7, 2023, Knoxville, TN, USA

ACM ISBN 979-8-4007-0125-2/23/06…$15.00
https://doi.org/10.1145/3583781.3590246

1 INTRODUCTION

FPGAs are now used in a wide range of machine learning [25], networking [12], and data mining applications [26], among others. To support these applications, FPGA designs often include intellectual property cores that potentially could include malicious circuitry. These multi-tenant scenarios and others that could arise from multiple independent customers sharing a cloud FPGA at the same time [5] raise the risk of malicious on-FPGA attacks. It is well-known that on-FPGA voltage can be easily manipulated using simple power wasting logic circuits, causing faults in unsuspecting user designs [11, 14, 20]. These fault injection attacks can compromise encryption circuits [13]. To ensure correct and uncompromised system operation, these attacks must be identified in the nanosecond to microsecond range and remediation must be taken to avoid the propagation of faulty results. In this paper, we outline a circuit and system-level methodology to detect timing failure conditions due to a voltage attack, rapidly suppress incorrect results generation, and regenerate potentially-tainted results.

Fault recovery for FPGAs has generally been focused on bitstream [18] and system-level [21] errors. Bit flips due to radiation are addressed using hardware redundancy [8] at the cost of significant overhead or partial FPGA reconfiguration [18]. At the system level, computation checkpoints and rollback are used to recover FPGA computation that has incurred errors [2, 21]. This approach requires significant checkpoint storage. Due to resource constraints, these approaches cannot be easily adapted to malicious multi-tenant computing activity in embedded and cloud environments.

Our approach successfully protects against voltage-induced timing faults at the functional block level. Input and output values of functional blocks (e.g., datapath elements) are stored in shallow shift registers until timing correctness can be assured. If an on-FPGA voltage sensor detects a significant voltage drop, pipeline data into the block is stalled and the potentially-faulty output data is flushed. Once the attack has been cleared, the stored input values are used to recalculate output values. Computation then proceeds normally. In our system, a time-to-digital converter (TDC) based voltage sensor is used. The amount of required buffering depends on the response time following voltage droop detection.

Our protection approach has been validated using a Stratix 10 1SX280 FPGA on a commercial DE10-Pro board [24]. Characterization and calibration was performed using a 512-bit adder operating at 200 MHz. Information gained from this evaluation was used to protect the 512-bit adder that serves as the primary computation component in a 1,024-bit Rivest-Shamir-Adleman (RSA) encryption circuit [17]. The protection mechanism is carefully detailed to examine latency penalties and resource overheads incurred in providing fault protection against ring oscillator (RO) based power wasters.

2 BACKGROUND AND RELATED WORK

FPGA multi-tenancy exists in many forms [9]. For example, FPGA applications often use intellectual property (IP) cores and interfaces (e.g., *shells*) provided by third-party vendors. Additionally, although the ability of multiple independent customers to share a commercial cloud FPGA is not currently supported, it is expected in the near future [10]. Thus, run-time protection against multi-tenant voltage attacks is an important concern [3].

Fault recovery in FPGAs has primarily focused on recovery from single-bit upsets due to radiation. For example, triple modular redundancy (TMR) triples the implementation of critical hardware components and uses majority voting to select valid outputs [8]. Underlying configuration bit upsets can be addressed via partial reconfiguration (also known as scrubbing [7]). These approaches are ineffective in addressing faults caused by voltage attacks since all replicated functional blocks with the same timing characteristics may be affected and the configuration memory is generally left unchanged.

Although voltage attack suppression has been addressed for both synchronous [20] and asynchronous [16] power wasters, fault recovery in multi-tenent FPGAs remains a concern. Timing faults in FPGAs can be addressed at both fine- and coarse-grained levels. A Razor implementation [4] duplicates flip flops in critical IP blocks in a effort to identify and correct timing faults. The clock signal for shadow flip flops arrives slightly later than for the original flops, providing for single signal recovery. Although suitable for processor pipelines, this fine-grained approach becomes inefficient for even modest-sized IP cores. Additionally, the custom variable-threshold transistor pair circuitry used for detecting possible flip-flop metastability is not feasible in FPGA implementations [22]. In contrast, computation checkpointing and rollback provides a system-level approach to FPGA fault recovery. Kilobytes of FPGA state are periodically stored at time checkpoints [2, 21] as computation proceeds. If computation errors are detected, state can be rolled back to the nearest checkpoint. Checkpointing requires user design modifications to facilitate fast interruption and rollback [1] that can incur a performance overhead.

Our approach fits between these fine- and coarse-grained techniques. Critical functional block inputs are retained in shallow buffers for a few cycles until core results can be assured to be unaffected by a potential attack. As a result, recovery storage and control circuitry is greatly simplified versus Razor and checkpointing approaches. Our approach has some similarities to microprocessor micro rollback [23] that is applied to individual pipeline stage inputs. In our case, we focus only on critical functional blocks inputs rather than on all pipeline stages. Another remediation approach detects FPGA faults and rolls back computation tens of clock cycles [15]. Our approach supports results regeneration in five or fewer cycles.

Time-to-digital (TDC) sensors have been previously used to detect the presence of power wasting circuits in an FPGA [6, 15, 27]. We use a TDC sensor to detect the activation of power wasters and deploy fault recovery. The integration of both detection and real-time, rapid recovery demonstrated in FPGA hardware forms the core of our contribution.

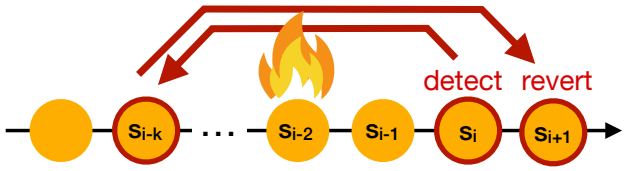

Figure 1: Sequence of states illustrates detection and recovery once the attack has been suppressed.

3 DETAILS OF THE RECOVERY SYSTEM

3.1 Overview

Our approach considers the protection of various types of functional blocks with voltage sensors that can be monitored in real time. These digital system blocks are responsive to computation halt and restart, as needed. Since the fast response times of the functional blocks are critical to safety, the responsiveness of blocks can be formalized with temporal logic properties that express latency bounds at the interface between sensor-based control and the block logic which responds to attack detection. Design requirements include:

- On attack detection, a functional block must move to a *safe* state within k cycles; In this case, *safe* indicates that no further potentially faulty computation will be performed, not that the attack has been suppressed.
- On attack detection, a block must undo k cycles of computation and revert to state from k cycles prior;
- A block will remain in state s_{i-k} until the attack is resolved. Then, the k potentially corrupted computations must be recalculated.

The requirements are depicted in Fig. 1. The value k will generally be small, as shown in our examples. Blocks protected by these requirements have competing objectives of performance (latency and throughput), and robustness. For high assurance, the properties could be verified at block level using a model checker.

3.2 Implementation

A high-level overview of our fault detection and recovery approach is shown in Fig. 2. An input shift register of depth k (bottom, left in the figure) preserves input data so results can be recalculated if timing errors caused by voltage attacks may have occurred. The synchronous output from the victim circuit is stored in a shift register of depth k (bottom, right in the figure). A value that emerges from the shift register output is assumed to be valid. Values inside the shift register may or may not be valid based on the TDC sensor value. The output shift register effectively serves as a buffer for results until they can be assured to be fault free. In the absence of faults, the output from the *Victim Circuit* is delayed by k cycles. As we demonstrate in Section 4, k is limited to two clock cycles in our 130 MHz RSA circuit, the amount of time needed to detect a voltage attack and initiate fault suppression.

Our fault detection system uses a time-to-digital converter (TDC) to identify sudden voltage drops. The 128-stage TDC contains a chain of fast fixed-purpose FPGA elements typically used to perform timing-critical arithmetic carry operations. The TDC computation is started by the rising edge of a 200 MHz clock signal. A set of

flip flops triggered by the same clock phase shifted 750 ps is used to determine how far the the adder carry propagates through the chain. The 128-bit value stored in the flip flops is interpreted as a Hamming weight. Large voltage drops across the FPGA power distribution network (PDN) result in a higher Hamming weight in the TDC output. TDC element placement is constrained to provide a predictable delay that is matched to the TDC clock frequency. During a fault-inducing voltage drop, the TDC input clock pulse will be delayed in traversing the carry chain, signaling a potential attack. Prior to design deployment, the sensor is calibrated to determine its maximum Hamming weight during normal design operation (e.g., a threshold).

During the design operation, each TDC Hamming weight reading is compared to the threshold value (Controller in Figure 2). If a TDC reading above the threshold value is detected, the *Unsafe* signal is triggered, stalling further operation in the *Victim Circuit* until voltage readings return to a safe level (e.g., the attack has been suppressed). The input shift register is also stalled, preserving saved inputs until attack suppression completion. An *OutValid* signal indicates the presence of valid output. As soon as the *Unsafe* signal is set, *OutValid* is negated indicating to the remainder of the user design outside of the *Victim Circuit* that the protected circuit is no longer consuming input nor generating outputs. Effectively, the *OutValid* signal can be used as a backpressure signal to suppress further input generation and as a flag to downstream logic to stop output processing until the attack has been suppressed.

Attack suppression is managed by the controller. The inverse of *OutValid*, the *Recovery Mode* signal, configures the multiplexer just before the *Victim Circuit* to obtain input from the input shift register. Once the attack has been cleared, the *Unsafe* signal is deactivated, allowing the input shift register to propagate saved, valid inputs from before the attack forward for output recalculation. The *Recovery Mode* signal is deactivated k cycles after the attack is suppressed (*Unsafe* is deactivated.) At this point, all recalculated values are in the output shift register and the reactivation of *OutValid* signals that new inputs to the *Victim Circuit* can be accepted and outputs from the output shift register are valid.

It should be noted that potentially faulty values in the output shift register are never output. As soon as *Unsafe* is asserted, these values become invalid and they are ignored.

4 ATTACK AND RECOVERY ANALYSIS

To assess our fault recovery approach, we evaluated two circuits in the presence of voltage attacks, a 512-bit adder and a complete 1,024-bit RSA circuit that includes a 512-bit adder. Attack detection and recovery circuitry were instrumented for both.

4.1 Characterization with a 512-bit adder

In an initial set of experiments involving just the adder, 15,000 single lookup-table ring oscillators (ROs) [20] were instantiated and activated probabilistically for five clock cycles per activation using an enable signal. Previous work [6] has shown that significant on-FPGA voltage drops can be achieved by toggling power wasters on and off at a fixed rate. For this work, we assume that the toggling occurs probabilistically, making it difficult to defend against repeated attacks. During each clock cycle, the wasters are

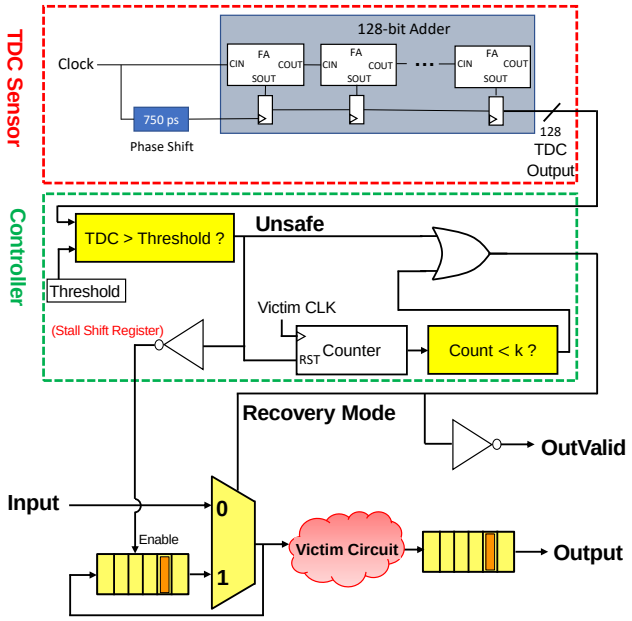

Figure 2: Detailed view of the voltage sensing and recovery system.

randomly enabled with a preset probability. The rate of activation ranged from 0% of clock cycles (never activated) to 6% of cycles. In other words, 0% indicates that the wasters are never enabled and 6% indicates a 6% probability that the wasters will be enabled in a given clock cycle. Once enabled, the wasters remain active for five clock cycles. Effectively, this approach induces significant voltage drops at unpredictable times, providing an ideal scenario to test the effectiveness of our recovery approach.

Each 512-bit vector used as input into the adder was randomly selected from a fixed set of vectors by an input generator module. Each vector stimulates an adder carry chain path of length between 80 and 440 stages. The power wasters were placed adjacent to the victim and TDC in the FPGA. It has previously been shown that power wasters located in an FPGA die can induce faults [19].

For the adder circuit, which operates at 200 MHz, the length of the input and output shift registers is $k = 5$. The sequence of signal activations associated with attack detection, triggering of *Unsafe* and *Recovery Mode* signals is shown in Fig. 3. The *PW Enable* signal activates the power wasters. The effect of power waster activation appears in the TDC-based sensor within one clock cycle. The *threshold* was set at 70 following pre-experiment calibration. Once the TDC value exceeds the threshold, the *Unsafe* signal is triggered, the entire design, including the input shift register, is stalled and the *Recovery Mode* signal is asserted. The *Recovery Mode* signal switches the input of the victim circuit to the input values stored in the input shift register although shifting is not enabled until the attack is suppressed. The *OutValid* signal, which is the negation of the *Recovery Mode* signal, is de-asserted to invalidate the output values and indicate that circuitry surrounding the protected circuit should stall. Once the *Unsafe* signal is deactivated (e.g., the

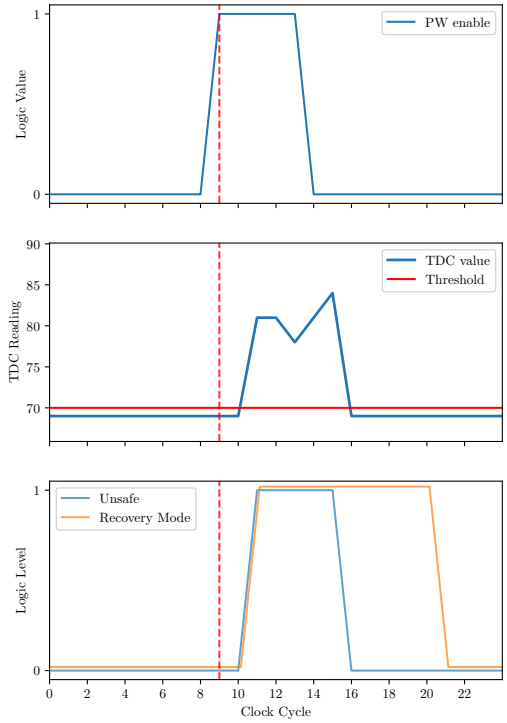

Figure 3: Recovery signal sequence when an attack is detected for a stand-alone adder with k=5.

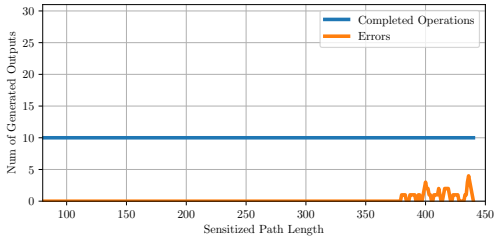

Figure 4: Unprotected system, power wasters activated with probability 3.5%.

attack is over), the *Recovery Mode* signal is de-asserted after k clock cycles to account for the delay in recomputing the potentially faulty results using the input values stored in the shift register, and normal operation continues. As annotated in Fig. 3, two clock cycles are consumed from the activation of the power wasters until the *Unsafe* signal is raised. Attacks that occur during the recovery period can be tolerated since input values are buffered until their output is guaranteed to be fault-free.

Fig. 4 illustrates the need for fault protection. To generate these results, experiments using adder input vectors of different lengths were performed. Ten experiments were performed for each vector length. Experiments were performed both with and without power waster activation. During normal operation (e.g., no power wasters activated) the full carry chain in the 512-bit adder can be used with

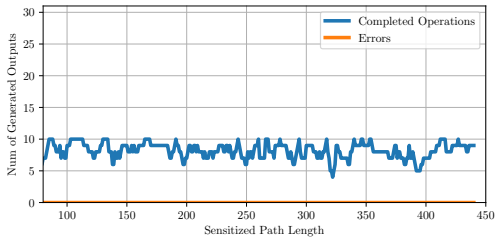

Figure 5: Protected system, power wasters on during same time span as computation in Fig. 4. Power wasters activated with probability 3.5%

no faults at 200 MHz. However, as shown in Fig. 4, when power wasters are enabled, faults are induced in the longer path lengths (e.g., length 380 to 440). In the experiments that generated the figure, the power wasters were enabled with a probability of 3.5%. The orange line in the figure illustrates how many trials resulted in faults. Fig. 5 illustrates the benefits and limitations of activating fault recovery. The results for each path length indicate the number of successfully completed operations in the same time period as each point on the blue line in Fig. 4. Since recomputation is performed, necessarily fewer operations are completed, although all of them are fault-free.

Effectively, fault recovery reduces the overall throughput of the circuit since some results need to be recalculated using buffered input values to overcome potential faults when power wasters are activated.

Throughput loss can be better quantified across a range of power waster activation probabilities. As seen in Fig. 6, as wasters are enabled at a rate approaching 6%, throughput is reduced by about 30% as the rate of output recalculation necessarily increases. In most cases, attackers will be quickly identified and suppressed, limiting attack frequency to a very low probability after the first activation.

Determining the shift register depth (k) is important in assuring fault-free operation. Faulty results generated due to the activation of power wasters have a higher chance of escaping the design with short shift register length. Fig. 7 shows the probability of error for different shift register depth values when the power wasters are activated with 3% probability for the 512-bit adder. Shift register sizes larger than four result in error-free operation.

Relaxing the TDC threshold affects the error probability of the results. A lower TDC threshold can give higher chance of recovering from all faults. In a final experiment with the 512-bit adder, we examine how relaxing the TDC threshold affects error probability and design throughput. Fig. 8 indicates that TDC threshold values that are greater than 72 lead to improved throughput at the cost of undetected faults. Since reliable operation is paramount, the selection of a fault-free TDC threshold during calibration is critical. Note that both the probability of error and throughput increase non-linearly with threshold. Higher thresholds allow for high throughput since fewer recalculations are performed. However, the likelihood of errors is much higher than the case when the predetermined threshold is used.

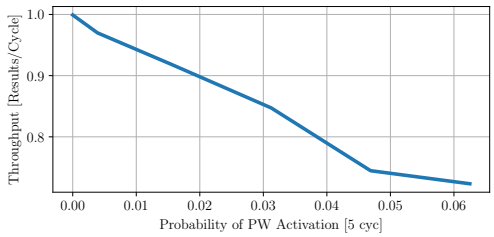

Figure 6: Throughput versus probability of power waster activation for a 512-bit adder with a TDC threshold of 70.

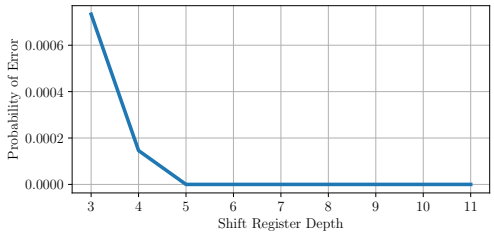

Figure 7: Probability of error as a function of the shift register depth for the 512-bit adder

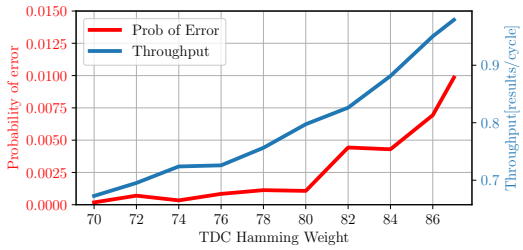

Figure 8: Fault probability and throughput versus TDC Hamming weight threshold for a PW activation rate of 6%.

4.2 Fault Recovery of an RSA adder

To evaluate the functionality of our fault recovery approach, we used the circuitry in Fig. 2 to protect the 512-bit adder in a 1,024-bit RSA encryption circuit. The adder was the only subcircuit which required protection since it was, by far, the most critical in terms of timing. Faults induced by the power wasters were determined to only affect the adder, not other RSA subcircuits.

The RSA circuit uses the Chinese Remainder Theorem (CRT) to speed up the operation of RSA encryption [17]. The CRT implementation of RSA requires the availability of the two large random numbers p and q which are used to generate the encryption key pair. It allows for the division of the exponent size in the fast exponentiation operation by two (512-bit exponentiation for 1024-bit RSA) resulting in an acceleration of the RSA encryption operation by a factor of four. A 512-bit adder is instantiated in the modular multiplication unit of the RSA encryption circuit which is used for fast exponentiation. A single run of RSA encryption requires ~3.6 million clock cycles. About 26% of these clock cycles are spent on ~968,000 512-bit addition operations, all of which are performed using a single 512-bit adder protected by our method. For this design, the protected 512-bit adder operates at 130 MHz. Voltage attacks induced by the 15,000 RO power wasters for this circuit can be identified within two 130 MHz clock cycles, leading to input and output shift registers sizes of $k = 2$, and a reaction time of 15 nanoseconds. The smaller shift register size compared to the 512-bit adder in the previous section is due to the differing circuit clock frequencies. This effect leads to a shorter delay between the deactivation of the power wasters and the end of recomputation. The long combinational path of the adder and its high operational frequency assure that it is the only circuit in the RSA affected by the voltage attack.

A threshold value for the TDC measurement was determined to be 68 through calibration. The RSA encryption circuit, without implementing the proposed fault recovery method, performs one RSA encryption on an input message block every 57 milliseconds. Adding the fault recovery circuitry introduced in Fig. 2 and setting the shift register size as $k = 2$, results in increasing the RSA circuit delay to 72 ms, a 26.3% increase compared to the base design. If the power wasters are enabled with the probability of 6%, the delay of the RSA circuit with fault recovery increases to 80 milliseconds, a 40.3% latency increase compared to the base design, while ensuring fault free operation in the presence of power wasters. This latency penalty is due to the re-computation of adder results after detecting the activation of the power waster.

The resource utilization of the RSA encryption circuit is shown in Table 1. The proposed recovery method adds an overhead of 6.0% to the look-up table (LUT) resources of the RSA circuit and 5.4% of its flop-flop (register) resources, while successfully protecting the design against faults injected by power wasters.

Table 1: resource utilization of the RSA encryption circuit and fault recovery method. Overhead row shows the added utilization overhead of the proposed fault recovery method

Circuit Name	# LUTs (Avail. 933K)	# Regs (Avail. 1.86M)
RSA Encryption	14825 (1.6%)	21277 (1.1%)
Recovery Circuit	900 (<1%)	1159 (<1%)
Overhead	6.0%	5.4%

4.3 Discussion

Our results lead to several observations and system optimizations, as noted below:

- The TDC threshold value for both the RSA and adder circuits remained stable across a range of operating conditions. However, to allow for flexibility, the TDC threshold value is written into a register at the beginning of computation and could be adjusted without requiring design recompilation if aging or other environmental effects are present.
- Our attack detection and recovery approach can be used for a range of feed-forward circuits, including multi-cycle circuits. The depth of the shift registers k indicates how many cycles are needed for recovery.

- Multiple close-to-critical path subcircuits could be protected by replicating the TDC sensor and protection circuit.
- Our focus on protecting critical path logic paths as opposed to all the logic circuits on the FPGA allows for low resource utilization overhead and no maximum frequency (F_{max}) performance overhead.

4.4 Comparison to Previous Work

Our approach has similarities to two recent attack protection and recovery techniques. Attia and Betz [2] propose an FPGA checkpointing approach that allows for state collection from flip flops, digital signal processing (DSP) blocks, and block RAMs. Since inputs to the latter two components can be difficult to retrieve following a fault, shadow registers are provided to buffer input values externally. Although this approach is similar to our input shift register technique, checkpointing collects large amounts of data for a much delayed computation restart. Our approach stores a small amount of input data ($k \leq 5$) and restarts computation a few cycles after an attack is cleared. Additionally, our method does not introduce a victim circuit timing penalty as opposed to an average checkpointing F_{max} performance overhead of 5.5% [2].

Luo and Xu [15] propose a full FPGA fault recovery system that uses a voltage sensor to detect a fault injection attack. Their system has the ability to regenerate faulty results after an attack by rereading input values from bulk on-chip storage. This technique resulted in a 47 cycle delay for output regeneration for an AES encryption circuit following a voltage attack. Our approach results in overall faster recovery for feed-forward circuits.

5 CONCLUSION

Recent FPGA usage trends towards multi-tenancy have introduced new security challenges due to the physical co-location of potential adversaries with their victims. This co-location increases the potential for remote fault injection attacks via voltage manipulation. In this paper, a new system for the automatic detection of voltage-based fault injection attacks and the subsequent recovery of potentially faulty computation results is presented. The system can react to the activation of fault-injecting power wasters in 15 nanoseconds for an RSA encryption circuit. In all instances, the victim is able to successfully recover from fault injection attacks with zero output faults at the cost of tolerable latency penalties. Future work will include the evaluation of countermeasures to disable power waster circuits prior to fault injection, as well as the assessment of the fault recovery approach in sophisticated circuits with complex state machines. Our current approach is also directly applicable to additional circuits with long critical paths such as Diffie-Hellman key exchange and elliptic curve calculation.

REFERENCES

[1] Sameh Attia and Vaughn Betz. 2020. Feel free to interrupt: Safe task stopping to enable FPGA checkpointing and context switching. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)* 13, 1 (2020), 1–27.

[2] Sameh Attia and Vaughn Betz. 2020. StateReveal: Enabling Checkpointing of FPGA Designs with Buried State. In *International Conference on Field Programmable Technology*. 206–214.

[3] Christophe Bobda, Joel Mandebi Mbongue, Paul Chow, Mohammad Ewais, Naif Tarafdar, Juan Camilo Vega, Ken Eguro, Dirk Koch, Suranga Handagala, Miriam Leeser, et al. 2022. The future of FPGA acceleration in datacenters and the cloud. *ACM Transactions on Reconfigurable Systems and Technology (TRETS)* 15, 3 (2022), 1–42.

[4] Dan Ernst et al. 2003. Razor: A low-power pipeline based on circuit-level timing speculation. In *IEEE/ACM International Symposium on Microarchitecture*. 7–18.

[5] Ognjen Glamočanin, Louis Coulon, Francesco Regazzoni, and Mirjana Stojilović. 2020. Are cloud FPGAs really vulnerable to power analysis attacks?. In *Design, Automation & Test in Europe Conference & Exhibition*. 1007–1010.

[6] Dennis RE Gnad, Fabian Oboril, and Mehdi B Tahoori. 2017. Voltage drop-based fault attacks on FPGAs using valid bitstreams. In *International Conference on Field Programmable Logic and Applications*. 1–7.

[7] Jonathan Heiner, Benjamin Sellers, Michael Wirthlin, and Jeff Kalb. 2009. FPGA partial reconfiguration via configuration scrubbing. In *International Conference on Field Programmable Logic and Applications*. 99–104.

[8] Adam Jacobs, Grzegorz Cieslewski, Alan George, Ann Gordon-Ross, and Herman Lam. 2012. Reconfigurable Fault Tolerance: A Comprehensive Framework for Reliable and Adaptive FPGA-Based Space Computing. *ACM Transactions on Embedded Computing Systems* 5, 4 (Dec. 2012), 21:1–21:31.

[9] Chenglu Jin, Vasudev Gohil, Ramesh Karri, and Jeyavijayan Rajendran. 2020. Security of Cloud FPGAs: A Survey. *arxiv* arXiv:2005.04867 (2020). http://arxiv.org/abs/2005.04867

[10] Ahmed Khawaja, Joshua Landgraf, Rohith Prakash, Michael Wei, Eric Schkufza, and Christopher J Rossbach. 2018. Sharing, Protection, and Compatibility for Reconfigurable Fabric with AmorphOS. In *USENIX Symposium on Operating Systems Design and Implementation*. 107–127.

[11] Jonas Krautter, Dennis R. E. Gnad, and Mehdi B. Tahoori. 2018. FPGAhammer: Remote Voltage Fault Attacks on Shared FPGAs, suitable for DFA on AES. *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2018, 3 (Aug. 2018), 44–68.

[12] Ricardo Lent. 2021. Towards Cognitive Networking using FPGA-Based Acceleration. *IEEE Journal of Radio Frequency Identification* 5 (April 2021), 222 – 231.

[13] Xiang Li, Russell Tessier, and Daniel Holcomb. 2022. Precise Fault Injection to Enable DFIA for Attacking AES in Remote FPGAs. In *IEEE International Symposium on Field-Programmable Custom Computing Machines*. IEEE, 1–5.

[14] Yukui Luo, Cheng Gongye, Yunsi Fei, and Xiaolin Xu. 2021. Deepstrike: Remotely-guided fault injection attacks on DNN accelerator in cloud FPGA. In *ACM/IEEE Design Automation Conference*. IEEE, 295–300.

[15] Yukui Luo and Xiaolin Xu. 2020. A Quantitative Defense Framework against Power Attacks on Multi-tenant FPGA. In *ACM/IEEE International Conference on Computer Aided Design*. 1–9.

[16] Hassan Nassar, Hanna AlZughbi, Dennis R. E. Gnad, Lars Bauer, Mehdi B. Tahoori, and Jörg Henkel. 2021. LoopBreaker: Disabling Interconnects to Mitigate Voltage-Based Attacks in Multi-Tenant FPGAs. In *ACM/IEEE International Conference on Computer Aided Design*. 1–9.

[17] Christof Paar and Jan Pelzl. 2009. *Understanding Cryptography: a Textbook for Students and Practitioners*. Number 7. Springer Science & Business Media, Chapter 7, 173–204.

[18] Matthew Parris, Carthik Sharma, and Ronald Demara. 2011. Progress in Autonomous Fault Recovery of Field Programmable Gate Arrays. *Comput. Surveys* 43, 4 (Oct. 2011), 31:1–31:30.

[19] George Provelengios, Daniel Holcomb, and Russell Tessier. 2020. Power distribution attacks in multitenant FPGAs. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 28, 12 (2020), 2685–2698.

[20] George Provelengios, Daniel Holcomb, and Russell Tessier. 2021. Mitigating Voltage Attacks in Multi-Tenant FPGAs. *ACM Transactions on Reconfigurable Systems and Technology (TRETS)* 14, 12 (June 2021).

[21] Andrew Schmidt, Bin Huang, Ron Sass, and Matthew French. 2011. Checkpoint/Restart and Beyond: Resilient High Performance Computing with FPGAs. In *IEEE International Symposium on Field-Programmable Custom Computing Machines*. 162–169.

[22] Edward Stott, Joshua M Levine, Peter YK Cheung, and Nachiket Kapre. 2014. Timing fault detection in FPGA-based circuits. In *IEEE International Symposium on Field-Programmable Custom Computing Machines*. 96–99.

[23] Yuval Tamir, Marc Tremblay, and David A. Rennels. 1988. The Implementation and Application of Micro Rollback in Fault-Tolerant VLSI Systems. In *Fault-Tolerant Computing Symposium*. 234–239.

[24] Terasic Technologies 2019. *DE10-Pro User's Manual*. Terasic Technologies.

[25] Shulin Zeng, Guohao Dai, Kai Zhong, Hanbo Sun, Guangjun Ge, Kaiyuan Guo, Yu Wang, and Huazhong Yang. 2020. Enabling efficient and flexible FPGA virtualization for deep learning in the cloud. In *IEEE International Symposium on Field-Programmable Custom Computing Machines*. 102–110.

[26] Bingyi Zhang, Rajgopal Kannan, and Viktor Prasanna. 2021. BoostGCN: A Framework for Optimizing GCN Inference on FPGA. In *IEEE International Symposium on Field-Programmable Custom Computing Machines*. 29–39.

[27] Kenneth M Zick, Meeta Srivastav, Wei Zhang, and Matthew French. 2013. Sensing nanosecond-scale voltage attacks and natural transients in FPGAs. In *ACM/SIGDA International Symposium on Field Programmable Gate Arrays*. 101–104.