RAM-Jam: Remote Temperature and Voltage Fault Attack on FPGAs using Memory Collisions

Md Mahbub Alam, Shahin Tajik, Fatemeh Ganji, Mark Tehranipoor, and Domenic Forte
Florida Institute for Cybersecurity (FICS) Research
ECE Department, University of Florida
{mahbub.alam, stajik, fganji}@ufl.edu, {tehranipoor, dforte}@ece.ufl.edu

***Abstract*—It has been demonstrated that with concrete hardware Trojans, a remote adversary can mount physical attacks, e.g., fault or side-channel attacks, against adjacent IP cores in an FPGA. In this work, we present a novel remote fault attack, called RAM-Jam, which exploits an existing weakness in the dual port RAMs of mainstream FPGAs. The possibility of concurrent writing of opposite logic values into these RAMs not only leads to data uncertainty but also causes transient short circuits. With a sufficient number of RAM collisions, there are severe voltage drops and excessive heat that result in timing faults as well as bit-flips in the FPGA's configuration memory. We conduct extensive experiments to evaluate the effectiveness of our fault injection technique and further present attacks against two applications, including a soft authentication scheme and the first remote fault attack against a deep neural network. Finally, we discuss potential countermeasures to prevent such attacks.**

***Index Terms*—Fault Attack; FPGA Security; Hardware Trojan; Remote; Deep Learning; Convolutional Neural Networks.**

I. Introduction

Field programmable gate arrays (FPGAs) have become indispensable parts of modern electronic systems. Due to the ever-growing number of configurable resources and flexibility on FPGAs, several intellectual property (IP) cores can coexist simultaneously on the same FPGA platform. In this case, shared resources on FPGAs can be a source of vulnerabilities, if one of the IP cores contain a hardware Trojan. Such hardware Trojans can be inserted in third party IP cores, where the cores are developed by untrusted users/entities involved in the supply chain and FPGA upgrade life cycle.

As a conventional solution, FPGA vendors are providing isolation schemes that separate different IP cores on the chip by creating spatial moats to avoid any communication or fault propagation between them. However, these moats provide logical rather than physical isolation. In other words, although a malicious circuitry in an IP has no logical access to other IPs, it can still exploit existing weaknesses at the physical level to cause damage to other IPs or leak information from them [1].

Such vulnerabilities have gotten a big deal of attention in the research community, see Fig 1. It has been shown that the exploitation of these vulnerabilities opens the doors for new advanced side-channel, fault injection, and covert-channel attacks, which can be mounted remotely by an adversary who has partial access to the FPGA. For instance, it has been demonstrated that extensive deployment of ring-oscillators (ROs) in an IP, can overheat the FPGA and drop the voltage in its power distribution network (PDN). In this case, either the functionality of the entire FPGA might be disrupted, or other IP cores might behave in an unpredictable manner and leak information, which can be exploited remotely by an adversary to extract assets, such as cryptographic keys [2]. Specific combinational circuits, such as long buffer chain-based time to digital converters (TDCs), can also be embedded in an IP core to spy and conduct side-channel analysis on the adjacent IP cores by measuring the voltage or temperature fluctuations. This can lead to leakage of cryptographic keys and other confidential data [3], [4]. However, since these hardware Trojans contain specific types of circuits, e.g., combinatorial loops, they leave signatures in the bitstream which can be detected with a high probability [5]. A key question is whether there exists a hardware Trojan that can be included in a valid bitstream without leaving a specific trace.

Our Contribution. In this work, we reveal a new vulnerability in FPGAs, which causes severe voltage drop and temperature increase inside the chip. The vulnerability relies on the fact that the FPGAs designed by major vendors (i.e., Xilinx, Intel, Microsemi, etc.) contain dual-port RAMs, where concurrent write operations to the same address are possible. Writing simultaneously opposite logic values to an address creates a memory collision, which leads to a transient short circuit. We demonstrate that creating sufficient memory collisions leads to a drop in voltage and an increase in temperature of the FPGA. Consequently, timing violations or bit-flips in configuration memory cells of the FPGA can occur. Therefore, an adversary can exploit this vulnerability as a hardware Trojan, called RAM-Jam in this paper, to mount remote attacks against adjacent IPs. We conduct several experiments with a different number of writing collisions to assess their adverse effect. Besides, we present two different attacks against an authentication scheme and a machine learning application. While in the former attack, we demonstrate that how the resulted timing faults can bypass a password-based authentication, in the latter, we show how RAM-Jam can lead to the misclassification of a neural network's inputs. Finally, we propose potential countermeasures against RAM-Jam.

It is worth noting that although FPGA vendors warn users in their documentation about possible data corruptions upon simultaneous writes to the same memory location, there are no indications that multiple collisions might cause voltage glitches or overheat the chip. For instance, according to a

978-1-7281-2667-8/19/$31.00 ©2019 IEEE
DOI 10.1109/FDTC.2019.00015

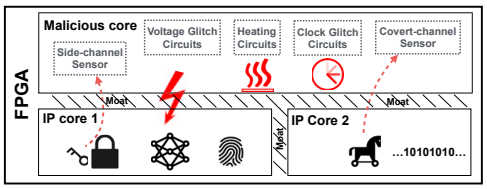

Fig. 1: Threat model considered for remote attack on FPGAs.

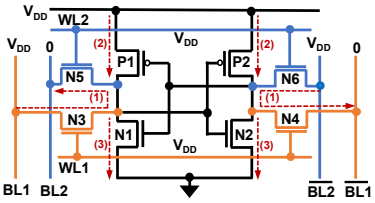

Fig. 2: Sources of short circuit paths during write collision.

Xilinx user-guide document: *"Conflicting simultaneous writes to the same location* ***never cause any physical damage*** *but can result in data uncertainty"* [6]. Based on our results, such RAM collisions have potentials indeed to cause physical damage to the FPGA. Moreover, in the same document, it is mentioned: *"There is* ***no dedicated monitor*** *to arbitrate the effect of identical addresses on both ports. It is up to you to time the two clocks appropriately"* [6]. With regard to this, in contrast to previously reported RO- and TDC-based Trojans, detection of a memory collision-based Trojan in the bitstream is more challenging since no Trojan trigger or payload is explicitly included in the design for our attack.

Our Threat Model. We assume that the FPGA resources are spatially shared between different IPs. In this case, the adversary either has partial access to a multi-tenant FPGA in a cloud or is capable of manipulating a 3rd party IP core to implant a hardware Trojan. The goal of the adversary is to remotely activate the hardware Trojan and inject a fault from the infected IP into other IPs to gain privileged access to a service, extract a cryptographic key, or simply mount a denial of service (DoS) attack. The IP cores are separated by moats on the FPGA, and therefore, there is no direct communication link between them. However, the PDN of the FPGA is shared between these IP cores. To inject a fault into adjacent IP cores and bypass the isolation, the adversary needs to design a hardware Trojan that upon activation drops the voltage in the shared PDN or overheats the chip. To pass the bitstream security checks, the Trojan circuit has to be included in a valid configuration, otherwise, its detection would be trivial.

II. RAM-Jam: Memory Collision Induced Fault Injection

A. Basics of Memory Collisions

In general, FPGAs offer two different types of embedded memory: block RAMs (i.e., dedicated RAMs) and distributed RAMs (i.e., logic cells or LUTs). While Block RAMs (BRAMs) are useful for fast, large, and flexible data storage, distributed RAMs are suitable for storing coefficients, state machines, and small buffers. Both of these memories can be configured as a single port or dual port RAM. The single port memories have only one data/address port, and hence, can only be read or written at a given time. However, the (true) dual-port memories have two completely independent access ports. The structure is fully symmetrical, and both ports are interchangeable. Data can be written/read to/from either or both ports simultaneously. Each write operation is synchronous, and each port has its own address, data in, data out, clock, and write enable, as shown in Fig. 5. The read and write operations are synchronous and require a clock edge.

Although dual-port RAM offers an excellent opportunity to maximize the bandwidth, it can potentially result in memory collision when accessing the same memory location from both ports at the same clock cycle. In the current state-of-the-art FPGAs, consider the following collision scenarios while designing an application: 1) *Read collision:* both ports are reading from the same address. Generally, the read operations complete successfully. 2) *Write collision:* both ports are writing different data to the same memory address. The memory location's data will be non-deterministic. 3) *Read during write or vice-versa:* one port is writing to, and the other port is reading from same memory address. Usually, the write operation is successful, but the resulting read memory value is uncertain.

Although the inner structure of SRAM-based memory is well-known, the exact implementation of the dual-port logic is vendor-specific. We expect to find a symmetric structure inside the dual-port logic as shown in Fig. 2. Similar circuit structure has been considered in [7] to analyze read-disturb-write mechanism. Considering this symmetric circuit implementation, write collision has been realized in this work. To obtain write collision, one writes logic '1' and logic '0' by using bitline $BL1$ and $BL2$ respectively. Both word lines ($WL1$ and $WL2$) become high (V_{DD}) at the same time, which causes three current paths as shown by dashed lines in Fig. 2. Since access transistors $N3$ (or $N4$) and $N5$ (or $N6$) are conducting, current flows (path 1) from one bitline to another bitline. Thus, the voltage at Q (or Q_b) changes to an indeterminate voltage between 0 and V_{DD}. Because of circuit symmetry, Q and Q_b will settle to values close to $V_{DD}/2$. Hence, transistors $P1$, $P2$, $N1$, and $N2$ of inverters will be conducting which leads to two more current paths (2 and 3). These current flows continue until SRAM cell settles to a logic value. If we repeat this collision at high frequency, SRAM cell will draw a significant amount of current from the power line, which can potentially cause a voltage drop in the power supply. Although a few write collisions do not create much impact, a large number of write collisions at the same time cause enough voltage drop and significant temperature rise.

B. Impact on Timing Constraints

Let two flip-flops $FF1$ and $FF2$ share a clock. A data signal is released from one register $FF1$ on a clock rising edge and then processed through the logic before being latched into the next register $FF2$ on the next clock rising

edge. Thus, the clock period T_{clk} has to be longer than the maximum data propagation time through the logic T_{pd} to ensure proper operation. The well-known setup time constraint is: $T_{clk} > T_{clk2q} + T_{pd,max} + T_{setup} - T_{skew}$ where T_{clk2q} delay is elapsed between the clock rising edge and the actual update of a register's output, T_{skew} is the skew or slight phase difference that may exist between the clock signals at the clock inputs of two different registers, and T_{setup} is the amount of time for which a D flip-flop input must be stable before the clock's edge to ensure reliable operation. The violation of this timing constraint is a straightforward means to inject faults into a circuit. One way of violating the timing constraint is by increasing the right-hand side of the above inequality. This may be achieved by increasing the data propagation time $T_{pd,max}$ through the combinational logic.

In the case of FPGA, the combinational logic is generally formed by look-up tables (LUTs), routing, multiplexers, etc. [8]. Routing consumes a large portion of the combinational circuit in FPGA, but it is hard to describe precisely the circuit design as it is vendor-specific. For the sake of simplicity, the pass transistor-based circuit can be considered for look-up tables and routing [8], [9]. Delay of a pass transistor circuit is approximately inversely proportional to the term $(V_{dd} - V_{th})$. Any decrease in supply voltage will induce an increase in the propagation delay of the pass transistor. By extension, the data propagation time through any logic block is increased as long as the FPGA is underpowered. Since temperature decreases transistor's mobility, it also increases propagation delay.

The proposed memory collisions lead to a sharp reduction in internal voltages and a sudden increase in temperature. Details will be discussed in Section IV-B. Both effects are capable of increasing the propagation delay of the logic and, therefore, failed timing constraints. Hence, the proposed memory collision mechanism can be used to inject faults in FPGAs. An intelligent timing of fault injection can ensure exploitable faults for various sensitive applications such as password authentication, neural network, crypto blocks, etc.

C. Impact on SRAM Storage Cell

Modern FPGAs use SRAM cells for bitstream configuration. Static noise margin (SNM) of the SRAM cell is an important parameter that determines the cell's resistance to noise during read, write, and hold operations. The cell is most vulnerable when accessed during a read operation because it must retain its state in the presence of the bitline precharge voltage. If the noise is higher than read SNM, it may lose its state during a read cycle. It is well known that SNMs are very sensitive to voltage and temperature fluctuations. Notably, the read noise margin is a strong function of the supply voltage. With the decrease of bitline and supply voltages, the read SNM decreases and becomes more vulnerable to bit-flip. Also, an elevated temperature decreases read noise margin [10]. The hold and write SNM also reduce with the decrease in supply voltage. In general, hold and write margins are higher than read SNM for six-transistors SRAM cell. Hence, SRAM cells in FPGA that only hold data are supposed to be less sensitive to supply voltage and temperature variation. On the contrary, SRAM cells used as dedicated memory are designed for random read access and more susceptible to bit-flips. In this work, we examine memory collision induced voltage lowering and temperature rising as noise sources in order to flip bits.

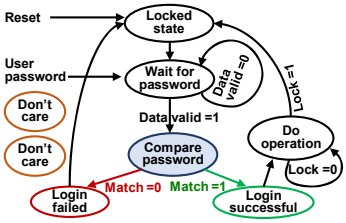

Fig. 3: A simple password authentication method.

III. RAM-Jam against Practical Applications

A. Fault Injection in Finite State Machine (FSM)

Sequential components are crucial for any digital IC as they are responsible for controlling the functionality of the entire system based on the user's real-time input and the current state. FSMs are the most popular models used to describe controller circuits of a system. The security of the overall system will be compromised if the FSM in the controller circuit is successfully attacked by injecting faults [11]. For instance, fault injections on crypto blocks, authentication modules, secure boot, and communication protocols can lead to leakage of a secure key, faulty output, and denial of service. Even if the data path is adequately protected, fault injection in FSM can leak encryption key as shown in by [12]. In this work, we explore the potential fault injection approach in an FSM using voltage and temperature glitches created by memory collision. As a proof-of-concept, we target an authentication protocol that verifies the status of a password. Note that, our approach can be extended to other applications as well.

Fig. 3 depicts the state transition diagram of an FSM that implements a password authentication protocol. The system remains in the locked condition and can only be unlocked by a correct password. The FSM is composed of 6 states: *Locked state*, *Wait for user password*, *Compare password*, *Login failed*, *Login successful* and *Do operation*. After reset, the system starts in *Locked state*. In the next rising edge of the clock, the system moves to *Wait for password* state and stays there until the user enters a password. The *Compare password* state compares the user password with the system password stored inside the system. The system password is only known to the authentic users and thus protects the system from unauthorized access. In case of password mismatch, it moves to *Login failed* state and later goes to *Locked state*. If a password matches, the user can successfully log into the system and enter *Do operation* state. The user can later return the FSM to *Locked state*.

For this FSM implementation, *Compare password* (highlighted in blue color in Fig. 3) is the most critical state because if a malicious user can bypass password comparison and move to *Login successful* state, he/she can gain access to the system

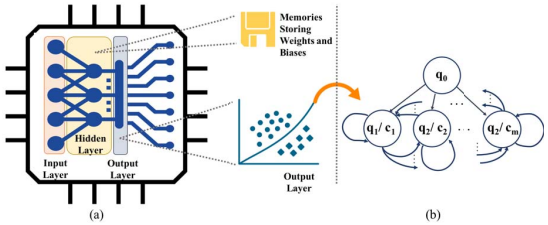

Fig. 4: (a) The architecture of a general NN pre-trained and implemented on an FPGA, and (b) An FSM representing the output layer of the NN. Here m is the number of classes. q_0 denotes the starting state of the FSM, and q_i $(1 \leq i \leq m)$ shows the state, to which the FSM transits after receiving the activation. Furthermore, a class label c_i $(1 \leq i \leq m)$ is assigned to the respective input by the output layer.

without the correct password. As described in Sec. II-B, fault injection can make an incorrect state transition occur given that a proper timing and propagation delay is achieved. Since this FSM contains 6 states and requires minimum 3 registers, at least 2 $(= 2^3 - 6)$ states remain as *don't care* (see Fig. 3). Fault injection may result in a transition to *don't care* states where the operation is undefined. The *don't cares* also provides an option to exploit fault injection in the FSM. In this work, we utilize memory collision to manipulate the supply voltage and operating temperature to inject faults in FSM. The main objective is to inject a fault during a transition that will cause the FSM to enter *Login successful* state through incorrect state transition or *don't care* states. RAM-Jam is successfully carried out in Sec. IV-C by using write collisions.

B. Fault Injection in Neural Networks

Neural networks (NNs) have gained enormous popularity over the last decade due to their wide range of applications including, but not limited to, image, audio, and video processing. Among the various types of NNs, Convolutional NNs (CNNs) are arguably the most widespread architectures employed to perform visual recognition tasks, where the data is continuous in a spatial or temporal manner. Albeit resource-hungry, the need has been felt for using CNNs, especially in embedded systems. As a prime example, autonomous cars and medical devices equipped with FPGAs embodying CNNs can be mentioned. In this regard, FPGAs offer the (re-)configurability that allows customized architectures, and their large degree of parallelism accelerates execution speeds [13].

Along with the design of hardware platforms fulfilling the requirements imposed by NNs, another line of research has been pursued that concerns how maliciously interfering with an NN can impair its performance, i.e., (potentially) result in misclassification. These malicious activities include poisoning, model reverse engineering, model stealing, membership inference, evasion and adversarial initialization (for more details, see [14]). In addition, with the emergence of fault injection methods targeting memories, injecting faults in NNs has recently attracted attention since their parameters are usually stored in memories. From a general point of view, a fault injection attack against a NN can be launched in input, hidden and output layers (see Fig. 4). The faults in the former are identical to perturbing the inputs, and therefore, are well studied in the literature. On the other hand, faults injected in hidden or output layers are interesting, particularly when the attacker does not aim to force the NN to converge to a particular class, but instead affect its reliability. Compared to other attacks on NNs, this type of fault injection attack - in particular, practical and physical ones - have been studied little thus far. For instance, Liu et al. suggest a simulation-based fault injection attack on NNs by changing some parameters, e.g., the biases [15]. However, neither a practical technique to inject a fault nor the complexity of launching such an attack is discussed. On this latter matter, before launching their attack, one has to determine the output class, to which the NN converges after increasing a bias. To address this, in another attempt, a laser fault injection attack against NNs was introduced [16], where a near-infrared diode pulse laser injected faults during the processing of the activation function. In addition to using a known, and effective method to inject faults, the authors of [16] provide insight into how their technique can be employed to target several functions in an NN and inject multiple faults simultaneously. This is of great importance as in some scenarios, similar to ours, where the goal of the adversary is to inject faults resulting in malfunction.

What RAM-Jam achieves: First, we stress that according to our threat model described in Sec. I, we are interested in injecting faults remotely in pre-trained NNs. In our attack, we target the hidden and the output layers of an NN. The mechanism of mounting an attack on the hidden layer is straightforward as we have already discussed that by mounting our attack, we can flip bits (i.e., weights of NN) stored in memories. On the other hand, the impact of our fault injection on the output layer can be modeled by the *transition* between the output classes, as illustrated in Fig. 4. More precisely, when a trained multi-class NN is given an input, after performing all calculations in the hidden layer, the results are delivered to the output layer to assign a class to the input. Consider the FSM that is in the starting state q_0, upon receiving the activation (i.e., the output from the hidden layer), the FSM goes from q_0 to q_i $(1 \leq i \leq m$, where m is the number of classes). Next, after feeding a new input into the NN, the states of the FSM is changed from q_i $(1 \leq i \leq m)$ to q_j $(1 \leq j \leq m)$. Moreover, the activation function embedded in the output layer accounts for assigning a class label c_i $(1 \leq i \leq m)$, i.e., transition between the states q_i $(1 \leq i \leq m)$. Note that this general model is valid irrespective of the activation function employed in the NN (typically, softmax, ReLu, sigmoid and tanh). Clearly, after modeling the output layer as an FSM, a similar procedure as proposed in Sec. III-A can be followed to inject a fault. Moreover, as can be understood from the above description, RAM-Jam does not require any knowledge regarding which activation function is used, on the contrary to the attack proposed in [16]. Besides this difference, we stress that RAM-Jam can be launched remotely, in contrast to the one presented in [16].

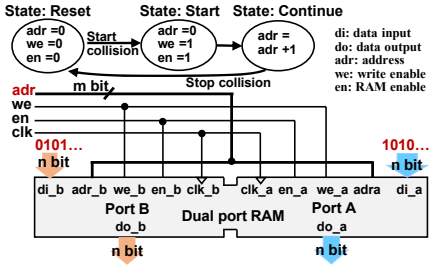

Fig. 5: Implementation of write collision in dual port RAM.

We conclude this section with the following take-home message. Besides the drastic consequences of our fault injection attack for NNs employed for classification purposes, it can further harm the performance of systems using NNs in the hope of obtaining a higher degree of fault tolerance, see, e.g., [17]. Note that the countermeasures aimed at enhancing the fault tolerance of NNs (see [17] for a comprehensive survey) are not effective against RAM-Jam.

IV. Experimental Results and Discussions

A. Experimental Setup

In this work, we implemented write collisions on Xilinx 7 series Artix-7 (XC7A100T-CS324) FPGAs within the Nexys 4 DDR Trainer Board. Artix-7 FPGAs are manufactured in $28nm$ process technology node and optimized for high-performance logic, low power, and large capacity. In addition, Artix-7 has built-in voltage and temperature sensors which can be used to monitor the internal die temperature and voltages. We utilized these built-in sensors and Xilinx Analog-to-Digital Converter (XADC) to observe the voltage and temperature during memory collision. The minimum sampling period of XADC is 1 second (s) for Artix-7 FPGA which has been considered in our experiments. The chipscope analyzer has been accessed through the JTAG chain. The JTAG interface has also been utilized to read, write, and monitor the internal signals of the hardware implementation.

We utilized logic cells or LUTs as dual port RAM to investigate the impact of large scale write collision. Fig. 5 illustrates the FPGA implementation for write collisions with major attributes: 1) Same clock signal (clk): a high-frequency clock is connected to both ports of the RAM. 2) Opposite data as input: port A and port B are connected to n bit data which are logically opposite. 3) Same address: m bit address of both ports are connected to a common address bus. The address also changes in each clock cycle to cause collisions in every possible address space. An FSM has been introduced to control the start and stop of a collision, enable write operation (we) and activate (en) RAM. Here, we initialized control parameters in *Reset* state. With the arrival of 'start collision' signal, write operation and RAM are enabled to activate the write collisions. The entire address space is hammered by changing the address in each clock cycle. Besides, the FSM clock is slower than the RAM clock. As a result, a consecutive number (RAM clock divide by the FSM clock) of write collision happens at each address during each FSM clock. A large number of

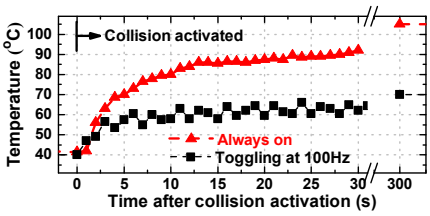

Fig. 6: Temperature characteristics during RAM-Jam.

dual port RAMs are instantiated following this approach. This nontrivial implementation is considered valid in electronic design automation tools. Hence, one can easily pass the design time check and exploit write collision as malicious code.

The data and address buses are configured as $8bit$ and $11bit$ respectively to obtain a $2KB$ dual port RAM. We vary the number of RAM usages by increasing the number of RAM instances for extensive investigation. The RAM operating frequency has been set to $500MHz$ which provides 500 collisions in every $1\mu s$. We put $8h'AA$ in one port and its complementary logic $8h'55$ in the other port. The memory addresses are sequentially accessed in every $10ns$ by implementing an FSM as described above. An 11-bit address width allows to traverse 2048 addresses of a RAM and to create a local impact on voltage and temperature. Each voltage and temperature data have been measured for 10 times, and the average value is considered to mitigate measurement noise. We use $100MHz$ FPGA onboard clock for the system. The higher frequency clocks are generated using Xilinx Clocking Wizard LogiCORE IP.

B. Mounting RAM-Jam: Experimental Results

We deployed dual port RAMs in the FPGA fabric to study the impact of memory collision on the die voltage and temperature. The memory collisions are enabled and disabled with a toggle signal. We consider two different scenarios: memory collisions are always on after activation and memory collisions are enabled and disabled at $100Hz$.

1) Impact on Die Temperature: We observed the XADC readings of voltage and temperature connected to FPGA through the JTAG chain. Fig. 6 shows the temperature characteristics. Here, 73% RAM occupation (22% LUTs) has been considered with 76000 collisions per μs. We found that temperature rises exponentially when collisions are always on. At $15s$, the temperature goes beyond $85°C$. As we know that commercial devices are specified to 0 to $85°C$, memory collision may cause improper operation by generating excessive heat. Also, we find that temperature keeps rising and reaches $105°C$ at $300s$ which makes the heating effect more significant. Hence, launching RAM-Jam, i.e., making memory collisions, can potentially create a denial of service attack. Since heating is a major problem in the newest technology devices, one can expect that the impact of memory collisions will create serious threats in newest technology FPGAs. We also toggled the activation of collisions at $100MHZ$ and found that XADC sensor reading oscillates and increases gradually. At $300s$, the temperature reaches approximately

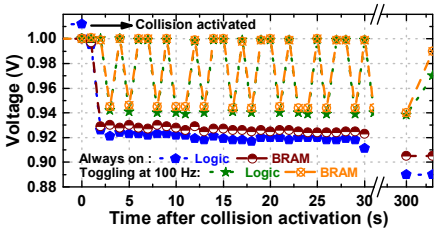

Fig. 7: Voltage characteristics during RAM-Jam.

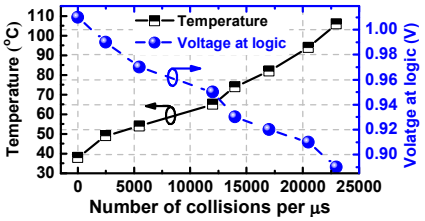

Fig. 8: Voltage and temperature values with respect to number of memory collisions.

$70°C$ and remains within FPGA specifications. As temperature changes gradually in a large device area, the built-in sensor cannot follow the sharp rise and drop of local temperature. Thus, toggling of memory collisions can be used in fault injection without being detected.

2) Impact on Internal Voltages: Voltage characteristics under memory collisions are shown in Fig. 7. Similar to temperature characterization, this experiment considered 76000 collisions per μs. We found that both internal logic core and BRAM voltages drop instantly to $0.91V$ from the nominal voltage ($1.0V$) when collisions are always on. With time both of these voltages experience a negligible drop. The instant voltage drop is approximately 10%, which is significant in time-sensitive applications. Although prior work [2] reported a similar voltage drop, their approach requires a large number of resources (40 thousand ring oscillators) and feedback loop. This work uses a different approach avoiding feedback loop and does not demand many resources. Since propagation delay increases with voltage drop caused by the memory collision, the violations of timing constraint become probable as discussed in Sec. II-B. Besides, the voltage drop in BRAM might alter the noise margin of the memory element and cause bit flips. Fig. 7 also shows the change in logic core and BRAM voltages subject to memory collision running with an activation frequency of $100Hz$. Both internal logic and BRAM voltages at XADC sensor oscillate from nominal $1.0V$ to $0.94V$. Voltage drops, in this case, are less than the always-on scenario. This can be explained by considering that sensor resolution is not enough to identify the instantaneous changes.

3) Number of Collisions vs. Temperature and Voltage: The amount of RAM has been varied in order to find correlations between memory collisions and voltage and temperature. Fig. 8 depicts the relationship. Here, collisions are always on, and data are sampled at $300s$ after collision activation. With the increase of RAM amount, temperature increases, and voltage reduces. The impact is significant when 12500 or more collisions per μs are occurring as temperature goes beyond $85°$ and voltage drops by 8%. A lower amount of collision is insufficient to cause a large instantaneous change in the FPGA operating condition. However, mounting RAM-Jam changes both temperature and voltage, whose combined effect might result in successful fault injection.

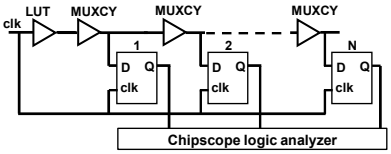

Fig. 9: Time to delay sensor used to measure delay.

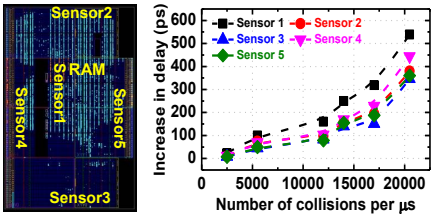

Fig. 10: Delay measured at sensors located at different position of FPGA. Each point is a average value of 10 measurements.

4) Impact on Delay Characteristics: Since memory collisions increase temperature and reduce the internal voltages of the FPGA, the delay characteristic of the FPGA fabric degrades, i.e., propagation delay increases. We implemented TDC sensors to measure the delay variation as described in [18]. Fig. 9 illustrates the basic concept of this sensor. A chain of buffers has been implemented using Xilinx 'MUXCY' primitives. Xilinx EDA tools provide a propagation delay of $23ps$ for a 'MUXCY' primitive in Artix-7 FPGA. Latches are connected with the buffers to measure how far the clk can propagate through this delay line while they are enabled by the same clk. The number of latches passing a single clock pulse is considered as propagation depth. LUTs introduce an initial delay. We consider $100MHz$ clk for this delay line and use a chipscope logic analyzer to observe delay depth.

Fig. 10 shows the location of TDC sensors in FPGA. We put four sensors at each edge of the FPGA and one sensor near XADC. We measured the delay depth of each TDC sensor before and after the activation of memory collision. Delay difference caused by memory collision is equal to the depth difference multiplied by the propagation delay of a 'MUXCY' primitive. The measured delay differences are shown in Fig. 10. The delay increases exponentially at each TDC sensor. Particularly, $Sensor1$ placed near XADC shows a significant increase in delay. At 22500 collision per μs, we see an approximate delay of $570ps$ at $Sensor1$. Delay increment at $Sensor3$ rises to $380ps$ and lowest among 5 sensors. $Sensor3$ is located at the bottom end of the FPGA and very far from the RAMs experiencing memory collisions. Thus, it shows that memory collisions are capable of degrading timing performances, even if a logical or physical barrier separates victim applications. The sensors 2, 3, and 4 also experience delay degradation. The delay degradation at all

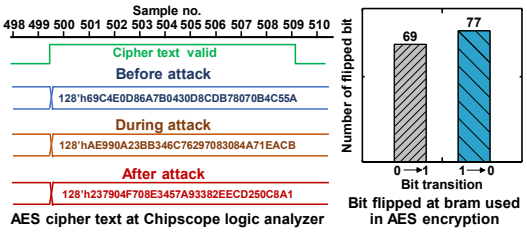

Fig. 11: Faulty ciphertext at AES and flipped bits at BRAM due to the memory collisions.

sensors is more notable at 12500 or more collisions per μs.

5) Impact on Bitstream Configuration: Since memory collisions lead to a significant voltage and temperature degradation, we verify the potential bit-flip occurrence in bitstream/binary configuration placed in the FPGA. In this regard, we implemented an AES-128 IP [19] into FPGA along with memory collision module. AES module occupies 10 instances of BRAM and 882 LUTs. The implemented AES IP runs at $200MHz$. The ciphertext of the AES-128 IP has been observed in the chipscope logic analyzer. The amount of memory collision used for this experiment is 20500 collisions per μs. Fig. 11 presents the AES ciphertext observed before, during, and after memory collisions. The ciphertext changes during memory collisions, and it changes again when collisions are deactivated. *This result indicates that memory collisions are not only causing timing fault but permanently changing the bitstream configuration*. To find the exact location of flipped bits, we use the bitstream readback option to obtain bitstreams before, during, and after memory collisions. In addition, we reverse engineered the bitstreams of Artix-7 FPGA to identify the LUT and BRAM locations.

The number of bits used for Artix-7 FPGA configuration is 30590880. The comparison of readback bitstreams shows that 146 bit flipped during memory collisions. The number and position of flipped bits remain the same after collision deactivation. The positions of these 146 bits are further examined. We found that none of these flipped bits are from LUT configurations. In contrast, they indicate the location of BRAMs used by S-Box of AES-128 IP. Fig. 11 shows the number of bits transitioned from logic '0' to '1' and '1' to '0'. These permanent changes in BRAMs are the reason behind the faulty ciphertext generated after the deactivation of memory collisions. To the best of our knowledge, this is the first remote attack on FPGAs that leads to bit-flips in BRAM.

The nature of bit-flips and its exploitation in practical applications are subject to future works. There have been several works [2] on extracting the secret keys by analyzing faulty ciphertexts. Since memory collisions are capable of generating faulty ciphertexts, this fault injection method can be used to extract keys as shown by [2].

C. Results of Fault Injections in FSM

We intend to utilize memory collisions to inject timing faults in the FSM. For the sake of demonstration, a simple password authentication FSM has been implemented as described in Sec. III-A. The state machines are represented by 3 bit registers. Six states are defined as: *Reset* (000), *Wait for password* (010) *Compare* (100), *Log in successful* (110), *Log in failed* (111), and *Do operation* (101). User password data is 32 bit and is compared with a 32 bit password stored in the system. If any timing violation induced by memory collisions allows bypass of *Compare* and provides access to *Log in successful* or *Do operation*, fault injection attempt is considered as successful. The system is designed to operate at $600MHz$ frequency. We apply a wrong password to FSM repeatedly which shows *Login failed* state. Next, we enable memory collisions and toggle it at $1MHz$ frequency. We count the number of rounds required to obtain a successful fault.

TABLE I: Statistics on fault injection attempts in FSM.

# Collisions per μs	Maximum temperature	# Rounds ($1\mu s$) for successful fault injection		
		Minimum	Mean	Maximum
12000	60 °C	4700	6000	9960
17000	67°C	4	315	667
20500	70°C	3	108	510

Table I presents the statistics on fault injection attempts for three different number of collisions: 12000, 17000, and 20500 per μs. We find that fault injection is highly successful for 20500 and 17000 collisions. The required number of rounds for these two scenarios are between 3 and 667. Only a few numbers of successful faults have been observed for 12000 collisions per μs. Although this experiment focused on a simple FSM, a similar control logic is present in a lot of digital designs. Thus, successful fault injection in this FSM shows that RAM-Jam also can affect other large digital circuits.

D. Result of Fault Injections in Neural Network

To assess the effectiveness of RAM-Jam, we implement an NN, more specifically, a CNN trained on the MNIST dataset for handwritten digit recognition [20], a widely accepted benchmark in the computer vision community. We chose this implementation as it is further optimized for FPGAs, namely, it supports fixed-point arithmetic, and a low silicon footprint (see [21] for more details). At first, we train CNN to detect $0-10$ digits with the support of Python Tensorflow and Keras CNN library. Then, the trained CNN is converted to a Verilog HDL representation for FPGA implementation. The implementation consists of convolution block, decision function, RAM, etc. We find the decision function (responsible for classification) has '12' states which are vulnerable to timing violations. This training and FPGA implementation can be easily extended for the detection of other objects using different CNN structures. The FPGA implementation occupied 2645 (4% of total) LUTs, 10 Block RAMs, and 18 DSP hardware primitives. The implementation is designed to operate at $500MHz$ frequency. We place memory collision and trained CNN module on the FPGA. They are physically separated by a barrier of 3 slices.

To the best of our knowledge, RAM-Jam is the first hardware demonstration of remote fault injection in NNs. As explained in III-B, a successful fault injection occurs

TABLE II: Statistics on fault injection in a digit-detector CNN.

# Collisions per μs	Maximum temperature	# Rounds ($1\mu s$) for successful fault injection		
		Minimum	Mean	Maximum
12000	60 °C	4000	5000	7000
17000	67°C	1	300	797
20500	70°C	1	100	324

when the NN outputs wrong classification result. For instance, we provide images of digit '7' to the CNN and verify the classification output. If another digit is shown at the output, we consider that fault injection is successful. Table II shows the statistics on fault injection attempts with the minimum, maximum, and mean attempts required for successful fault injection. Here, three different memory collision scenarios with 12000, 17000, and 20500 collisions per μs have been considered. Statistics are based on a hundred attempts made for each case. We generate repeated memory collision pulses of $1\mu s$ duration and count the number of rounds required for successful fault injection. Successful fault injections are observed for all memory collision scenarios. Specifically, for 20500 number of collisions, fault injection happens very frequently from the first round to 324 rounds. A similar observation is also made for 17000 collisions per μs case. The number of rounds required for successful fault injection is much higher in the case of 12000 collisions and only a few faults were observed. The minimum number of rounds is 4000. Since this scenario causes a delay drop of $150ps$, its impact is lower than in other cases. A maximum temperature of 70°C is shown by XADC sensor during injecting faults. As fault injection is possible with different amounts of memory collisions, a malicious user can choose the number of collisions based on the available resources in a shared FPGA environment.

V. Potential Countermeasures

Since the hardware Trojan proposed in this work can be inserted remotely into FPGAs, the bitstream can be checked prior to the configuration for the Trojan detection [5]. However, many legitimate applications utilize dual port RAMs to improve the performance of their circuit. Unfortunately, one cannot extract the intended access pattern to RAMs by analyzing the bitstream, since the user can set them during runtime. Therefore, in contrast to including RO- and TDC-based hardware Trojans [2], initializing dual port RAMs in the bitstream does not automatically reveal any malicious patterns that could be detected. In this case, upon initialization of any dual port RAM, only a warning can be issued about the possibility of a hardware Trojan.

As a conventional countermeasure, applications can also be designed as more resilient to fault attacks. For instance, by changing the encoding of FSMs and having larger hamming distances between states, one can prevent a sudden transition to a critical state caused by a timing fault. Moreover, by detecting the *don't care* states, one can mitigate possible transitions to the so-called locked states [11]. Countermeasures relying on redundancy can make a circuit more resilient against faults too. However, since in our attack the chip experiences a global voltage drop that affects the entire chip, such countermeasures should be considered more carefully.

VI. Conclusion

In this paper, we presented a novel fault injection mechanism on FPGAs using memory collisions. We demonstrated that by generating a sufficient number of write collisions in dual port RAMs, an adversary can create severe voltage drops and excessive heat on the chip. This vulnerability can be exploited as a hardware Trojan to cause timing violations and bit-flips. The hardware Trojan can be implanted remotely and activated without having any physical access to the chip. To show the effectiveness of our proposed fault injection mechanism, we mounted an attack against an authentication scheme. Furthermore, we presented the first remote fault attack against a NN engine. Finally, we discussed countermeasures.

References

[1] S. Trimberger *et al.*, "Security of FPGAs in data centers," in *2nd Intrl. Verification and Security Wksp*. IEEE, 2017, pp. 117–122.

[2] J. Krautter *et al.*, "Fpgahammer: Remote voltage fault attacks on shared FPGAs, suitable for DFA on AES," *IACR Trans. on Cryptographic Hardware and Embedded Systems*, pp. 44–68, 2018.

[3] F. Schellenberg *et al.*, "An inside job: Remote power analysis attacks on FPGAs," in *2018 Design, Automation & Test in Europe Conf. & Exhibition*. IEEE, 2018, pp. 1111–1116.

[4] M. Zhao *et al.*, "FPGA-based remote power side-channel attacks," in *Symp. on Security and Privacy*. IEEE, 2018, pp. 229–244.

[5] D. R. Gnad *et al.*, "Checking for electrical level security threats in bitstreams for multi-tenant FPGAs."

[6] Xilinx, "7 Series FPGAs Configuration User Guide," 2016.

[7] R. K.-H. Lo *et al.*, "Dual port SRAM read-disturb-write mechanism and design for test," in *Joint Intrl. Symp. on e-Manufacturing and Design Collaboration & Semiconductor Manufacturing*. IEEE, 2017, pp. 1–4.

[8] F. Li *et al.*, "Power modeling and characteristics of field programmable gate arrays," *Trans. on Comp.-Aided Design of Integrated Circuits and Systems*, vol. 24, no. 11, pp. 1712–1724, 2005.

[9] M. M. Alam *et al.*, "Recycled FPGA detection using exhaustive lut path delay characterization," in *Intrl. Test Conf.* IEEE, 2016, pp. 1–10.

[10] S. Birla *et al.*, "Static noise margin analysis of various SRAM topologies," *Intrl. J. of Engineering and Tech.*, vol. 3, no. 3, p. 304, 2011.

[11] A. Nahiyan *et al.*, "Avfsm: a framework for identifying and mitigating vulnerabilities in fsms," in *Proc. of the 53rd Annual Design Automation Conf.* ACM, 2016, p. 89.

[12] B. Sunar *et al.*, "Sequential circuit design for embedded cryptographic applications resilient to adversarial faults," *Trans. on Computers*, vol. 57, no. 1, pp. 126–138, 2008.

[13] G. Lacey *et al.*, "Deep learning on FPGAs: Past, present, and future," *arXiv preprint:1602.04283*, 2016.

[14] K. Grosse *et al.*, "Adversarial initialization–when your network performs the way i want," *arXiv preprint:1902.03020*, 2019.

[15] Y. Liu *et al.*, "Fault injection attack on deep neural network," in *IEEE/ACM Intr. Conf. on Comp.-Aided Design*, 2017, pp. 131–138.

[16] J. Breier *et al.*, "Deeplaser: Practical fault attack on deep neural networks," *arXiv preprint: 1806.05859*, 2018.

[17] M. Alam *et al.*, "Enhancing fault tolerance of neural networks for security-critical applications," *arXiv preprint:1902.04560*, 2019.

[18] D. R. Gnad *et al.*, "Voltage drop-based fault attacks on FPGAs using valid bitstreams," in *27th Intrl. Conf. on Field Programmable Logic and Applications*. IEEE, 2017, pp. 1–7.

[19] "Opencores," 2019. [Online]. Available: https://opencores.org/projects/tiny_aes

[20] Y. LeCun *et al.*, "Gradient-based learning applied to document recognition," *Proc. of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[21] "Verilog generator of neural net digit detector for FPGA," 2018. [Online]. Available: https://github.com/ZFTurbo/Verilog-Generator-of-Neural-Net-Digit-Detector-for-FPGA