A Comprehensive Survey on Non-Invasive Fault Injection Attacks

Amit Mazumder Shuvo, *Graduate Student Member, IEEE,* Tao Zhang, *Graduate Student Member, IEEE,* Farimah Farahmandi, *Member, IEEE,* and Mark Tehranipoor, *Fellow, IEEE*

***Abstract*—Non-invasive fault injection attacks have emerged as significant threats to a spectrum of microelectronic systems ranging from commodity devices to high-end customized processors. Unlike their invasive counterparts, these attacks are more affordable and can exploit system vulnerabilities without altering the hardware physically. Furthermore, certain non-invasive fault injection strategies allow for remote vulnerability exploitation without the requirement of physical proximity. However, existing studies lack extensive investigation into these attacks across diverse target platforms, threat models, emerging attack strategies, assessment frameworks, and mitigation approaches. In this paper, we provide a comprehensive overview of contemporary research on non-invasive fault injection attacks. Our objective is to consolidate and scrutinize the various techniques, methodologies, target systems susceptible to the attacks, and existing mitigation mechanisms advanced by the research community. Besides, we categorize attack strategies based on several aspects, present a detailed comparison among the categories, and highlight research challenges with future direction. By underlining and discussing the landscape of cutting-edge, non-invasive fault injection, we hope more researchers, designers, and security professionals examine the attacks further and take such threats into consideration while developing effective countermeasures.**

***Index Terms*—Non-invasive attacks, fault injection, threat model, security, assessment, target system, mitigation techniques.**

I. Introduction

With the increasing reliance on computing systems for critical applications, ensuring their security and reliability has become an urgent and paramount concern. Among the emerging threats in hardware security, fault injection attacks have garnered significant attention. These attacks can introduce faults or errors into system-level security-critical operations, intending to compromise their confidentiality, integrity, and/or availability [1], [2]. Fault injection attacks have been carried out using invasive or semi-invasive methods, where adversaries have to physically modify or manipulate the hardware or software components of the targeted system through sample preparation (e.g., chip depackaging) in most laser and micro-probing fault injections [3]–[6]. Such invasive or semi-invasive techniques, albeit sometimes extremely effective, are often cumbersome, calling for costly setup with direct access to the system. Besides, the physical damage/altering renders perceptible footprints, making these invasive solutions less promising options when stealthy intrusions are prioritized [7]–[9]. In contrast, non-invasive fault injection techniques can eliminate mandatory physical modification by exploiting inherent vulnerabilities of system operations through clock [10], voltage [11], or electromagnetic (EM) [12] glitches, enabling security compromise more covertly without sacrificing effectiveness. On top of that, quite a few emerging non-invasive techniques are proposed and demonstrated to cause disturbance on target platforms *remotely* through run-time overclocking [13] and undervolting [14], [15] within a device.

Amit Mazumder Shuvo, Tao Zhang, Farimah Farahmandi, and Mark Tehranipoor are with the Department of Electrical and Computer Engineering, University of Florida (email: amazumdershuvo@ufl.edu, tao.zhang@ufl.edu, farimah@ece.ufl.edu, and tehranipoor@ece.ufl.edu)

Although non-invasive fault injection attacks manifest excellent capabilities and the potential to subvert the security of crucial infrastructure, the majority of academia and leading industry, unfortunately, may not raise sufficient awareness on such issues. The rapid development of cutting-edge attacks and countermeasures in the landscape of non-invasive fault injection research also motivates us to present a comprehensive survey to close the gap in the following aspects.

- A wide range of target systems vulnerable to non-invasive fault injection attacks are investigated in the literature. These systems include Internet of Things (IoT) devices, cryptographic circuits, microcontrollers, microprocessors, FPGAs, and cloud computing infrastructures [16]–[20]. However, there is a lack of comprehensive research on characterizing the impacts of different microelectronic platforms on the effectiveness of particular fault injection attack strategies.
- There are emerging non-invasive attack techniques that appear to be more threatening, e.g., being stealthier, more effective, and easier to launch than their conventional counterparts, with an extensive set of security threats and threat models [13]–[15], [21]. These novel attacks call for dedicated introduction, investigation, and discussions to understand their working principles for countermeasure development,
- Countermeasures have also been proposed against fault injection adversaries, including approaches such as *fault detection* and *fault mitigation*. Fault detection aims to catch the induced errors, while fault mitigation can minimize the negative impacts of injected faults (e.g., hardware redundancy, software-based/algorithm-based mitigation, etc.) [22]–[27]. Unfortunately, extant surveys did not review, organize, and discuss these countermeasures systematically based on their benefits, limitations, and the types of attacks they are designed to counter.
- In addition to countermeasures, quite a few pre-silicon fault injection assessment frameworks have been developed to evaluate the vulnerability of a hardware design prior to tape-out using modern electronic design automation (EDA) tools [28], [29]. Their applications for assessing the viability of fault injection attack strategies remain relatively unexplored and require more elaboration.

Although existing literature portrays some surveys on fault injection attacks concentrating on different implementation platforms and attack strategies, most of them lack state-of-the-art non-invasive fault injection attacks [30]. Furthermore, recent surveys overlook exploring these techniques within the context of threat models, fault injection methodologies, assessment frameworks, and countermeasures [31]. Consequently, the existing fault injection attack classifications presented in the literature lack a comprehensive taxonomy specifically addressing the non-invasive techniques while considering the abovementioned criteria.

To address these limitations, we investigate and analyze the state-of-the-art non-invasive fault injection techniques, steps and strategies of the attacks, associated threats with target devices, assessment frameworks to evaluate the vulnerability, and countermeasures to thwart the attacks. We also categorize both the attack strategies and countermeasures based on our comprehensive investigation. To the best of our knowledge, this is the first attempt to explore the differences and similarities among several non-invasive fault injection techniques, providing helpful insights on effective defenses against emerging threats and inspiring further research in this field. Our main contributions in this paper are summarized as follows.

- We explore the target devices of the attacker, attack vectors, and the related threat models in detail.
- We propose a taxonomy of the non-invasive fault injection attacks by investigating various variants, such as physical and non-physical attacks on hardware and software-based platforms.
- We discuss simulation-based assessment frameworks of the non-invasive attacks and their objectives. In addition, we discuss existing mitigation strategies and countermeasures against non-invasive fault injection attacks.
- We identify open research challenges and future directions to enhance the resilience of systems against non-invasive fault injection attacks.

The rest of this paper is organized as follows. Section II provides objectives and characteristics of non-invasive fault injection attacks with their differences from semi-invasive and invasive attacks, Section III describes potential threats and target devices of these attacks, Section IV provides a detailed taxonomy of non-invasive fault injection attacks, Section V describes assessment frameworks, Section VI discusses mitigation techniques to counter these attacks, Section VII provides open research challenges with future research direction and Section VIII concludes the survey.

II. Overview of Non-invasive Fault Injection Attacks

A. Objective

A non-invasive fault injection attack aims to compromise a targeted device's confidentiality, integrity, and/or availability by leveraging the implementation weakness. By manipulating the system's functionality through controlled injection of faults, an adversary seeks to steal cryptographic keys, private information, configuration bits, device's firmware, modify restricted memory contents, and potentially gain unauthorized access or control. In general, the adversary discovers the potentially vulnerable locations by analyzing a design's hardware and injects faults with precise control over timing and locations using non-invasive techniques.

B. Characteristics

A non-invasive fault injection attack requires minimal cost to set up the target device since one is assumed to intercept the device's communication traffic or connect it to a test circuit for in-depth analysis of system functionality and timing. Once correctly discovered, fault injection attacks can be easily reproduced at a negligible cost. Additionally, these attacks leave no detectable traces, making them a formidable threat to the security of various devices [1]. Moreover, these attacks are *active* in nature, which means run-time manipulation of signals directed toward the device and malicious modification on the benign system behaviors, rendering static countermeasures less useful. Therefore, detecting such attacks on specific devices often demands substantial time and effort, involving activities like reverse engineering through software disassembly or comprehending intricate hardware designs. This underscores the need for an in-depth understanding of the chip architecture and associated software for devising effective non-invasive fault injection attacks.

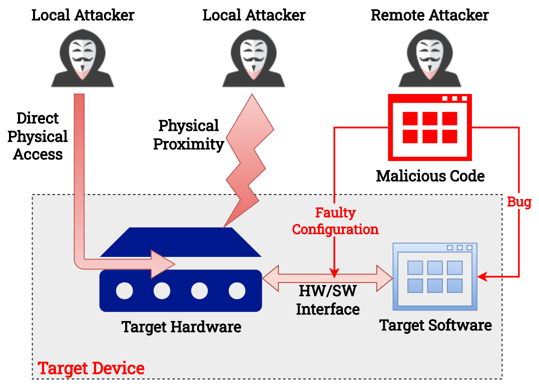


Fig. 1: Various non-invasive fault injection techniques

Note that most of these attacks are *physical attacks* since they require direct physical access or proximity to the target devices. However, an unprivileged adversary can leverage software or hardware-software interface to conduct specific non-invasive fault injection attacks remotely without requiring physical access. Figure 1 illustrates a very high-level overview of these non-invasive fault techniques with both local and remote attackers. We present a detailed discussion on several non-invasive fault injection techniques afterward (see Section IV).

C. Difference From Invasive and Semi-Invasive Fault Injection Attacks

Besides non-invasive fault injection, there are two other important categories in this landscape, i.e., *invasive* and *semi-invasive* attacks. We compare and contrast them with non-invasive attacks as follows, with a summary of the comparison illustrated in Table I.

Invasive Fault Injection Attacks: Invasive fault injection attacks mandate direct entry into a device's internal components after removing and destroying chip packaging [32]. Typically, these attacks can be very effective as they can directly access and read the internal signals physically. Meanwhile, they demand costly setups equipped with micro-probing and focused ion beam (FIB) facilities [33], [34]. It is also expertise-intensive to maneuver the high-end equipment precisely for these attacks. Nonetheless, initiating such attacks necessitates a minimal preliminary understanding of the functional behaviors of both the chip and software, often involving consistent methods across various products [1].

Semi-invasive Fault Injection Attacks: There exist significant contrasts between non-invasive and invasive attacks, giving potential for semi-invasive attacks that share characteristics of both [35]. Although both semi-invasive and invasive attacks need to process the chip packaging (e.g., removal), semi-invasive approaches maintain the chip's passivation layer intact without establishing connections with internal lines [1]. As such, like non-invasive ones, semi-invasive techniques can be more affordable and easier to reproduce than invasive solutions by using physical measures such as laser or optical illuminations [36], [37].

TABLE I: List of Differences Among Non-invasive, Semi-invasive and Invasive Fault Injection Attacks

	Non-invasive Attacks	**Semi-invasive Attacks**	**Invasive Attacks**
Cost of attacks	Low	Medium	High
Level of physical tampering required	Low	Medium	High
Level of knowledge required of the target functionality	High	Medium	Low
Type of attacks	Physical and non-physical	Only Physical	Only Physical
Examples of techniques	Clock and voltage glitching, overheating, EM fault, software fault, remote hardware faults, etc.	Laser and optical fault injection attacks	Micro-probing, reverse engineering, IC modification, etc.

III. Threat Models and Target Devices

Several well-known threats introduced by non-invasive fault injection attacks and the associated threat models have been covered in the literature. These attacks are performed in various manners (see Section IV) to induce hardware bit-flips, bit-set-resets, and software faults to compromise system-level security. Diverse target platforms adhere to their specific architectures and operational principles, resulting in a wide range of potential targets for non-invasive fault injection attacks. These targets may include instructions within microcontrollers or the security-sensitive register values in FPGA or ASIC implementations. In essence, this rich diversity of target options is a source of inspiration for skilled adversaries to develop versatile and adaptable attack strategies that can be tailored to exploit the unique characteristics of each platform. Therefore, it is imperative to discuss particular threats related to different target devices. We elaborate on this topic as follows.

A. Microprocessor and Microcontroller-Based Systems

A single or multi-bit fault can threaten the security of a CPU or an embedded processor. Generally, the processor instructions can be classified into three groups according to the specific architectural components in which they are executed, i.e., arithmetic/logical operations, memory writing/reading, and branch instructions [20]. An attacker can introduce a fault during the instruction fetch stage by either altering the opcode or arguments of the instruction that cause the instruction to be substituted by another instruction. In addition, faulting branch instructions can maliciously modify the control flow of programs since their configuration determines branch destinations [20]. As such, an adversary can leverage non-invasive fault injection attacks to effectively compromise both the data and control flow integrity during the run-time microelectronic operations [19]. Consequently, a computational error by executing faulty instructions or modified control flow may leak sensitive information and pose a security threat to the overall system. Moreover, faults induced by non-invasive techniques can result in erroneous data within the memory, causing an unauthorized modification of the sensitive information (e.g., Data integrity violation) [38].

B. FPGA

The primary goal of attacking an FPGA-based system includes corrupting the configuration bitstream or maliciously altering the functionality of the implemented design. An attacker can violate the system's integrity by introducing bit-flip or bit-set reset faults into the configuration bits of a running FPGA [39]–[41]. Similarly, injecting bit-flip faults into the configurable logic blocks (CLBs) may bypass the built-in security mechanism of an implemented design (e.g., crypto modules) or violate data confidentiality [17]. Recent investigations have unveiled hardware-based fault injection techniques applicable to shared cloud FPGAs (multi-tenant model). These tactics revolve around the manipulation and subsequent overloading of the shared power distribution system (PDS), leading to the induction of voltage drooping. By deliberately timing these malicious voltage drops, attackers can induce timing violations that slow down the transmission of deep neural network (DNN) weight packages between off-chip memory and on-chip buffer and result in a faulty package capture at the receiving end (*adversarial weight duplication attack*). As a result, these faulty packages can lead to the duplication of DNN weight parameters, negatively affecting the performance of the DNN [42]. Recent strides have also been made in developing software-based fault injection strategies targeting FPGAs. These strategies highlight the potential for a spatially and logically segregated attacker within one FPGA fabric region to inject faults into a victim tenant in a separate region. By exploiting the creation of supply voltage drops through deliberate malicious switching activity, these tactics manifest as software-initiated fault attacks on multi-tenant FPGAs [43].

C. Hardware Accelerators

A recently developed form of non-invasive GPU fault injection attack, known as the *overdrive attack*, exploits vulnerable software interfaces to the GPU driver [44]. This enables customization of GPU hardware settings, including dynamic voltage and frequency scaling (DVFS). By capitalizing on this capability, researchers have introduced random faults during the execution of GPU kernels, consequently inducing occurrences of silent data corruption (SDC). By exploiting this SDC-triggering fault injection, an investigation involved the injection of faults into a GPU-accelerated kernel running the AES (Advanced Encryption Standard). Remarkably, this approach facilitated the recovery of all encryption keys [45].

Moreover, the Rowhammer attack [46], which enables attackers to corrupt data in adjacent memory locations by iteratively accessing a row within modern DRAM chips, is not limited to solely CPU-based homogeneous systems. The landscape of integrated CPU-GPU heterogeneous systems in mobile system-on-chip (SoC) has given rise to a web-based GPU-accelerated Rowhammer attack on shared memory [47], [48]. These integrated configurations share The memory subsystem between the CPU and GPU. The study demonstrated the GPU's potential to amplify the Rowhammer attack. By exploiting bit flips from Rowhammer, the researchers managed to escape the confines of the Firefox sandbox. This escape enabled them to breach address space layout randomization (ASLR), originally intended to secure arbitrary read and write

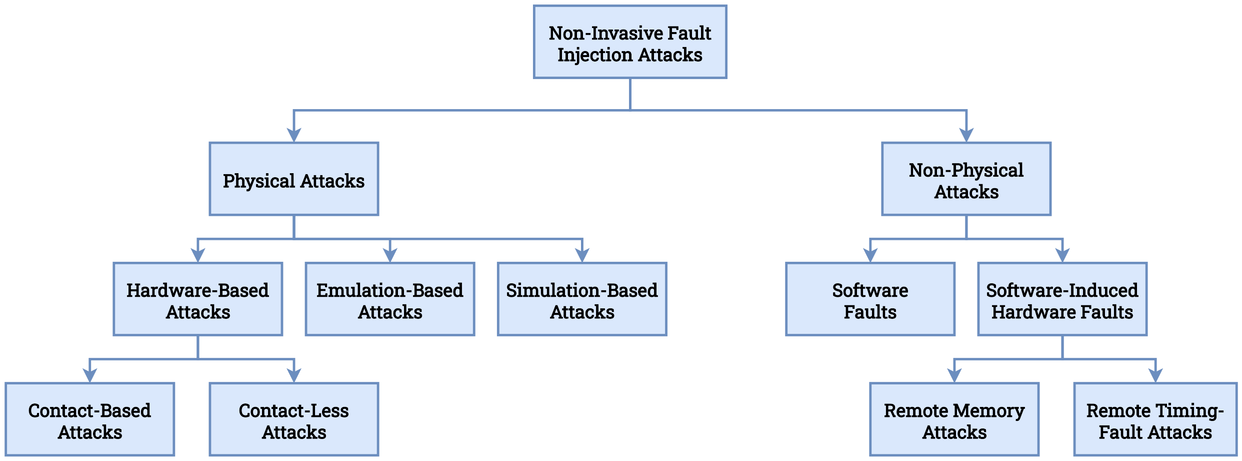


Fig. 2: Taxonomy of non-invasive fault injection attacks.

capabilities of remote code execution. The modern GPU's extensive parallelism and high memory bandwidth introduce a potential vulnerability to attacks like Rowhammer, primarily because these GPUs can generate memory accesses significantly faster than CPUs. Apart from these, modern ML/AI accelerators or other ASIC accelerators are also vulnerable to the aforementioned fault-injection attacks.

D. Application Specific Integrated Circuits (ASIC)

Since a modern SoC consists of a wide range of components, including the functional blocks of intellectual properties (IPs), bus protocols, interconnects, memory blocks, power management units, etc., several non-invasive fault injection attacks targeting these components are also very common. An attacker tries to generate corrupted and/or prematurely encrypted output through non-invasive fault injection techniques on the cryptographic IPs (e.g., AES, RSA, MAC, ECC, etc.). These fault injection techniques can be paired with advanced fault analysis techniques such as differential fault analysis (DFA), differential fault intensity analysis (DFIA), and fault sensitivity analysis (FSA) to extract the secret keys [49]–[51]. In addition, premature results can either undermine the security levels of cryptographic implementations or bypass the security features. Several hardware and software fault injection techniques induce bit-flips into the configuration bits of a functional block within a chip, ultimately compromising the system's security [52], [53].

Various software-driven fault-injection attacks have demonstrated the capability to substantially reduce the supply voltage during software computations executed on a CPU within an integrated CPU-FPGA SoC [54]. Furthermore, instances of Rowhammer attacks have been orchestrated, originating from the FPGA component, resulting in the corruption of the host system's main memory within integrated CPU-FPGA setups [55]. Moreover, distinct implementations and methodologies for exploitation have been devised to take advantage of this vulnerability within native environments. It has led to the attainment of kernel privileges by initiating bit flips on page table entries. Such actions have been carried out in diverse scenarios, encompassing web browsers where the objective has been to defeat the constraints of the JavaScript sandbox. Additionally, these techniques have been extended to virtual machines within cloud computing environments [45].

Each component of modern SoC can be interconnected with others or interfaced with system memory via physical interconnects. Bus protocols with specialized designs have been created and adopted (e.g., advanced peripheral bus (APB), advanced extensible interface (AXI), peripheral component interconnect express (PCIe), etc.) to facilitate reliable data exchange through interconnections among these diverse SoC components [56]–[58]. The hardware constituents associated with these protocol layers are susceptible to various non-invasive fault injection techniques [45]. Recent research also underscores the ability of software-based attacks to circumvent secure communication channels established between ASIC components. In this case, the primary goal of an adversary is to induce bit-flip faults and maliciously alter the desired flow of data transactions. These scenarios may potentially lead to denial of service (DoS) attacks and prevent secure data availability at the desired time.

IV. Taxonomy of Non-invasive Fault Injection Attacks

This section presents an all-encompassing classification of non-invasive fault injection attacks. This classification is structured around a set of characterizations, including the types of threats posed, the specific targets of these attacks, the diverse attack vectors employed, and the fault injection techniques utilized. Figure 2 illustrates the taxonomy of these attacks. Note that any non-invasive fault injection attack can be classified into two major categories: physical and non-physical attacks based on the availability of physical proximity. We elaborate on each category and sub-category with their comparison in the following subsections.

A. Physical Attacks

When a non-invasive fault injection attack requires physical access or physical proximity to the target devices, it is then categorized as a *physical attack*. In such scenarios, the attacker must introduce the fault into the device via accessible pins like clock input and power supply or through mediums such as thermal or electromagnetic radiation. These entry points allow the attacker to affect the device physically from the external environment. Although the level of access and interaction with the target device vary depending on the attacker's skill set and fault-injection setup, this hands-on access can potentially amplify the attack's impact as the attacker gains more control over the injection process.

By leveraging a physical fault injection technique, an adversary can directly target a hardware device running a security-sensitive operation to violate confidentiality or integrity. On the other hand, it is also possible to emulate the fault injection attacks in an emulator or generate a simulation-based attacking environment using commercial EDA tools. These approaches allow an attacker to observe the attack's impact on the emulated hardware or in the simulation result before attacking the original target device. In addition, emulation and/or simulation also allow the reproduction of the attack scenarios with different attack intensities for further research. Therefore, a physical attack can be further classified into *hardware-based attacks*, *emulation-based attacks*, and *simulation-based attacks* as depicted in Figure 2. Note that an emulation/simulation-based attack is a pre-analysis or preparation for in-field attacks instead of an actual attack. It signifies that the principal goal of these attacks is to assess the feasibility and challenges associated with fault injection techniques prior to executing the authentic attacks on a hardware device. Therefore, in an emulation/simulation-based attack, the attacker applies the same fault injection techniques they intend to employ on the physical device during a hardware-based attack.

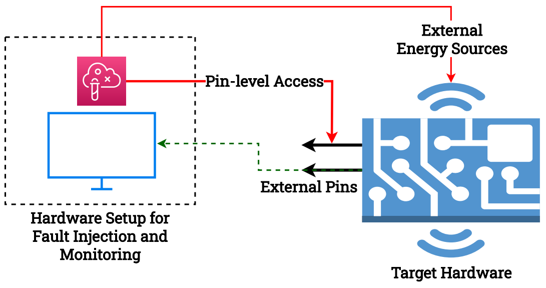
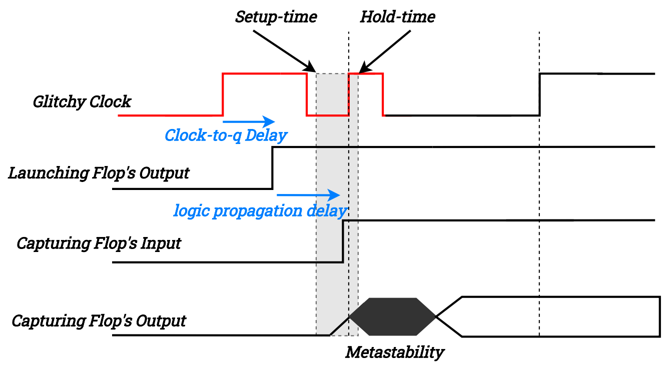


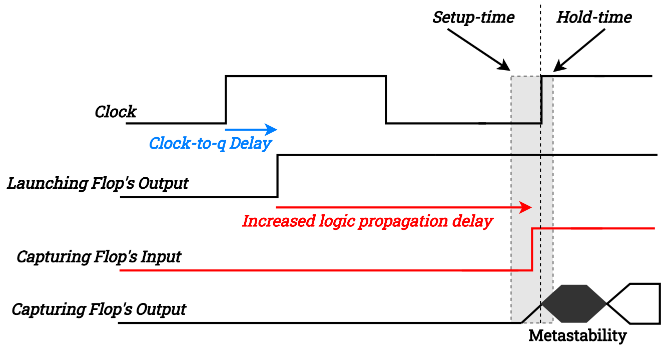
Fig. 3: Hardware-based fault injection approach into target hardware.

1) Hardware-Based Attacks: The approaches used in hardware-based fault injection attacks involve augmenting the targeted system by incorporating specialized hardware components. This augmentation facilitates the deliberate introduction of faults into the system, enabling subsequent observation of their consequences [30]. However, this external hardware setup may be a bit expensive in some cases. Hardware-based fault injection techniques can be executed using either *contact-based* methods, where the external interface of the integrated circuit is disrupted (e.g., using active probes at the pin level or engaging in tampering), or *contact-less* methods, which involve directing external energy resources towards specific design components. Figure 3 illustrates the high-level diagram of a hardware-based fault injection technique, including an external setup for fault injection and attack monitoring. In the latter approach, certain segments of the hardware design are subjected to external influences such as electromagnetic signals or thermal waves, thereby corrupting the functionality of the digital elements within the hardware. We briefly describe different hardware-based attacks as follows.

a) Contact-Based Attacks: Contact-based fault injection methodologies typically utilize pin-level tools to introduce faults. These tools encompass active probes that are directly connected to the pins of integrated circuits, such as processor chips [31]. This technique grants users complete control over the location of fault injection and fault timing, enabling a consistent and continuous injection as required. For instance, an adversary can manipulate the clock pin of a device during a security-critical operation, thereby instigating clock glitches within the system's clock. Consequently, the timing window available for capturing input data diminishes, potentially resulting in a flip-flop violating its setup-time constraint. This violation forces the flip-flop into a metastable state, which in turn can trigger a bit-flip fault at the flip-flop's output (see



(a)



(b)

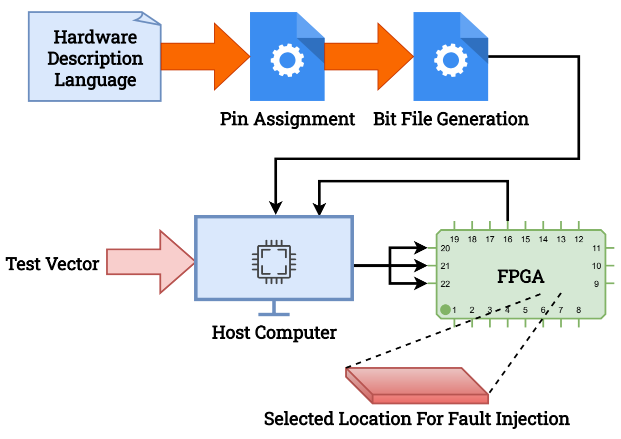
Fig. 4: Setup-time violations cause timing faults to hardware registers by (a) shortening the clock period and (b) decreasing supply voltage.

Figure 4a). Similarly, an attacker can induce drastic voltage glitches on the power supply pins to increase logic propagation delays along the timing paths. Consequently, metastability may arise, resulting in compromised security due to the eventual induction of bit-flip timing faults within the system (see Figure 4b).

b) Contact-Less Attacks: In contrast, the contact-less physical approach to fault injection involves using an external device capable of emitting high-energy radiation or waves, which can disrupt the regular operations of a hardware circuit [31]. These energy emissions encompass electromagnetic radiation and excessive heat. Typically, attackers deploy active probes to generate electromagnetic (EM) radiation, leading to the induction of eddy currents within the loop interconnects (e.g., ring stripes in the on-chip power network) of the running device. Consequently, this process triggers a localized IR drop within the circuit, thereby increasing logic propagation delays and eventually provoking transient faults. Furthermore, overheating also contributes to the increase of logic propagation delays within the targeted hardware, subsequently giving rise to transient faults. Notably, it is crucial to exercise precise control over both the physical proximity and the intensity of the energy source to prevent accidental impairment or even irreversible damage to the target devices.

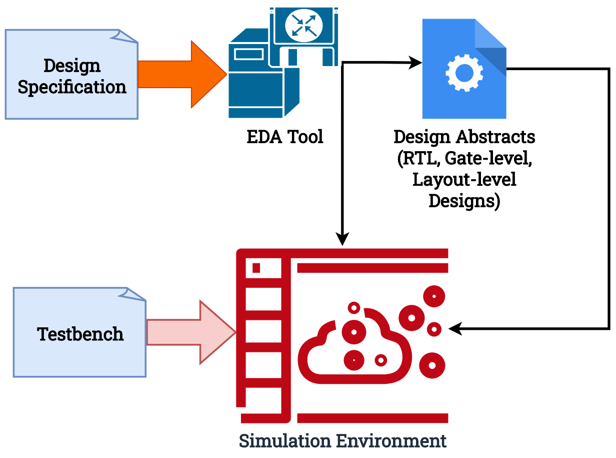
2) Emulation-Based Attacks: Fault emulation techniques combine the performance of hardware-based approaches with the precision of software-based controls. As a result, their implementation encompasses both software and hardware phases [31]. A notable advantage of the emulation-based approach is its unrestricted flexibility in selecting fault locations. Moreover, these techniques eliminate the need for a physical rendition of the component dedicated to fault injection and reliability evaluations.

This approach synthesizes and implements an emulated



Hardware Description Language
Pin Assignment
Bit File Generation
Test Vector
Host Computer
FPGA
Selected Location For Fault Injection

(a)



(b)

Fig. 5: (a) Emulation-based fault injection approach. (b) Simulation-based fault injection approach

version of the target device on programmable hardware, such as FPGAs. The process involves a software-driven workflow that generates a bit file using the source hardware description language (HDL) representation of the target device [31]. Subsequently, this bit file is employed to configure the FPGA. Following this, an attacker adheres to the same flow to designate specific segments of the FPGA for fault injection. The fault injection process involves applying testbenches with faulty test vectors. Note that the master software on a host computer manages the entire process, including FPGA programming and flow coordination. Additionally, the host scrutinizes the consequences of the fault injections, thereby enabling a comprehensive analysis of their effects. The high-level flow of the emulation-based fault injection attack is illustrated in Figure 5a.

It is essential to recognize that while FPGAs are often employed for emulating target hardware, substantial disparities exist between FPGA and ASIC implementations across various dimensions, such as design complexities, design adaptability, power consumption, timing, area, etc. These discrepancies imply that the hardware emulated on an FPGA may not accurately mirror the characteristics of the original target hardware due to differences in the underlying implementation platforms [59], [60]. Consequently, the susceptibility of the target hardware to fault injection may vary depending on where it is instantiated. Hence, a designer must take into account these differences between original and emulated hardware while analyzing the design's susceptibility to emulation-based fault injection attacks.

3) Simulation-Based Attacks: Unlike the emulation-based counterpart, the method of simulation-based fault injection creates a simulation model replicating the system being examined [30]. The process also involves the development of a detailed simulation model of the fault injection techniques employed. The construction of these simulation models is facilitated through commercial EDA tools specifically developed to design and validate hardware components at the pre-silicon stage. It is worth mentioning that a simulation-based fault injection technique is an integral part of the pre-silicon security verification of a chip to reduce the cost of post-silicon validation [61]. Recent research suggests that several assessment frameworks are established that utilize simulation-based frameworks to analyze and quantify a design's susceptibility to potential fault injection attacks [28], [29], [62].

With a simulation-based fault injection framework, faults can be deliberately introduced into diverse design abstractions of the intended target, spanning various pre-silicon levels such as register transfer level (RTL), gate level, or physical layout level. Specially developed testbenches within the simulation environment feed the target design with necessary stimuli to inject and excite the desired fault locations on time. After that, a designer utilizes EDA tools (Xcelium from *Cadence*, Z01X from *Synopsis*, etc.) to simulate the design with the injected faults [63], [64]. The same simulation environment is also harnessed for assessing and analyzing the simulation outcomes. Generally, the EDA tools are deployed on a server accessible via a host computer. This setup facilitates the orchestration of the simulation processes and the subsequent evaluation of results within a controlled and systematic environment. The high-level flow of the simulation-based fault injection attack is illustrated in Figure 5b.

Note that the accuracy of pre-silicon assessment obtained from a simulation-based framework depends on the characterization of fault injection techniques using EDA tools and the differences between the pre-silicon and post-silicon phases of a design concerning design parameters and specifications [65]. These differences are the key reason for the deviation between a simulation-based attack and a hardware-based actual attack. Therefore, a designer must consider the unpredictable factors (e.g., on-chip process variation, temperature, noise, fluctuations in supply voltage, etc.) causing these differences that emerge during chip fabrication or in-field operation [28].

B. Non-Physical Attacks

In cybersecurity, a *non-physical attack* represents a distinctive breed that operates independently of physical access or proximity to the target device. In this context, the primary intent of attackers is to manipulate the software or firmware embedded within the operational device, employing tactics that jeopardize the overall system's confidentiality and integrity. Furthermore, malicious adversaries can exploit this avenue to install untrusted software or firmware, thereby procuring unauthorized access or facilitating the escalation of privileges. An intriguing facet of non-physical attacks is the transition they affect in the threat model. Departing from conventional local attackers who require physical presence, non-physical attacks usher the possibility of remote attackers equipped solely to execute code locally. Attackers can infiltrate device

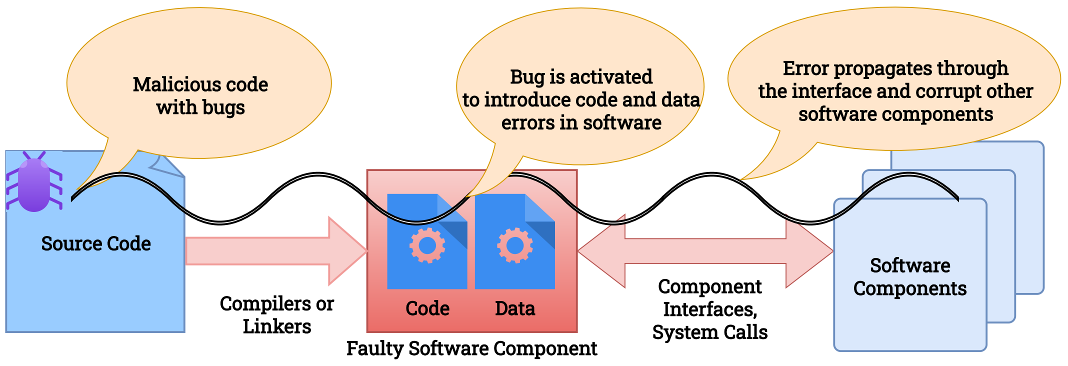

Malicious code with bugs
Bug is activated to introduce code and data errors in software
Error propagates through the interface and corrupt other software components
Source Code
Compilers or Linkers
Code
Data
Faulty Software Component
Component Interfaces, System Calls
Software Components

Fig. 6: Software fault injection and propagation into software components

locations that elude physical access with complex hardware setups by harnessing non-physical fault injection techniques.

While pinpointing fault locations and determining optimal timings in the context of non-physical attacks face challenges, these exploits offer the notable advantage of obviating the need for resource-intensive hardware setup. The execution of non-physical attacks mandates only deploying dedicated malicious software or specialized attack code instead of understanding in-depth hardware details. In terms of objectives and motivations, non-physical attacks can be launched in two different forms, i.e., *software faults* and *software-induced hardware faults*. Software faults focus on exploiting vulnerabilities intrinsic to the software architecture of the target system while *software-induced hardware faults* delve into software-triggered attacks capitalizing on alteration of the functional behavior of hardware circuitry. We discuss these two categories in detail in the following subsections.

1) Software Faults: In general, the software fault injection process can be categorized into three main phases: (1) making a malicious alteration to the source code, (2) activating the bug to impact the targeted software component, and (3) allowing the error to spread through the interface, potentially corrupting other connected software components. Figure 6 illustrates the high-level overview of these three phases. The first phase is initiated by replacing a subset of program instructions with alternative instructions (e.g., code error). These alterations might include instructions with distinct operands or nop (No-operation) commands. Such manipulations are strategically designed based on prevalent software errors found within real microprocessors or embedded systems [66]. These malicious modifications are introduced into the program code either at the source code level or within the binary executable. The selection of code location to inject a fault depends on the intention of the type of fault. For example, to impact variable assignments, faults are placed in code segments involving variable assignments. Alternatively, target locations can be selected based on susceptibility, as indicated by software complexity metrics [67].

The second phase begins with the flow of the faults induced by malicious codes through the compilers and linkers to the target code. To emulate the permanent nature of software errors, an attacker should inject these faults into the target code before its execution begins. However, there is also a chance of synchronously introducing software faults at the execution time of the relevant code. In recent investigations, this constraint has been relaxed, allowing asynchronous injection of code alterations during runtime [66]. By triggering the failures at a specific or random time, this strategy simplifies the implementation and execution of fault injection experiments, especially in scenarios where the system operates in a stable state. Depending on initialization, variable assignment, checking, and function definition, threat models adopted by software faults can be described as follows [68].

- *Faulty Initialization:* The initial value is substituted with a faulty value, or alternatively, the relevant instructions are altered to *nops* in the absence of an initial value.
- *Erroneous Assignment:* The erroneous assignment involves either assigning an incorrect value to the destination with the correct destination remaining unassigned or the assignment itself is omitted (achieved by substituting the relevant instructions with no-operation commands, *nops*).
- *Faulty Condition Checking:* The branch instruction is substituted with a *nop* when a condition check is absent, or alternatively, the condition check is modified incorrectly, deviating from the intended branching.
- *Modified Function:* An original user-defined sequence of instructions is substituted with an alternative user-defined sequence of instructions, assuming that the defective instructions must fit within the spatial confines of the original instructions.

It is important to note that software-based fault injection can be extended to manipulating individual bits, bytes, or words within memory locations or registers of a device (e.g., data error). In certain instances, attackers might endeavor to emulate the impact of hardware malfunctions (e.g., CPU, bus, or memory faults) by introducing disruptions into memory or hardware registers through software interventions [66].

In the final phase, the disruptions of the victim codes have the potential to propagate through software interfaces, resulting in the corruption of other software elements and ultimately leading to operational malfunctions. Moreover, the injection of software errors commonly focuses on software components that furnish a generalized application programming interface (API). Due to unexpected interactions among components and incorrect utilization of the API (e.g., interface error), software faults can emerge within these components, which are not solely developed for specific systems [66], [69]. For instance, an operating system (OS) providing an API for device drivers is exposed to third-party suppliers to incorporate support for new devices and is vulnerable to software faults.

2) Software-Induced Hardware Faults: In addition to non-invasive fault injection attacks targeting software components, it is also likely that a remote attacker possesses privileged software to access the elements within the target hardware components (e.g., functional logic or memory cells) through the utilization of hardware-software interfaces. These specific attacks are recognized as *software-induced hardware faults*.

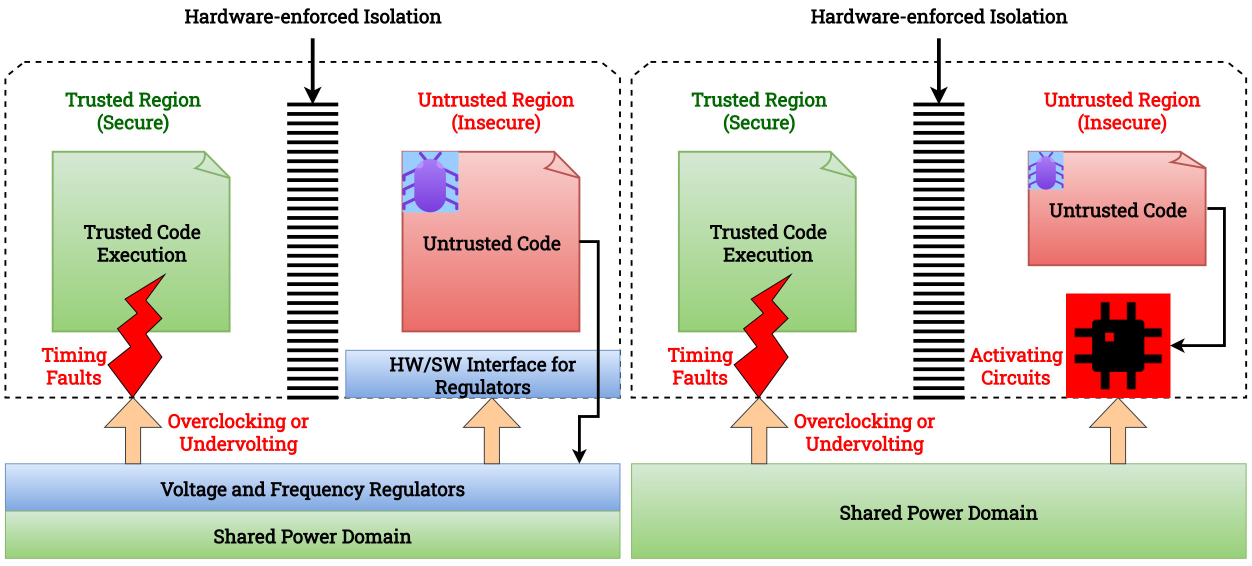

Hardware-enforced Isolation
Trusted Region (Secure)
Untrusted Region (Insecure)
Trusted Code Execution
Untrusted Code
Timing Faults
HW/SW Interface for Regulators
Overclocking or Undervolting
Voltage and Frequency Regulators
Shared Power Domain
(a)
Hardware-enforced Isolation
Trusted Region (Secure)
Untrusted Region (Insecure)
Trusted Code Execution
Untrusted Code
Timing Faults
Activating Circuits
Overclocking or Undervolting
Shared Power Domain
(b)

Fig. 7: Remote timing fault-injection techniques to corrupt a trusted code execution (a) by configuring the HW/SW interface and (b) by activating malicious power-hungry circuits.

It is essential to recognize that, like hardware-based fault injection attacks, the core objective of these attacks is to introduce timing faults into the hardware registers, deliberately distorting functional behavior and violating the system's confidentiality and integrity. Moreover, attackers seek to manipulate memory content through forceful reads and writes to specific memory addresses. According to the victim targets, we classify these attacks into two main categories, i.e., *remote timing-fault attacks* and *remote memory attacks*. We discuss them in detail as follows.

a) Remote Timing-Fault Attacks: Recent investigations indicate that a privileged adversary can exploit the DVFS feature [70] to compromise a hardware device through remote fault injection [13], [14], [21]. Prominent processor manufacturers such as Intel and AMD have developed contemporary processors that furnish privileged software interfaces, allowing for the dynamic regulation of runtime frequency and operational voltage [71]. This dynamic regulation ensures that voltage and frequency are seamlessly adjusted in accordance with the instantaneous current demand. Although modern SoCs compartmentalize hardware components into distinct trusted and untrusted regions, the power management system remains a shared resource across all regions. As a result, it becomes feasible for an untrusted entity to remotely manipulate the frequency and voltage regulators beyond the limits stipulated by the vendor. Sometimes, attackers may remotely activate some malicious circuits from an untrusted region that draw significant power from the shared PDN and reduce the supply voltage. These manipulations can potentially breach the contents of trusted hardware boundaries, which host security-sensitive computations. Generally, these remote hardware attacks manifest as undervolting or overclocking actions, introducing software-induced timing faults within the secure region of the targeted hardware. Figure 7a and Figure 7b depict these two common remote timing-fault injection techniques from an untrusted region and their intrusions on the trusted region.

The literature presents numerous examples of attack scenarios against ARMv7 *TrustZone* [72] and Intel *SGX* [73] by exploiting DVFS features. In these attacks, remote attackers have set their sights on various objectives, such as extracting cryptographic keys, compromising RSA signature chain authentication by introducing self-signed untrusted applications, tampering with memory safety mechanisms (e.g., through incorrect array indexing or flawed dynamic memory allocation), inducing faults in key derivation processes, etc. We present some of the representative instances as follows.

- *CLKSCREW Attack:* In this attack, a malicious kernel driver is developed to attack a code in the ARM TrustZone at a higher privilege than the kernel. This rogue kernel driver improperly configures the frequency regulator, surpassing the vendor's recommended limits, effectively overclocking the CPU, and deliberately introducing timing faults [13]. Throughout the attack, the attacker meticulously manages the fault timing to enable precise injection into the victim thread such that the fault only affects the target program segments as intended, avoiding any undesired corruption of attacking code or non-targeted code.
- *Plundervolt Attack:* This attack allows a highly privileged adversary to access the hardware-software interface of the voltage regulator and decrease the CPU voltage beyond vendor-specified limits during a security-sensitive compilation in the Intel CPU's SGX enclave [14]. Similar to the *CLKSCREW* attack, the adversary lowers the CPU voltage to cause timing faults using malicious software code. In addition, the attacker can fine-tune the undervolting parameters to ensure that all other codes, except for the victim code, remain unaffected during the attack.
- *DeepStrike Attack:* In addition to targeting an ASIC, a remotely orchestrated attack can employ power glitching to compromise a DNN running on a cloud-based FPGA [74]. Initially, an attacker utilizes time-to-digital converter (TDC) sensors to capture distinct patterns of voltage fluctuations associated with the execution of various DNN layers (e.g., side-channel leakage) and infer the precise timing of an attack (*attack scheduler*). Afterward, he injects glitches using a controllable power-hungry circuit (*power striker*) in the shared power distribution network of the cloud-FPGA to induce timing violations

and data loading faults. Consequently, these faults disrupt the FPGA DSP kernel and misclassify the victim DNN application. Note that, without utilizing the DVFS feature, this attack uses programmable logic on the FPGA to implement the power striker and to generate power glitches.

- *DFaulted Attack:* This attack triggers software errors within the CPU due to undervolting attacks guided by an FPGA on a combined FPGA-CPU system. The primary objective of this attack is to activate a power plundering circuit, such as a ring oscillator deployed on the FPGA, to generate power disturbances in the shared power distribution network. As a result, these power disruptions lead to timing faults during the execution of security-sensitive computations within the processing system, ultimately undermining the system's security [75].
- *Lightning Attack:* Apart from targeting general-purpose CPUs, remote adversaries can leverage DVFS faults to launch attacks on GPU accelerators. An example of such an attack is the *Lightning* attack, which strategically targets the sensitive elements of a DNN model using hardware-induced transient faults facilitated by the DVFS feature [76]. To enhance the efficacy of this attack, attackers devise a sensitive target search algorithm to identify vulnerabilities in DNN models and utilize a genetic algorithm to determine the necessary DVFS parameters. Recent investigations on Nvidia GPU demonstrate that the attack substantially diminishes the accuracy of various convolutional neural networks (CNN).

In addition to the previously mentioned attacks, recent research has shown the occurrence of software-induced timing fault-injection attacks on various hardware targets [15], [21], [43]. These attacks can be categorized further depending on the attacker's target, resulting in distinctions between remote timing-fault attacks on CPUs and remote timing-fault attacks on FPGAs.

b) Remote Memory Attacks: Due to the high density of memory cells in modern semiconductor devices, electromagnetic coupling exists between the capacitors of neighboring dynamic random access memory (DRAM) cells. This intercell crosstalk can be exploited to alter the bits stored in a DRAM cell by aggressively accessing adjacent cells [77]–[79]. Recent investigations suggest that a user-level program developed by a remote attacker can repeatedly access a specific memory address, leading to voltage fluctuations along the row of DRAM cells. When there are numerous activations in the same row, they cause the wordline to switch on and off rapidly. These voltage fluctuations along a row's wordline disrupt nearby rows, causing some of their cells to discharge at an accelerated rate. If a cell loses too much charge before being restored to its original state, it experiences a bit-flip fault [46]. It's important to note that the attacker only requires the ability to execute software code on the target device and doesn't need physical access to the DRAM cells to carry out this attack [47]. While more advanced attacks demand a deeper understanding of computer architecture, this approach allows the attacker to exploit the physical characteristics of DRAM to compromise the system's security.

C. Comparison Among Different Non-Invasive Fault Injection Techniques

In this section, we outline the distinctions and similarities observed among the non-invasive fault injection techniques discussed in the preceding sections (see Section IV-A and Section IV-B). A comprehensive overview of this comparative analysis, organized according to various factors, is presented in Table II. In the context of fault injection attacks, it is evident that contact-based physical attacks offer a high level of control over fault timing and the ability to observe the impact of the fault. However, they come with significant drawbacks, including a heightened risk of damaging the target hardware, low attack reproducibility, limited control over fault timing, and high development costs. Contact-less attacks share many of these characteristics with contact-based attacks, with the main distinction being relatively less control over fault timing and location.

In contrast, emulation and simulation-based attacks provide a highly reproducible approach, utilizing programmable or software-guided environments. These methods offer several advantages, including minimal additional hardware costs and a low risk of damaging the target. Simulation-based attacks, in particular, pose no risk of physical damage since they operate on high-level models of actual hardware. They also offer very high levels of observability and controllability in fault injection.

Unlike physical attacks, all non-physical attacks are cost-effective, require no physical access or proximity, and pose minimal risk of target damage, making them attractive alternatives. However, they do have limitations in terms of controlling fault injection and observing the impact of faults compared to physical attacks. It is important to note that software faults are entirely permanent, while all others are primarily transient.

Given the current trends in the semiconductor industry, the costly setups required for physical attacks are becoming less relevant and less attractive to skilled attackers. Consequently, researchers and designers are increasingly focusing on the emulation or simulation of physical attacks for early design analysis. On the other hand, non-physical attacks are gaining attention due to their stealthy, dangerous, and easily reproducible nature. Remote timing-fault attacks, in particular, can impact interconnected processors equipped with DVFS features. Furthermore, developing mitigation techniques against emerging non-physical attacks requires further investigation. As a result, these non-physical attacks are becoming prominent areas of interest for future exploration among researchers, designers, and potential attackers.

V. Assessment Frameworks

Assessing the security of post-silicon devices against potential fault-injection attacks is both challenging and expensive. Researchers propose pre-silicon assessment frameworks for evaluating a design's susceptibility to such threats to address this issue. During the pre-silicon design and verification phase, these frameworks are developed using EDA tools at various abstraction levels (such as RTL, gate level, and layout level). The key reasons for developing and researching these assessment frameworks are briefly described as follows.

1) Since physical hardware is unavailable at the pre-silicon stage, designers rely on simulation-based fault injection attacks for assessments. These early evaluations yield vulnerability metrics that quantitatively express the feasibility of an attack with reduced cost [28], [62], [80]. In contrast, post-silicon assessments of non-invasive fault injection attacks are too late and impractical because identifying bugs at this stage makes it costly to deploy the mitigation techniques [81].
2) Defenses against non-invasive attacks that rely on spatial and temporal redundancy come with additional power, performance, and area (PPA) overheads [22]–[24]. These countermeasures are often implemented at a global scale

TABLE II: A Comprehensive Comparison Among Several Non-invasive Fault Injection Techniques

Factors	**Physical Attacks**				**Non-Physical Attacks**		
	Contact-Based Attacks	**Contact-Less Attacks**	**Emulation-Based Attacks**	**Simulation-Based Attacks**	**Software Fault Attacks**	**Remote Timing-Fault Attacks**	**Remote Memory Attacks**
Physical access needed?	Yes	No	No	No	No	No	No
Physical proximity needed?	No	Yes	No	No	No	No	No
Cost of extra hardware	Medium	High	Low	Very Low	Very Low	Very Low	Very Low
EDA Tool Required	No	No	Yes	Yes	No	No	No
Risk of damaging the target	High	High	Low	No	Low	Low	Low
Actual fault injected?	Yes	Yes	No	No	Yes	Yes	Yes
Attack modelling required?	No	No	Yes	Yes	No	No	No
Reproducibility of attack	Low	Low	Very High	Very High	High	High	High
Complexity of attack	High	High	Medium	Medium	Low	Low	Low
Cost of attack development	High	High	Medium	Medium	Low	Low	Low
Control on fault timing	High	Medium	High	Very High	Medium	Medium	Medium
Control on fault location	Low	Medium	Medium	Very High	Medium	Low	Medium
Observability of impact	High	High	Medium	Very High	Low	Low	Low
Target of attack	Hardware	Hardware	Hardware	Software Model	Software	Hardware	Hardware
Generator of attack	Hardware	Hardware	Hardware	Software	Software	Software	Software
Fault duration	Transient	Transient	Transient	Transient or Permanent	Permanent	Transient	Transient
HW/SW interface required?	No	No	No	No	No	Yes	No
Type of Attacker	Local	Local	Local	Local	Local or Remote	Remote	Remote

without precise identification of the specific vulnerability to fault attacks. However, pre-silicon assessment frameworks can pinpoint vulnerabilities and propose localized countermeasures with minimal PPA overhead [28].

This survey takes these reasons into account and examines the following assessment frameworks that utilize simulation-based methodologies to assess physical non-invasive fault injection attacks.

- *TVVF:* Introducing setup-time violations in hardware registers through non-invasive techniques can seriously threaten a system's confidentiality and integrity. This concern necessitates a framework that offers a metric known as the TVVF (Timing Violation Vulnerability Factor), which assesses how susceptible a hardware structure is to deliberate fault injection attacks that result in setup time violations. To calculate the probabilistic metric TVVF, the gate-level netlist of a hardware design is analyzed using EDA tools. It is important to note that TVVF comprises two distinct components: firstly, the likelihood of introducing a specific fault into the hardware structure, and secondly, the likelihood of this fault propagating to the observable output of the structure [80].
- *AVFSM:* Modern hardware devices utilize Finite State Machines (FSM) to control the flow of the system's operation. However, an adversary can inject faults and force an FSM state to transit to an unauthorized state accessible to him. This scenario can potentially help the adversary violate the system's confidentiality or integrity. To address and evaluate this vulnerability of FSMs to fault-injection attacks, the AVFSM framework is proposed. This framework focuses on detecting susceptibility to fault injection attacks in FSMs arising during the gate-level synthesis process conducted by Electronic Design Automation (EDA) tools. It employs an ATPG (Automatic Test Pattern Generation)-based assessment technique to extract the state transition graph from the gate-level netlist of a design. Finally, this assessment quantifies the vulnerability of each individual state transition and the overall susceptibility of an FSM to fault injection attacks [62].
- *SoFI:* To evaluate a system's susceptibility to fault-injection attacks properly, a set of design-specific security rules must be defined. This set of rules is termed security properties, and violation of each property under an attack

signifies the violation of the system's security. The SoFI framework is developed and proposed to evaluate the susceptibilities of integrated circuits to fault-injection attacks depending on the security properties inherent to a design [82]. Considering these design-specific security properties, SoFI examines a gate-level netlist, identifies security-sensitive locations (e.g., potential targets for an attacker to exploit and compromise security properties), and assesses the feasibility of fault injection in these locations using setup time violations.
- *LDTFI:* Accurately evaluating non-invasive hardware-based timing faults necessitates proper timing analyses and the consideration of signal latency in dynamic timing simulations. Traditional assessment frameworks developed at the RTL or gate-level abstraction do not account for timing variations incurred during the generation of the physical layout. As a solution, a framework, namely LDTFI, is proposed to conduct a layout-aware evaluation of cryptographic modules (e.g., AES) concerning fault-injection attacks paired with DFA. Through post-layout static timing analysis and post-layout timing simulations, LDTFI provides a probabilistic metric indicating the likelihood of a timing fault causing the leakage of AES secret keys [28]. Unlike the previously mentioned approaches, this framework leverages precise timing data from the physical design to assess its vulnerability to timing faults induced by clock glitches.
- *FISHI:* Instead of using a traditional monolithic device, an untrusted entity can potentially insert malicious chiplets into a (SiP) [83]. This scenario can lead to both internal and remote power fault injection attacks, made possible by the inherent opacity of the semiconductor supply chain. Recent research has proposed a framework named FISHI, which aims to incorporate a root-of-trust chiplet within the SiP. This chiplet enables real-time monitoring of power-noise variations at the system level and facilitates near-sensor Machine-learning (ML) inference to detect anomalies caused by attacks [81]. FISHI utilizes a time-to-digital converter (TDC) sensor solely developed to collect power profiles from specific applications, serving as a reference database for regular operation. Subsequently, the framework incorporates a dedicated hardware (ML) engine that measures the disparities between the power fluctuations observed during runtime and the fault-free reference databases, thus aiding in the detection of potential fault-injection attacks.

As previously noted, these simulation-based frameworks are limited to analyzing physical fault injection attacks. Moreover, there is a lack of advanced methodologies for assessing a system's susceptibility to non-physical fault injection attacks initiated through software. While specific tools can emulate or inject software faults into a system, they cannot effectively quantify its vulnerability to software-induced fault injection attacks [67], [84], [85].

VI. Countermeasures Against Non-Invasive Fault Injection Attacks

Scientists and researchers have proposed numerous countermeasures to safeguard against confidentiality and integrity breaches resulting from fault injection attacks. However, it is essential to note that not all of these mitigation approaches are suitable for addressing non-invasive attacks. It implies that some of these approaches solely focus on semi-invasive or invasive attacks. Traditional countermeasures primarily involve detecting fault injection or abnormalities within the targeted circuit. These countermeasures aim to monitor and assess the outcomes of intrusion or error detection, subsequently rectifying any faulty outputs as needed. According to their implementations, these countermeasures can be broadly classified into two major groups: Hardware-level countermeasures and Software-level countermeasures. The following subsections will provide concise descriptions of these countermeasures.

A. Hardware-Level Countermeasures

As their name implies, these countermeasures are integrated as hardware elements to identify or mitigate the impact of non-invasive fault injection attacks. The primary objective of these countermeasures is to protect a system against attacks on hardware components. This survey presents specific instances of hardware-level countermeasures as outlined below.

- *Standard Cell Hardening:* A combinational or sequential cell can be hardened against fault injection attacks by implementing redundant (e.g., duplication or triplication) logic [86]–[89]. The main objective of these techniques is to detect faults by comparing the original and redundant output. After fault detection, dedicated error correction techniques must be applied to nullify the impact of the fault. However, these techniques suffer from significant PPA overheads due to the additional circuitry for fault detection and error correction [87], [90].
- *Time Redundancy:* This countermeasure is applied to cryptographic circuits, where it involves performing an initial encryption followed by a redundant encryption [22]. Its primary purpose is to enable the simultaneous detection of errors when encountering DFA, achieved through repeated rounds and comparing the original and redundant results. This technique can be implemented at either the RTL or gate level but introduces performance overhead due to the redundancy it entails. Furthermore, it's worth noting that the biased fault model resulting from DFIA can potentially compromise the effectiveness of this countermeasure, as it can inject faults into both the original and redundant components with a high probability of success.
- *Clock-Check Block (CCB):* To mitigate the risks associated with fault sensitivity analysis, one can utilize a clock-check block as a potent countermeasure. This block can identify any undesirable deviations in the clock signal and adjust the cipher accordingly, as demonstrated in [24]. This approach comes with a drawback, namely, the need for additional logic circuits, which increases area utilization and resource overhead.
- *Configurable Delay Block (CDB):* The primary goals of CDB include identifying setup timing violations and eliminating the relationship between fault sensitivity and confidential data to thwart DFA and FSA attacks [23]. It is achieved by utilizing a sequence of configurable delay elements, such as inverters, to examine irregularities in the clock signal or any increments in critical path delays. Although this approach remains applicable at both RTL and gate levels, it suffers from area and resource overheads.
- *Security-Aware FSM:* A strategy can be employed to address and mitigate the susceptibility of FSMs (Finite State Machines) to timing faults arising from using EDA tools during synthesis. This approach considers vulnerabilities during the early stages of design. Nevertheless, it relies on RTL and gate-level abstractions for FSM analysis and countermeasure implementation without directly considering physical-level vulnerabilities. Furthermore,

to implement this countermeasure, additional circuitry is necessary to manage state transitions, ensuring the transition from an authorized state to a protected state, as ensured by security-aware encoding [91].

- *Disabling HW/SW Interface:* When the interface between hardware regulators and software drivers is deactivated, it becomes impossible for an unauthorized entity to manipulate the operating voltage and frequency maliciously. Recent research illustrates that Intel employs this method to safeguard against potential exploitation of the DVFS feature. However, while effective in its specific context, this approach is considered ad-hoc and does not address the fundamental causes of remote timing-fault attacks. Additionally, it does not protect against non-invasive physical attacks, as discussed in [14].
- *Isolated Power Domain:* Hardware components located in both trusted and untrusted regions can be functionally isolated by utilizing distinct power domains. This countermeasure is implemented to thwart remote attackers from exploiting a shared power domain. Nonetheless, this approach demands the management of separate regulators, which is intricate and costly [13].
- *Limit-Checking For Voltage and Frequency:* Additional limit-verification circuitry can be introduced to enforce strict voltage or frequency regulation boundaries. These rigid limits safeguard against remote attackers, confining their actions within the vendor's predefined thresholds. However, this approach necessitates early decisions during the hardware design stage, and manufacturing process variations can alter the operational limits of the regulators. Moreover, imposing specific limits can limit the regulator's adaptability across various devices [13].
- *Layout-Aware Path Adjustments:* Instead of adding supplementary hardware elements, innovative layout-aware countermeasures are suggested to hinder attackers from taking advantage of the consequences of fault injection attacks. A physical design engineer can manipulate the timing path within the fan-in cone of security-critical registers by altering the physical layout's components (such as resizing logic gates, incorporating pairs of inverters as buffers, adjusting placement and routing, etc.) [28]. These modifications introduce multiple faults and induce undesired changes in functional behavior, rendering an attacker's controlled fault injection attack uncontrollable. It is worth noting that these countermeasures result in minimal power, performance, and area overhead.
- *Sensor-Based Fault Detection:* The presence of non-invasive fault injection attempts can be identified by utilizing digital sensors seamlessly integrated into the functional logic of a device. Recent research highlights the efficacy of sensors employing an FTC (Faults-to-Time Conversion) mechanism, which relies on a dual line of buffer chains constructed with HVT (High-Threshold Voltage) and LVT (Low-Threshold Voltage) cells [92], [93]. This sensor is designed to detect alterations in logic propagation delays within the buffer chains, which occur when a fault injection attack is initiated. It is important to emphasize that this sensor is effective in detecting not only non-invasive but also invasive or semi-invasive fault injection attempts. However, it comes with the drawback of increased power consumption and more significant physical space requirements, primarily due to the additional hardware components necessary to post-process sensor outputs.

B. Software-Level Countermeasures

These countermeasures are incorporated either within the software or at the algorithmic level, primarily aimed at addressing software-related vulnerabilities. Some notable examples of software-level countermeasures are further elaborated on as follows.

- *Redundant Computation:* Countermeasures against fault injection attacks can be implemented in the software execution of a design by utilizing methods like recalculation, redundant computation, instruction duplication, and eventually error correction, as outlined in [94]. Nevertheless, this approach is not a practical solution for ASIC hardware and is only viable for implementations based on microprocessors or microcontrollers. Incorporating redundancy and recalculations introduces a notable design overhead, which is a significant concern associated with this approach.
- *Fault-Resistant Crypto Algorithm:* It has been a good practice to adjust a cryptographic algorithm to enhance its resistance against fault-injection attacks. Nonetheless, these fault-tolerant approaches are limited in scope, as they are primarily suitable for cryptographic applications. In addition, they are susceptible to statistical attacks and come with a notable drawback in terms of performance, as they introduce significant overhead [14].
- *Randomization of Instruction Execution:* Incorporating randomization, achieved through inserting nop loops into the runtime execution of the target code, can effectively thwart remote attackers from achieving precise timing control for an attack. Nevertheless, this countermeasure has limitations, particularly regarding safeguarding against attacks relying on runtime profiling for precise timing synchronization [13].
- *Application and Compiler Hardening:* Specific standard library functions can be strengthened by incorporating validation checks for arithmetic operations and corresponding reciprocal operations (e.g., multiplication followed by a division). Recent research indicates that this fortification method has been integrated into the software development kit of the Intel SGX enclave to counter remote fault injection attacks [14]. Moreover, incorporating checksum integrity verification during code compilation and running sensitive code multiple times can offer software-level safeguards against remote fault injection attacks, as mentioned in [13]. However, these methods come at the cost of performance overhead, primarily due to the added validation steps, and they can have a detrimental effect on energy efficiency.
- *Rowhammer Pattern Detection:* Recent studies suggest that several defense techniques have been created to safeguard DRAM against Rowhammer attacks. Software-based protections commonly concentrate on detecting the Rowhammer pattern using the PMU (Performance Monitor Unit) to identify anomalies in cache activities as indicative of a Rowhammer attack [78]. When a high cache miss rate is identified, further analysis of the memory access pattern is conducted to ascertain the presence of a Rowhammer attack [79]. Nonetheless, this software-based protection mechanism is intricate and challenging to implement.

VII. Challenges and Future Research

Despite extensive research and investigations into the feasibility of non-invasive fault injection attacks, the successful execution of such attacks remains a formidable challenge.

Firstly, acquiring a post-silicon hardware device for physical attacks is often financially expensive. Even when a target device is accessible, the attacker requires an expensive setup and specialized equipment to gain physical access or proximity to the target devices. For example, tampering with clock ports or power supply voltages and generating clock or voltage glitches necessitates additional hardware circuitry. In addition, local fault injection techniques like electromagnetic (EM) faults or overheating demand the precise generation of electromagnetic or thermal energy, with meticulous control over proximity (e.g., distance from the target device) and localization (e.g., specific target location within a device). As a result, researchers frequently opt for pre-silicon assessments instead of evaluating susceptibility to such attacks on post-silicon devices. They endeavor to model various fault injection techniques and characterize attack parameters (e.g., fault types, injection timing, injection location, etc.) to assess non-invasive attack vulnerabilities during the pre-silicon phase. However, our survey finds the inherent complexity and challenges associated with developing assessment frameworks at the pre-silicon stage. While commercial sign-off tools for ASIC design can effectively profile non-invasive fault injection attack parameters, their accuracy may be compromised by unpredictable post-silicon device factors such as circuit noise, environmental changes, on-chip process variations, etc [28]. Hence, these frameworks must be improved or re-designed by considering these factors.

Secondly, executing software-based non-physical attacks presents its own set of significant challenges and limitations. Injecting software faults into locations inaccessible to software is practically impossible. Even when specific locations are partially accessible, controllability and observability limitations hinder in-depth analysis of the impact of fault injection attacks [30]. Furthermore, if access to the source code executing the attack is restricted, injecting software-induced faults into the system becomes infeasible. Beyond these constraints, conducting remote software-based attacks on hardware devices introduces unique challenges compared to traditional physical attacks. The attacker must exercise precise control over the timing of fault injection and uphold high timing accuracy throughout the attack. Typically, preventing unintended alterations in non-targeted code within a noisy and intricate target device is challenging [13]. Nevertheless, ensuring that only the victim codes are affected without impacting other code segments during the attack is crucial. Additionally, controlling regulator operating limits and maintaining parameters like undervolting or overclocking involves heuristic and iterative methodologies.

Our comprehensive survey underscores a notable lack of assessments concerning the feasibility of software-based remote fault injection attacks. Furthermore, like 2D Systems-on-Chip (SoCs), non-invasive attacks pose a significant security threat in 2.5D or 3D heterogeneous System-in-Package (SiPs) [83]. Therefore, researchers in the field of hardware security must concentrate on developing assessment frameworks and methodologies to evaluate the potential for fault injection attacks on SiPs.

VIII. Conclusion

In conclusion, this paper thoroughly examines various non-invasive fault injection techniques, encompassing both hardware-based and software-based platforms. Our comprehensive survey systematically categorized these techniques according to threat models, target devices, and attackers' objectives. Furthermore, we delve into assessing attack vulnerabilities on target devices, utilizing EDA tool-based frameworks, and exploring the deployment of countermeasures to mitigate these threats. In summary, the insights and findings presented in this survey serve as a valuable resource and pave the way for further advancements in this domain by providing a deep understanding of non-invasive fault injection techniques.

References

[1] M. Tehranipoor and C. Wang, *Introduction to hardware security and trust*. Springer Science & Business Media, 2011.

[2] S. Bhunia and M. M. Tehranipoor, *Hardware security: a hands-on learning approach*. Morgan Kaufmann, 2018.

[3] S. Tajik, H. Lohrke, F. Ganji, J.-P. Seifert, and C. Boit, "Laser fault attack on physically unclonable functions," in *2015 workshop on fault diagnosis and tolerance in cryptography (FDTC)*. IEEE, 2015, pp. 85–96.

[4] C. Helfmeier, D. Nedospasov, C. Tarnovsky, J. S. Krissler, C. Boit, and J.-P. Seifert, "Breaking and entering through the silicon," in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, 2013, pp. 733–744.

[5] C. Boit, C. Helfmeier, and U. Kerst, "Security risks posed by modern ic debug and diagnosis tools," in *2013 Workshop on Fault Diagnosis and Tolerance in Cryptography*. IEEE, 2013, pp. 3–11.

[6] M. T. Rahman, M. S. Rahman, H. Wang, S. Tajik, W. Khalil, F. Farahmandi, D. Forte, N. Asadizanjani, and M. Tehranipoor, "Defense-in-depth: A recipe for logic locking to prevail," *Integration*, vol. 72, pp. 39–57, 2020.

[7] J.-M. Dutertre, V. Beroulle, P. Candelier, S. De Castro, L.-B. Faber, M.-L. Flottes, P. Gendrier, D. Hély, R. Leveugle, P. Maistri *et al.*, "Laser fault injection at the cmos 28 nm technology node: an analysis of the fault model," in *2018 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*. IEEE, 2018, pp. 1–6.

[8] J. G. Van Woudenberg, M. F. Witteman, and F. Menarini, "Practical optical fault injection on secure microcontrollers," in *2011 Workshop on Fault Diagnosis and Tolerance in Cryptography*. IEEE, 2011, pp. 91–99.

[9] H. Wang, Q. Shi, D. Forte, and M. M. Tehranipoor, "Probing assessment framework and evaluation of antiprobing solutions," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 6, pp. 1239–1252, 2019.

[10] M. Agoyan, J.-M. Dutertre, D. Naccache, B. Robisson, and A. Tria, "When clocks fail: On critical paths and clock faults," in *International conference on smart card research and advanced applications*. Springer, 2010, pp. 182–193.

[11] L. Zussa, J.-M. Dutertre, J. Clediere, and A. Tria, "Power supply glitch induced faults on fpga: An in-depth analysis of the injection mechanism," in *2013 IEEE 19th International On-Line Testing Symposium (IOLTS)*. IEEE, 2013, pp. 110–115.

[12] M. Dumont, M. Lisart, and P. Maurine, "Electromagnetic fault injection: How faults occur," in *2019 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*. IEEE, 2019, pp. 9–16.

[13] A. Tang, S. Sethumadhavan, and S. J. Stolfo, "Clkscrew: Exposing the perils of security-oblivious energy management." in *USENIX Security Symposium*, vol. 2, 2017, pp. 1057–1074.

[14] K. Murdock, D. Oswald, F. D. Garcia, J. Van Bulck, D. Gruss, and F. Piessens, "Plundervolt: Software-based fault injection attacks against intel sgx," in *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2020, pp. 1466–1482.

[15] P. Qiu, D. Wang, Y. Lyu, and G. Qu, "Voltjockey: Breaching trustzone by software-controlled voltage manipulation over multi-core frequencies," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 195–209.

[16] T. Bonny and Q. Nasir, "Clock glitch fault injection attack on an fpga-based non-autonomous chaotic oscillator," *Nonlinear Dynamics*, vol. 96, no. 3, pp. 2087–2101, 2019.

[17] G. Canivet, P. Maistri, R. Leveugle, J. Clédière, F. Valette, and M. Renaudin, "Glitch and laser fault attacks onto a secure aes implementation on a sram-based fpga," *Journal of cryptology*, vol. 24, no. 2, pp. 247–268, 2011.

[18] C. O'Flynn, "Fault injection using crowbars on embedded systems." *IACR Cryptol. ePrint Arch.*, vol. 2016, p. 810, 2016.

[19] J. Balasch, B. Gierlichs, and I. Verbauwhede, "An in-depth and black-box characterization of the effects of clock glitches on 8-bit mcus," in *2011 Workshop on Fault Diagnosis and Tolerance in Cryptography*. IEEE, 2011, pp. 105–114.

[20] A. Barenghi, G. M. Bertoni, L. Breveglieri, M. Pellicioli, and G. Pelosi, "Injection technologies for fault attacks on microprocessors," *Fault Analysis in Cryptography*, pp. 275–293, 2012.

[21] Z. Chen, G. Vasilakis, K. Murdock, E. Dean, D. Oswald, and F. D. Garcia, "{VoltPillager}: Hardware-based fault injection attacks against intel {SGX} enclaves using the {SVID} voltage scaling interface," in *30th USENIX Security Symposium (USENIX Security 21)*, 2021, pp. 699–716.

[22] S. Patranabis, A. Chakraborty, P. H. Nguyen, and D. Mukhopadhyay, "A biased fault attack on the time redundancy countermeasure for aes," in *International workshop on constructive side-channel analysis and secure design*. Springer, 2015, pp. 189–203.

[23] S. Endo, Y. Li, N. Homma, K. Sakiyama, K. Ohta, D. Fujimoto, M. Nagata, T. Katashita, J.-L. Danger, and T. Aoki, "A silicon-level countermeasure against fault sensitivity analysis and its evaluation," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 23, no. 8, pp. 1429–1438, 2014.

[24] J. Zhang, N. Wu, F. Ge, F. Zhou, and X. Zhang, "Countermeasure against fault sensitivity analysis based on clock check block," *IEICE Electronics Express*, vol. 15, no. 11, pp. 20 180 433–20 180 433, 2018.

[25] B. Gierlichs, J.-M. Schmidt, and M. Tunstall, "Infective computation and dummy rounds: Fault protection for block ciphers without check-before-output," in *Progress in Cryptology–LATINCRYPT 2012: 2nd International Conference on Cryptology and Information Security in Latin America, Santiago, Chile, October 7-10, 2012. Proceedings 2*. Springer, 2012, pp. 305–321.

[26] H. Bar-El, H. Choukri, D. Naccache, M. Tunstall, and C. Whelan, "The sorcerer's apprentice guide to fault attacks," *Proceedings of the IEEE*, vol. 94, no. 2, pp. 370–382, 2006.

[27] M. H. Tehranipoor, N. Pundir, N. Vashistha, and F. Farahmandi, *Hardware Security Primitives*. Springer, 2023.

[28] A. Mazumder Shuvo, N. Pundir, J. Park, F. Farahmandi, and M. Tehranipoor, "LDTFI: Layout-aware timing fault-injection attack assessment against differential fault analysis," in *IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE, 2022.

[29] N. Pundir, H. Li, L. Lin, N. Chang, F. Farahmandi, and M. Tehranipoor, "SPILL:security properties and machine-learning assisted pre-silicon laser fault injection assessment," in *ISTFA 2022*. ASM International, 2022, pp. 225–236.

[30] H. Ziade, R. A. Ayoubi, R. Velazco *et al.*, "A survey on fault injection techniques," *Int. Arab J. Inf. Technol.*, vol. 1, no. 2, pp. 171–186, 2004.

[31] M. Eslami, B. Ghavami, M. Raji, and A. Mahani, "A survey on fault injection methods of digital integrated circuits," *Integration*, vol. 71, pp. 154–163, 2020.

[32] N. Asadizanjani, M. T. Rahman, and M. Tehranipoor, "Physical assurance," *Cham Switzerland: Springer Nature Switzerland AG*, 2021.

[33] Q. Shi, N. Asadizanjani, D. Forte, and M. M. Tehranipoor, "A layout-driven framework to assess vulnerability of ics to microprobing attacks," in *2016 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. IEEE, 2016, pp. 155–160.

[34] F. Farahmandi, M. R. Muttaki, H. M. Kamali, and M. Tehranipoor, "PALLET: Protecting analog devices using a last-level edit technique," in *IEEE International Conference on Physical Assurance and Inspection of Electronics (PAINE)*. IEEE, 2023.

[35] S. P. Skorobogatov, "Semi-invasive attacks–a new approach to hardware security analysis," University of Cambridge, Computer Laboratory, Tech. Rep., 2005.

[36] S. P. Skorobogatov and R. J. Anderson, "Optical fault induction attacks," in *Cryptographic Hardware and Embedded Systems-CHES 2002: 4th International Workshop Redwood Shores, CA, USA, August 13–15, 2002 Revised Papers 4*. Springer, 2003, pp. 2–12.

[37] M. Tehranipoor, N. Nalla Anandakumar, and F. Farahmandi, "Laser fault injection attack (fia)," in *Hardware Security Training, Hands-on!* Springer, 2023, pp. 235–257.

[38] G. Sivathanu, C. P. Wright, and E. Zadok, "Ensuring data integrity in storage: Techniques and applications," in *Proceedings of the 2005 ACM workshop on Storage security and survivability*, 2005, pp. 26–36.

[39] A. Duncan, F. Rahman, A. Lukefahr, F. Farahmandi, and M. Tehranipoor, "Fpga bitstream security: a day in the life," in *2019 IEEE International Test Conference (ITC)*. IEEE, 2019, pp. 1–10.

[40] D. R. Gnad, F. Oboril, and M. B. Tahoori, "Voltage drop-based fault attacks on fpgas using valid bitstreams," in *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2017, pp. 1–7.

[41] G. L. Nazar and L. Carro, "Fast single-fpga fault injection platform," in *2012 IEEE international symposium on defect and fault tolerance in VLSI and nanotechnology systems (DFT)*. IEEE, 2012, pp. 152–157.

[42] A. S. Rakin, Y. Luo, X. Xu, and D. Fan, "{Deep-Dup}: An adversarial weight duplication attack framework to crush deep neural network in {Multi-Tenant}{FPGA}," in *30th USENIX Security Symposium (USENIX Security 21)*, 2021, pp. 1919–1936.

[43] J. Krautter, D. R. Gnad, and M. B. Tahoori, "Fpgahammer: Remote voltage fault attacks on shared fpgas, suitable for dfa on aes," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 44–68, 2018.

[44] M. Sabbagh, Y. Fei, and D. Kaeli, "A novel gpu overdrive fault attack," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2020, pp. 1–6.

[45] H. Naghibijouybari, E. M. Koruyeh, and N. Abu-Ghazaleh, "Microarchitectural attacks in heterogeneous systems: A survey," *ACM Computing Surveys*, vol. 55, no. 7, pp. 1–40, 2022.

[46] O. Mutlu and J. S. Kim, "Rowhammer: A retrospective," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 8, pp. 1555–1571, 2019.

[47] Y. Kim, R. Daly, J. Kim, C. Fallin, J. H. Lee, D. Lee, C. Wilkerson, K. Lai, and O. Mutlu, "Flipping bits in memory without accessing them: An experimental study of dram disturbance errors," *ACM SIGARCH Computer Architecture News*, vol. 42, no. 3, pp. 361–372, 2014.

[48] P. Frigo, C. Giuffrida, H. Bos, and K. Razavi, "Grand pwning unit: Accelerating microarchitectural attacks with the gpu," in *2018 IEEE Symposium on Security and Privacy (sp)*. IEEE, 2018, pp. 195–210.

[49] M. Joye and M. Tunstall, *Fault analysis in cryptography*. Springer, 2012, vol. 147.

[50] N. F. Ghalaty, B. Yuce, M. Taha, and P. Schaumont, "Differential fault intensity analysis," in *2014 Workshop on Fault Diagnosis and Tolerance in Cryptography*. IEEE, 2014, pp. 49–58.

[51] Y. Li, K. Sakiyama, S. Gomisawa, T. Fukunaga, J. Takahashi, and K. Ohta, "Fault sensitivity analysis," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2010, pp. 320–334.

[52] J. Breier and X. Hou, "How practical are fault injection attacks, really?" *IEEE Access*, vol. 10, pp. 113 122–113 130, 2022.

[53] K. M. Abdellatif and O. Hériveaux, "Silicontoaster: A cheap and programmable em injector for extracting secrets," in *2020 Workshop on Fault Detection and Tolerance in Cryptography (FDTC)*. IEEE, 2020, pp. 35–40.

[54] D. G. Mahmoud, S. Hussein, V. Lenders, and M. Stojilović, "Fpga-to-cpu undervolting attacks," in *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2022, pp. 999–1004.

[55] Z. Weissman, T. Tiemann, D. Moghimi, E. Custodio, T. Eisenbarth, and B. Sunar, "Jackhammer: Efficient rowhammer on heterogeneous fpga-cpu platforms," *arXiv preprint arXiv:1912.11523*, 2019.

[56] "AMBA APB Protocol Specification," [Online], https://developer.arm.com/documentation/ihi0024/latest/, Accessed on 11/06/2023.

[57] "Learn the architecture - An introduction to AMBA AXI," [Online], https://developer.arm.com/documentation/102202/0300/AXI-protocol-overview, Accessed on 11/06/2023.

[58] "PCI Express* Architecture," [Online], https://www.intel.com/content/www/us/en/io/pci-express/pci-express-architecture-devnet-resources.html, Accessed on 11/06/2023.

[59] I. Kuon and J. Rose, *Quantifying and exploring the gap between FPGAs and ASICs*. Springer Science & Business Media, 2010.

[60] S. Y. Neyaz, I. Saxena, N. Alam, and S. A. Rahman, "Fpga and asic implementation and comparison of multipliers," in *2020 International Symposium on Devices, Circuits and Systems (ISDCS)*. IEEE, 2020, pp. 1–4.

[61] S. Dey, J. Park, N. Pundir, D. Saha, A. M. Shuvo, D. Mehta, N. Asadi, F. Rahman, F. Farahmandi, and M. Tehranipoor, "Secure physical design," *Cryptology ePrint Archive*, 2022.

[62] A. Nahiyan, K. Xiao, K. Yang, Y. Jin, D. Forte, and M. Tehranipoor, "AVFSM: A framework for identifying and mitigating vulnerabilities in fsms," in *2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC)*. IEEE, 2016, pp. 1–6.

[63] "Xcelium Logic Simulator — Cadence," [Online], https://www.cadence.com/en_US/home/tools/system-design-and-verification/simulation-and-testbench-verification/xcelium-simulator.html, Accessed on 11/06/2023.

[64] "Z01X Functional Safety Assurance," [Online], https://www.synopsys.com/verification/simulation/z01x-functional-safety.html, Accessed on 11/06/2023.

[65] L. Wu, G. Ribera, N. Beringuier-Boher, and S. Picek, "A fast characterization method for semi-invasive fault injection attacks," in *Cryptographers' Track at the RSA Conference*. Springer, 2020, pp. 146–170.

[66] R. Natella, D. Cotroneo, and H. S. Madeira, "Assessing dependability with software fault injection: A survey," *ACM Computing Surveys (CSUR)*, vol. 48, no. 3, pp. 1–55, 2016.

[67] H. Madeira, D. Costa, and M. Vieira, "On the emulation of software faults by software fault injection," in *Proceeding International Conference on Dependable Systems and Networks. DSN 2000*. IEEE, 2000, pp. 417–426.

[68] W.-I. Kao, R. K. Iyer, and D. Tang, "Fine: A fault injection and monitoring environment for tracing the unix system behavior under faults," *IEEE Transactions on Software Engineering*, vol. 19, no. 11, pp. 1105–1118, 1993.

[69] A. Johansson and N. Suri, "Error propagation profiling of operating systems," in *2005 International Conference on Dependable Systems and Networks (DSN'05)*. IEEE, 2005, pp. 86–95.

[70] E. Le Sueur and G. Heiser, "Dynamic voltage and frequency scaling: The laws of diminishing returns," in *Proceedings of the 2010 international conference on Power aware computing and systems*, 2010, pp. 1–8.

[71] R. Hebbar and A. Milenković, "Pmu-events-driven dvfs techniques for improving energy efficiency of modern processors," *ACM Transactions on Modeling and Performance Evaluation of Computing Systems*, vol. 7, no. 1, pp. 1–31, 2022.

[72] "TrustZone for Cortex-M – Arm®," [Online], https://www.arm.com/technologies/trustzone-for-cortex-m, Accessed on 11/06/2023.

[73] "Intel® Software Guard Extensions," [Online], https://www.intel.com/content/www/us/en/developer/tools/software-guard-extensions/overview.html, Accessed on 11/06/2023.

[74] Y. Luo, C. Gongye, Y. Fei, and X. Xu, "Deepstrike: Remotely-guided fault injection attacks on dnn accelerator in cloud-fpga," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2021, pp. 295–300.

[75] D. G. Mahmoud, D. Dervishi, S. Hussein, V. Lenders, and M. Stojilović, "Dfaulted: Analyzing and exploiting cpu software faults caused by fpga-driven undervolting attacks," *IEEE Access*, vol. 10, pp. 134 199–134 216, 2022.

[76] R. sun, P. Qiu, Y. Lyu, J. Dong, H. Wang, D. Wang, and G. Qu, "Lightning: Leveraging DVFS-induced transient fault injection to attack deep learning accelerator of gpus," *ACM Transactions on Design Automation of Electronic Systems*.

[77] R. Qiao and M. Seaborn, "A new approach for rowhammer attacks," in *2016 IEEE international symposium on hardware oriented security and trust (HOST)*. IEEE, 2016, pp. 161–166.

[78] Y. Jiang, H. Zhu, H. Shan, X. Guo, X. Zhang, and Y. Jin, "Trrscope: Understanding target row refresh mechanism for modern ddr protection," in *2021 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. IEEE, 2021, pp. 239–247.

[79] Z. B. Aweke, S. F. Yitbarek, R. Qiao, R. Das, M. Hicks, Y. Oren, and T. Austin, "Anvil: Software-based protection against next-generation rowhammer attacks," *ACM SIGPLAN Notices*, vol. 51, no. 4, pp. 743–755, 2016.

[80] B. Yuce, N. F. Ghalaty, and P. Schaumont, "TVVF: Estimating the vulnerability of hardware cryptosystems against timing violation attacks," in *2015 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. IEEE, 2015, pp. 72–77.

[81] T. Zhang, M. L. Rahman, H. M. Kamali, K. Z. Azar, M. Tehranipoor, and F. Farahmandi, "Fishi: Fault injection detection in secure heterogeneous integration via power noise variation," in *2023 IEEE 73rd Electronic Components and Technology Conference (ECTC)*. IEEE, 2023, pp. 2188–2195.

[82] H. Wang, H. Li, F. Rahman, M. M. Tehranipoor, and F. Farahmandi, "SoFI: Security property-driven vulnerability assessments of ics against fault-injection attacks," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2021.

[83] N. Vashistha, M. L. Rahman, M. S. U. Haque, A. Uddin, M. S. U. I. Sami, A. M. Shuvo, P. Calzada, F. Farahmandi, N. Asadizanjani, F. Rahman *et al.*, "Toshi-towards secure heterogeneous integration: Security risks, threat assessment, and assurance," *Cryptology ePrint Archive*, 2022.

[84] J. A. Duraes and H. S. Madeira, "Emulation of software faults: A field data study and a practical approach," *Ieee transactions on software engineering*, vol. 32, no. 11, pp. 849–867, 2006.

[85] J. Aidemark, J. Vinter, P. Folkesson, and J. Karlsson, "Goofi: Generic object-oriented fault injection tool," in *2001 International Conference on Dependable Systems and Networks*. IEEE, 2001, pp. 83–88.

[86] Y. Aguiar, F. Wrobel, S. Guagliardo, J.-L. Autran, P. Leroux, F. Saigné, A. Touboul, and V. Pouget, "Radiation hardening efficiency of gate sizing and transistor stacking based on standard cells," *Microelectronics Reliability*, vol. 100, p. 113457, 2019.

[87] J. Teifel, "Self-voting dual-modular-redundancy circuits for single-event-transient mitigation," *IEEE Transactions on Nuclear Science*, vol. 55, no. 6, pp. 3435–3439, 2008.

[88] Y. Li, A. Breitenreiter, M. Andjelkovic, J. Chen, M. Babic, and M. Krstic, "Double cell upsets mitigation through triple modular redundancy," *Microelectronics Journal*, vol. 96, p. 104683, 2020.

[89] V. Petrovic and M. Krstic, "Design flow for radhard tmr flip-flops," in *2015 IEEE 18th International Symposium on Design and Diagnostics of Electronic Circuits & Systems*. IEEE, 2015, pp. 203–208.

[90] H. Zhengfeng and L. Huaguo, "A novel radiation hardened by design latch," *Journal of Semiconductors*, vol. 30, no. 3, p. 035007, 2009.

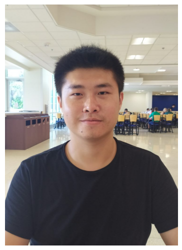
[91] A. Nahiyan, F. Farahmandi, P. Mishra, D. Forte, and M. Tehranipoor, "Security-aware fsm design flow for identifying and mitigating vulnerabilities to fault attacks," *IEEE Transactions on Computer-aided design of integrated circuits and systems*, vol. 38, no. 6, pp. 1003–1016, 2018.

[92] M. R. Muttaki, T. Zhang, M. Tehranipoor, and F. Farahmandi, "FTC: A universal sensor for fault injection attack detection," in *2022 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. IEEE, 2022, pp. 117–120.

[93] M. R. Muttaki, B. T. Barker, M. Tehranipoor, and F. Farahmandi, "FTC—A universal low-overhead fault injection attack detection solution," in *ISTFA 2022*. ASM International, 2022, pp. 386–391.

[94] A. Barenghi, L. Breveglieri, I. Koren, G. Pelosi, and F. Regazzoni, "Low-cost software countermeasures against fault attacks: implementation and performances trade offs," in *Proceedings of 5th Workshop on Embedded Systems Security-WESS*. Citeseer, 2010.

AMIT MAZUMDER SHUVO received his BSc in Electrical and Electronics Engineering from Bangladesh University of Engineering and Technology in 2017. He is currently a 3rd year Ph.D. student and a graduate research assistant at the Florida Institute for Cybersecurity Research (FICS) within the Electrical and Computer Engineering (ECE) Department at the University of Florida. His research focuses on fault injection attack assessment, property-driven security assurance, tamper detection, and secure heterogeneous integration.



TAO ZHANG received the B.S. and M.S. degrees from Northwest University and University of Electronic Science and Technology of China, in 2016 and 2019, respectively. He is currently a Ph.D. candidate at the Department of Electrical and Computer Engineering, University of Florida. His research focuses on side-channel security, FPGA security, and heterogeneous integration security. He published 10+ peer-reviewed publications in premier venues, including the Design Automation Conference (DAC), IEEE Electronic Components and Technology Conference (ECTC), and the European Test Symposium (ETS). He is a recipient of the DAC Young Fellowship in 2020 and 2021 and serves as a reviewer of multiple renowned IEEE/ACM journals and conferences.

FARIMAH FARAHMANDI received the B.S. and M.S. degrees from the Department of Electrical and Computer Engineering, University of Tehran, Iran, in 2010 and 2013, respectively, and the Ph.D. degree from the Department of Computer and Information Science and Engineering, University of Florida, in 2018. She is an Assistant Professor at the Department of Electrical and Computer Engineering, University of Florida. Her research has been sponsored by SRC, AFRL, DARPA, and Cisco. Her research interests include design automation of System-on-Chips and energy-efficient systems, formal verification, hardware security validation, and post-silicon validation and debug. Her research has resulted in two books, seven book chapters, and several publications in premier ACM/IEEE journals and conferences, including IEEE Transactions on Computers, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Design Automation Conference (DAC), and Design Automation and Test in Europe (DATE). She is a member of ACM. Her research has been recognized by several awards, including IEEE System Validation and Debug Technology Committee Student Research Award, the Gartner Group Info-Tech Scholarship, a nomination for the Best Paper Award in ASPDAC 2017, and the DAC Richard Newton Young Student Fellowship. She is currently serving as the Founding Director for the Florida Institute for Cybersecurity Research (FICS). She has served on many technical program committees as well as organizing committees of premier ACM and IEEE conferences.

MARK TEHRANIPOOR is currently the Intel Charles E. Young Preeminence Endowed Chair Professor of cybersecurity with the University of Florida, where he is currently serving as the Chair for the Department of Electrical and Computer Engineering (ECE). His current research interests include hardware security and trust, supply chain security, IoT security, VLSI design, and test and reliability. He is a fellow of ACM, a Golden Core Member of IEEE CS, and a member of ACM SIGDA. He was a recipient of a dozen of the Best Paper Awards and nominations, as well as the 2008 IEEE Computer Society (CS) Meritorious Service Award, the 2012 IEEE CS Outstanding Contribution, the 2009 NSF CAREER Award, and the 2014 AFOSR MURI Award. He received the 2020 the University of Florida Innovation of the year as well as the Teacher/Scholar of the Year Awards. He co-founded the IEEE International Symposium on Hardware-Oriented Security and Trust (HOST) and IEEE International Conference on Physical Assurance and Inspection of Electronics (PAINE). He serves on the program committee of more than a dozen leading conferences and workshops. He has also served as the Program and General Chair for a number of IEEE and ACM-sponsored conferences and workshops (HOST, ITC, DFT, D3T, DBT, NATW, and more). He served as an Associate Editor for IEEE Transactions on Computers, JETTA, JOLPE, TODAES, IEEE Design & Test Magazine, and IEEE Transactions on Very Large Scale Integration (VLSI) Systems. He is currently serving as a founding Editor-in-Chief for the Journal on Hardware and Systems Security (HaSS).