Fault Injection Attacks on Embedded Applications : Characterization and Evaluation

Zahra Kazemi

Université Grenoble Alpes

THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ DE GRENOBLE ALPES

Préparée dans le cadre d'une cotutelle entre

Spécialité : **NANO ELECTRONIQUE ET NANO TECHNOLOGIES**

Arrêté ministériel : 25 mai 2016

Présentée par

Zahra KAZEMI

Thèse dirigée par **Vincent BEROULLE, Professeur, Université Grenoble Alpes**
et codirigée par **David HÉLY, Maitre de Conférence, Université Grenoble Alpes**

préparée au sein des **Laboratoire de Conception et d'Intégration des Systémes (LCIS)**
dans **les École Doctorale d'Electronique, Electrotechnique, Automatique, Traitement du Signal (EEATS) et le Département d'Ingénierie Électrique**

Attaques par injection de fautes sur les applications embarquées : caractérisation et évaluation

Fault Injection Attacks on Embedded Applications: Characterization and Evaluation

Thèse soutenue publiquement le **3 Février, 2022**,
devant le jury composé de :

Monsieur Jean-Max DUTERTRE
Professeur, Ecole des Mines, Rapporteur

Monsieur Pascal BENOIT
MCF HDR, Université de Montpellier, Rapporteur

Monsieur Lilian BOSSUET
Professeur, Université de Saint Etienne, Président du jury

Madame Noémie BÉRINGUIER-BOHER
Hardware Security Expert, Brightsight SGS, Examinatrice

Monsieur Ioannis PARISSIS
Professeur, Université de Grenoble Alpes, Examinateur

Monsieur Mahdi FAZELI
Professeur, Université de Halmstad, Suéde, Invité

ACKNOWLEDGMENTS

First, i would like to sincerely thank my thesis directors, Prof.Vincent Beroulle and Dr.David Hély, for their time, help, and support during these years. It would take another dissertation to describe their positive effects on my work and my life in France. Their integrity and meticulousness have greatly influenced me as a researcher and an individual. This work would not have been possible without their continuous encouragement, guidance, and enthusiasm for my research.

In addition, i express special thanks to my co-advisor, Dr. Mahdi Fazeli, for his help, support, and fruitful discussions throughout my Ph.D. journey. He taught me the skills of a professional researcher, such as exposing a scientific problem and approaching it from different perspectives. Thank you for enlightening my life and supporting me like a brother through these years. Likewise, I would like to acknowledge Dr. Thanos Papadimitriou for being an exemplary mentor to start my Ph.D. This was an excellent opportunity to work with him and learn immensely from him.

I must express my gratitude to the jury members for their useful and productive questions and discussions. I would like to thank Prof. DUTERTRE and Dr. Pascal Benoit for all the time and effort that they put on carefully reading the thesis and writing an accurate and detailed report.

I also want to take this opportunity to thank the SERENE-IoT project, the Laboratoire de Conception et d'Integration des Systémes (LCIS) Laboratory, the Grenoble EEATS doctoral school, Grenoble Alpes Cybersecurity Institute, and IDEX for financially supporting this work, the conferences, and all travel.

I am deeply grateful to Dr. Peyman Pouyan for his generous encouragements and help during my Ph.D. journey. Also, i would like to give a special thanks to one of my best friends Dr. Ionela Prodan for supporting me like a sister, even in the most challenging moments. Last but not least, i would like to thank all my friends, colleagues, and labmates. I feel quite fortunate to meet and working many brilliant people at LCIS. Thank you, Jennyfer Duberville, Carole Seyvet, Caroline Palisse, Raphael Tavares De Alencar, Ehsan Aerabi, Amir Alipour, Ashkan Azarfar, Amin Norollah, Mahdi Talebi, Afef Kchaou, Baptiste Pestourie, Cyril Bresch, Johan Laurant, and Zishan Ali. Finally, I would not forget my family for their continuous love and support throughout all these years.

"This thesis is dedicated to my beloved parents, Farhad Kazemi and Elham Choolabi, for their unconditional love and support throughout all these years."

Abstract

The security assessment of IoT devices against potential software and hardware-based threats is now a necessary task for embedded software developers. Having physical access to the target devices makes hardware security a significant concern to consider in IoTs. Among the hardware security attack techniques, fault injection attacks such as clock glitching are one of the most practical attacks which are non-invasive and low-cost. They can interfere with the expected operations and cause serious malfunctions in the targeted device. Regarding this, an efficient security assessment framework and methodology against fault injection attacks are needed to properly evaluate the embedded devices.

It is often difficult for the software developer to use a fault injection platform correctly. Therefore, this thesis focuses on designing an easy-to-use platform dedicated to clock glitching attacks in order to evaluate the vulnerabilities of embedded software applications. This work proposes an open-source evaluation platform followed by high-level assessment methodologies. Then, a characterization process based on a preliminary simulation approach is presented to improve the experimental fault injection parameters. Finally, the impacts of the injected faults are analyzed and studied in an open-source medical application (Sec-Pump) as a case study. The platform and the methodology proposed in this thesis can successfully identify the security vulnerabilities in an embedded application and guide the software developer to mitigate such attacks.

Keywords— Hardware Security, Embedded Systems, Fault Injection Attacks, Clock Glitching

Résumé

L'évaluation de la sécurité des appareils IoT contre les menaces logicielles et matérielles potentielles est désormais une tâche nécessaire pour les développeurs de logiciels embarqués. Avoir un accés physique aux appareils cibles fait de la sécurité du matériel une préoccupation importante à prendre en compte dans les IoT. Parmi les techniques d'attaque de sécurité matérielle, les attaques par injection de fautes telles que les problémes d'horloge sont l'une des attaques les plus pratiques, non invasives et peu coûteuses. Ils peuvent interférer avec les opérations attendues et provoquer de graves dysfonctionnements dans l'appareil ciblé. À cet égard, un cadre d'évaluation de la sécurité efficace et une méthodologie contre les attaques par injection de fautes sont nécessaires pour évaluer correctement les dispositifs embarqués.

Il est souvent difficile pour le développeur de logiciels d'utiliser correctement une plate-forme d'injection de fautes. Par conséquent, cette thése s'est concentrée sur la conception d'une plateforme facile à utiliser dédiée aux attaques par défaut d'horloge afin d'évaluer les vulnérabilités des applications logicielles embarquées. Ce travail propose une plateforme d'évaluation open source suivie de méthodologies d'évaluation de haut niveau. Ensuite, un processus de caractérisation basé sur une approche de simulation préliminaire est présenté pour améliorer les paramétres expérimentaux d'injection de fautes. Enfin, les impacts des failles injectées sont analysés et étudiés dans une application médicale open source (Sec-Pump) en tant qu'étude de cas. La plate-forme et la méthodologie proposées dans cette thése peuvent identifier avec succés les vulnérabilités de sécurité dans une application embarquée et guider le développeur de logiciels pour atténuer de telles attaques.

Mots clés— Sécurité matérielle, Systémes Embarqués, Attaques par Injection de Fautes, Glitch d'horloge

CONTENTS

LIST OF FIGURES

LIST OF TABLES

List of Abbreviations

IoT The Internet of Things

IoMT Internet of Medical Things

IIoT Industrial Internet of Things

SCA Side Channel Attack

FIA Fault Injection Attack

CF Control Flow

DF Data Flow

CSC Combine Shifted Clocks

CDCF Combine Different Clock Frequencies

VM Voltage Multiplexing

VSC Voltage Short Circuiting

VGPG Voltage Glitching using Pulse Generator

DCM Digital Clock Manager

MMCM Mixed-Mode Clock Manager

DRP Dynamic Reconfigurable Port

1 INTRODUCTION

Contents

The Internet of Things (IoT) connects objects and devices of all types over the Internet, either wired or wireless. This technology has transformed many aspects of our daily lives, and there are many useful applications for these devices. For instance, IoTs help to make smarter, safer, more comfortable, and energy-efficient homes [1]. Moreover, they have a high potential to improve and automate healthcare services. The Internet of Medical Things (IoMT) has already been employed at home and in hospitals to enhance the safety and efficiency of medical services [2]. The banking sector and other financial areas are also striving to make use of IoT and benefit from it. This way, the customers can always stay in touch with their bank, which makes it possible to gather more data about their behavior and preferences [3]. Additionally, IoTs are employed in industrial sectors such as manufacturing, energy, mining, and transportation [4]. These intelligent devices in the industry, so-called Industrial IoT (IIoT), are usually connected to a central system that can monitor, collect, exchange, and analyze the gathered data.

Despite all of these applications and achievements for IoTs, their rapid technology usage comes with various security challenges. The security attacks against them can result in dangerous and costly outcomes, i.e., it can reveal personal information in public [5,6]. The attacks can be launched against any IoT assets and facilities. They can potentially damage or disable a system's regular operation, which can cause severe economic damage to the owners/users. An example includes an attack on medical IoT systems and taking control of the monitoring mechanisms [7]. Moreover, the personal data from an embedded sensor inside or close to the patient's body can be collected and transferred to the adversary. Regarding this, performing security assessments against real examples like medicine injection pumps is required to demonstrate the potential risks of numerous security flaws of life-critical medical IoTs [8, 9]. As an example, an attacker can maliciously modify the

functionality of a pump and target running software and raise the amount of the injected insulin over time [9].

In general, the attacks against IoT embedded devices can be classified into three main categories, including 1) Network, 2) Software, and 3) Hardware Attacks. In practice, an attack can employ any or all of these approaches. In principle, the Network-based Attacks could be applied remotely at any point of the interconnected IoTs. There are various studies that show IoTs are susceptible to Network Attacks such as Denial of Service and Spoofing [10]. The second class of security attacks against IoTs is applied at the software level. They can be applied at various software abstraction layers, such as in high level and low level. For instance, some High-level Software Attacks are brute force attacks that target an application that consists of a pair of input/output to get authenticated or to reveal the information. Other examples aim to inject malware or manipulate the machine-level code at lower levels and hijack the application's execution flow [11]. Besides software attacks, numerous security threats exist against the user-accessible targets named Hardware Attacks [7, 12]. These attacks become critical when the attacker can have direct physical access to measure the device operating parameters (e.g., power and propagation delay signal) or can tamper with the external inputs of the targeted embedded device.

There are various techniques to apply Hardware Attacks against embedded systems. Two of the main techniques which are considered in this thesis are the Side-Channel Attacks (SCA) and the Fault Injection Attacks (FIA). One can perform SCAs to extract useful information (e.g., cryptographic key) by observing the physical characteristics such as power consumption or electromagnetic emissions while the device is executing a specific operation. In FIAs, the attacker tries to manipulate the device's input or emit different energy rays to circumvent the security checks (e.g., user authentication) or execute an arbitrary code that causes unintended behavior of the target (i.e., by changing the application's control flow). There are also multiple combinations of FIA and SCA, where the SCA takes advantage of fault impacts on the targeted device. For example, in [13,14], a FIA reduces the number of rounds of a cryptographic algorithm, and the SCA can extract the pass key faster.

To design a secure embedded system, one needs to follow a set of assessment procedures against different types of threats in the three domains of network, software, and hardware. The first two fields have been investigated and extensively analyzed over the past years [15]. Subsequently, numerous countermeasures are proposed and employed to secure the systems against such vulnerabilities [16, 17, 18]. However, securing the system against network and software attacks is often inadequate to achieve the desired security protection, especially in easily accessible targets. Thus, it is required to consider the hardware security and to include different evaluation processes against physical threats. To design an efficient hardware evaluation methodology, one first needs to understand the fundamentals of these attacks. Regarding that, the background and concepts of the Hardware Attacks are described in the following.

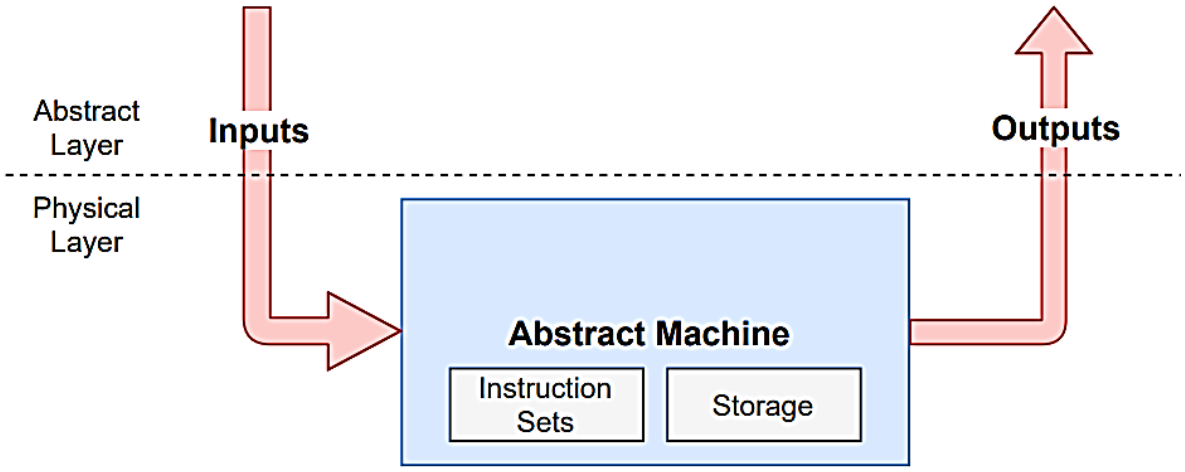

Figure 1.1: The Abstract Machine of an Embedded Systems

1.1 Basic Concepts of Hardware Attacks

Embedded systems, at the beginning of their emergence and early developments, were not studied in terms of security as of today. They were mainly considered as models called abstract machines. Figure 1.1 shows an example of such abstract machines, including the inputs, the outputs, and the relevant operation sets, usually at the high level of the system. In this case, it was assumed that an attacker could only access and compromise the input and output of this system model [19, 20]. Figure 1.2 shows an example of an encryption machine in which the attacker can choose plain texts freely, observe the generated cipher texts, and mathematically break a cryptography algorithm (to reveal the key). Similar attacks can be applied against different designs to discover their internal executive algorithm and extract secret information. These attacks, as they were mostly utilized to hack the cryptography systems, were termed Crypto-Analysis attacks. Since then, the ability and success rate of many cases in classical crypto-analysis attacks have been reported [21].

Along with the classical analysis attacks, the implementation-specific characteristics of the embedded systems, which are referred to as side-channels, became an important concern in the field of security. In this regard, the vulnerabilities of the newfangled category were reported in security systems in which they were considered very secure from the mathematical point of view. The NSA, for example, pointed to signs of detected vulnerabilities in encrypted and teletyped messages by a conventional oscilloscope [22]. Besides, in the 1990s, N. Kocher et al. presented a new topic called Hardware Attacks, which officially came alongside the conventional attacks of that time [20, 23]. In these attacks, vulnerabilities related to the hardware layer of the target system were analyzed in a more specific way compared to the classical attacks. Since then, Hardware Attacks have emerged as a new and powerful approach for attackers to compromise the accessible embedded systems and their security [6].

A Hardware Attack is based on the interactions of an embedded system with its external en-

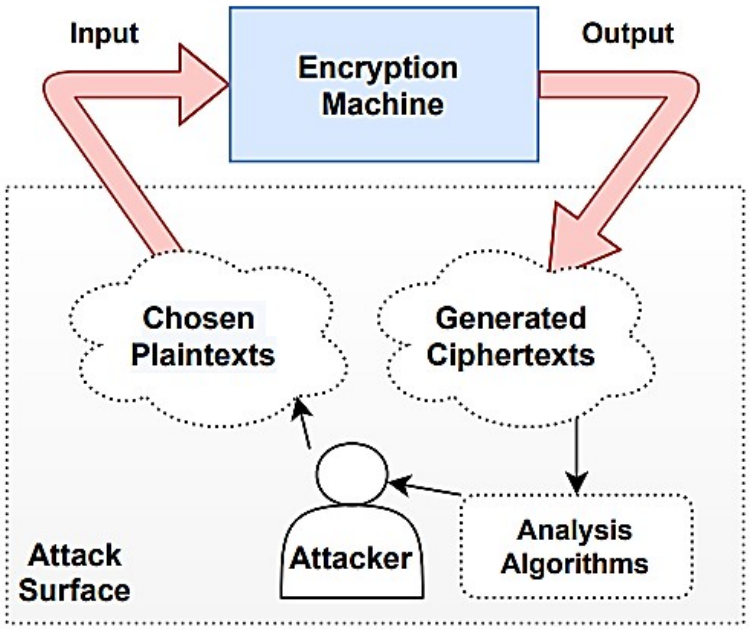

Figure 1.2: An Example of Traditional Assumption for Crypto-analysis Attacks against an Encryption Machine

vironment [6]. Figure1.3 shows the taxonomy of Hardware Attacks. Depending on the goal of the attacker (e.g., to change the expected behavior of the system or to obtain encrypted information), the appropriate attack model and method must be determined [6,24]. Accordingly, two of the most important approaches including 1) Side-Channel Attacks; 2) Fault Injection Attacks, are studied, where the main focus in this thesis has been on FIA.

1.1.1 SIDE-CHANNEL ATTACKS

Side-Channel Attacks are based on gathering the target's produced unintentional outputs, physical characteristics, or observable signals while executing the software. They usually do not require manipulation of the target device, and the attacker can passively evaluate the system interactions [19,25]. An attacker can take advantage of these available side-channel parameters, such as the power consumption [20], the electromagnetic emanation [26], and the thermal signature [27], to mount an SCA in order to obtain the critical data and to leak the secrets from an embedded device [7]. Figure 1.4 depicts some common side-channel parameters, including Electromagnetic emissions, Timing information, Acoustic leakage, and Power signals, which have leaked from a target device during data processing. All types of discovered side-channels have been reviewed in [19].

The SCA's most basic instances have been performed against cryptographic machine implementations to extract their secret key [28,29]. The subsequent works employed a similar approach on various side-channels and targeted implemented ciphers (e.g., DES, AES, and RSA) inside the

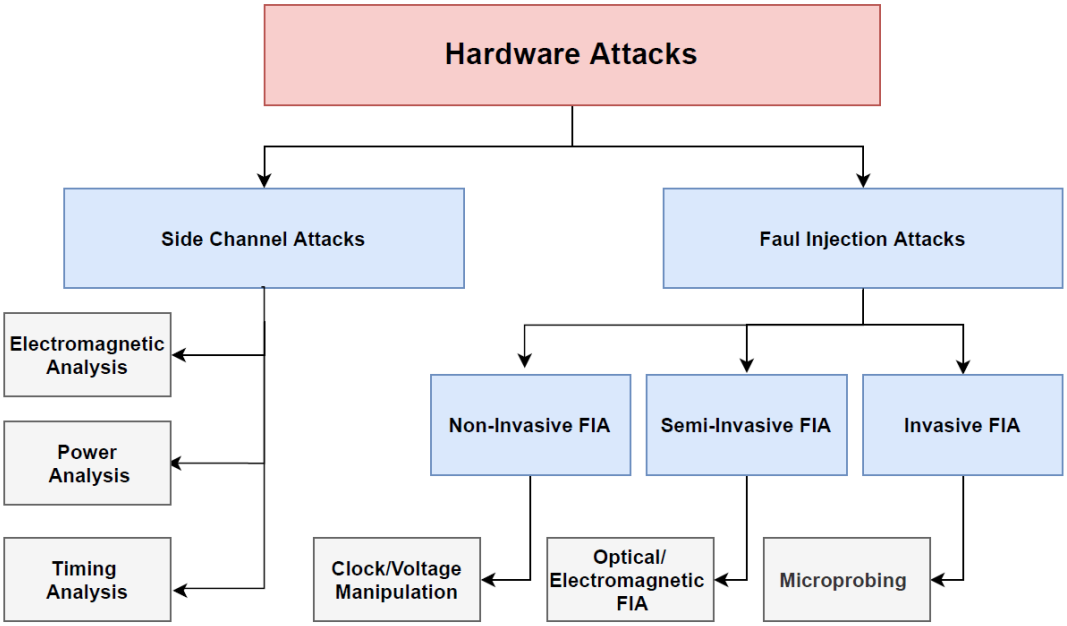

Figure 1.3: Taxonomy of Physical Attacks

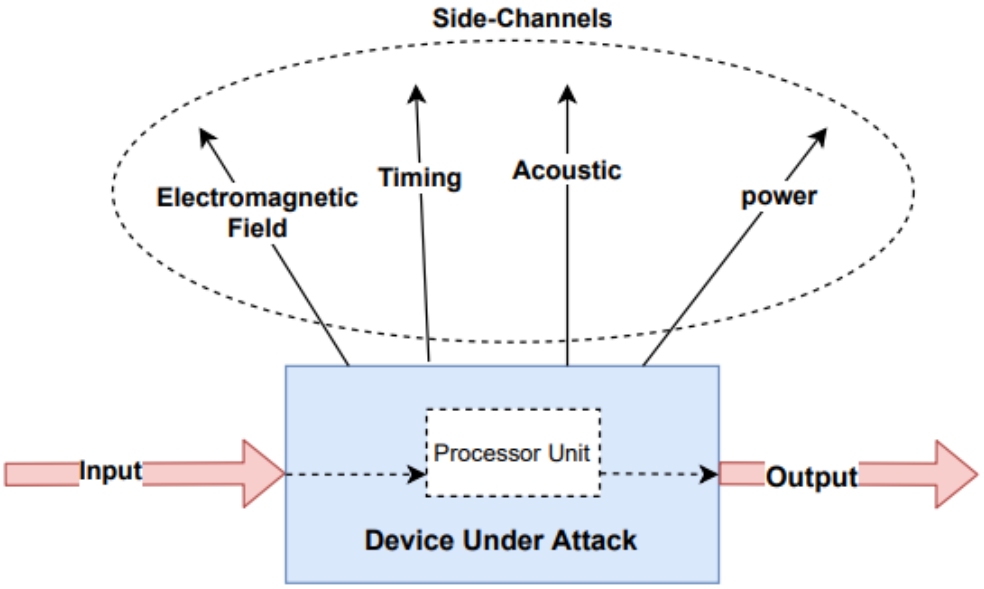

Figure 1.4: Side-Channels from an Embedded Device

smart-cards, mobile phones, personal computers, dedicated ASICs, and different microprocessor architectures. These examples demonstrate that SCAs can pose a significant challenge to the security of IoT devices.

SCAs are performed in two phases:1) Interaction phase, 2) Exploitation phase. The first phase is to find an observable and measurable physical characteristic of a target and monitor that, generally without any modification or opening up the chip (usually during the normal operational mode). Note that this step often needs full access to the device for the attacker, who can run it iteratively several times. Occasionally, based on the level of control an attacker may have on a device, the SCA can be performed with the appropriate input vectors to get optimized results [25]. The exploitation phase analyzes the collected data to extract the important physical information related to the non-functional and internal operations. Although different SCAs may have their unique interaction phase (way of data measurement), they all follow a very similar approach to their exploitation phase [30].

1.1.2 FAULT INJECTION ATTACKS

Fault attacks are the noticeable type of physical attacks, in which the expected and secure behavior of the targeted devices is liable to be jeopardized.Fault Injection Attacks have been designed and introduced in various methods, such as 1) By manipulating the inputs of the device (such as clock or voltage); 2) By stressing the target through changing its surrounding conditions (such as raising the temperature); 3) By emitting energy rays (such as electromagnetic or laser). They can either modify the process of software execution or the stored values inside the memory locations. FIAs can be classified into three categories, namely: 1) Invasive, 2) Non-invasive, and 3) Semi-invasive [31].

Invasive attacks are the type of intrusions performed by de-packaging the Integrated Circuit (IC) and modifying the physical properties to do some probing. Invasive attacks, such as Microprobing, laser, or optical fault injection, are the strongest physical attacks without imposing any restrictions on accessing the inside of the chips [32]. These kinds of attacks are very powerful in precise space-time positioning, and they can reveal considerable secret information from internal parts of the chip to the attackers. However, they are expensive, mostly irreversible, and complicated. In addition, they have to be performed by qualified specialists in laboratories equipped with special devices. They are usually applied against very secure devices such as smart-cards or complex Commercial Off-The-Shelves (COTS) components, in which the target is in the form of industrial products, and the attacker usually has more features and knowledge [33]. Nonetheless, in some cases, such attacks are not appropriate methods for individual IoT hackers because they are too costly for them, and also, they do not have access to wide-range facilities.For the consumer IoTs, the attack would be in most cases too costly for the gain.

The semi-invasive attack is a type of FIA that stands between invasive and non-invasive at-

tacks [34, 35]. This type of attack can involve de-packaging the IC layers to gain access to the internal surface of the target, but normally, the passivisation layer remains unimpaired. For instance, optical fault injection is a semi-invasive attack in which the protection layer of the device has to be ruined [34].

Finally, non-invasive attacks are the type of non-destructive attacks that can be accomplished by only utilizing pin-probing or bus-snooping without damaging the package [32]. Two of the simplest non-invasive fault attacks are tampering with the device clock signal and/or the supply voltage, the so-called clock and voltage glitch attacks [36].

Among those different kinds of FIAs, this thesis considers the non-invasive FIAs against secured embedded applications because there are more threatening than invasive attacks for IoT devices. This is for three main reasons:

- They do not require any physical tampering; refer to the owner of the targeted device might not notice the attack and trust in functionality and security.
- They can be reproduced and updated by using low-cost and easy-to-access equipment, even in a small laboratory [37].
- They have proven that a high success rate can be achieved in a short time [38].

Since such threats can jeopardize embedded software, it is necessary for software developers to evaluate the potential vulnerabilities due to FIAs. Therefore, there should be a systematic security assessment approach to identify the security assets in terms of important functions and data, discover the vulnerabilities, define risks, and illustrate the probability and consequences of the potential successful Hardware Attacks. Discovering the impacts of physical FIA is not always straight forward. So, one needs to understand the FIA effects and their propagation through different levels. Figure 1.5 shows an example of the propagation of an injected fault through layers of an embedded system along with its effects on the target.

These effects can be classified into different categories:

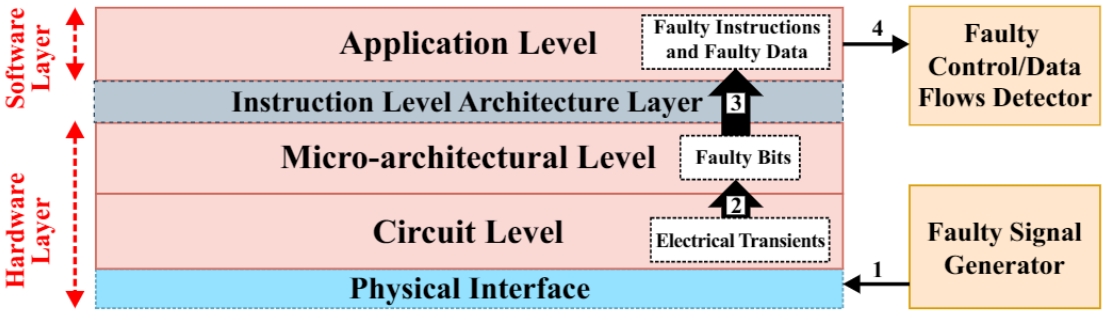

Figure 1.5: Fault propagation through different layers, (1) Fault Injection, (2) Fault Manifestation, (3) Fault Propagation, (4) Fault Exploitation

- Faults at Circuit-Level: The physical stress on any target interface leads to transient electrical faults like transient voltage glitches or current spikes at the circuit level resulting in gate faulty behavior.
- Faults at Micro-Architectural Level: Transient electrical faults might be captured by the latches and flip-flops in the system's data or control paths, resulting in erroneous micro-architectural states or data.
- Faults at Software/Application Level: Faulty values captured by different micro-architectural blocks would cause errors in the control or data flow of the running software. In other words, a fault at the micro-architectural level manifests itself as a deviation in the correct instruction flow or as a faulty operand or opcode at the software level. Note that the faults at the software level can be exploited in different manners.

In general, the exploited vulnerabilities at application level can be modeled as 1) Control-Flow Corruptions (CF-Corrupt) and/or 2) Data-Flow Corruptions (DF-Corrupt) at the application level. The CF-Corrupt can occur by disrupting the intended order of instructions, branches, or statements of the embedded software. Accordingly, several works have shown that even non-invasive FIAs such as clock/voltage glitching attacks can lead to CF-Corrupt by skipping or repeating one instruction or by replacing that with another instruction [39, 40]. Other CF-Corrupt instances happen when the evaluation step of a conditional branch has been skipped, and the incorrect branch is taken [41]. FIAs can alter the conditional branch instructions, which are used to implement loops and change conditions in security checks of embedded software.

DF-Corrupt happens when the attacker compromises the integrity or confidentiality of processed data by disturbing the targeted MCU. For example, they can corrupt a single bit, a single byte, multiple bytes, or a single word of a security-critical variable in various ways (e.g., flip, set, reset, random) [42,43]. Furthermore, one can exploit the vulnerable arithmetical/logical instructions to generate intermediate or final faulty results. By repeating this procedure and obtaining more erroneous computational values, it is possible to leak secret and sensitive information [44]. On the other hand, FIAs can affect the memory operations such as load, store, and copy instructions and lead to DF-Corrupt [45]. These manipulated values can directly affect the system behavior, mainly while other computational or condition evaluation instructions are using them.

Considering the mentioned vulnerabilities, the security specialists aim to mitigate the destructive consequences of FIA by developing a set of software and hardware level defensive countermeasures. For example, redundancy-based protection methods are one of the most utilized software-based methods, in which one important instruction is repeated, and its result is compared; Then, one detects the mismatch and runs the error management routine (if necessary) [46]. Another approach is named the duplication method without comparing and fault detection steps [47]. The signature-based protection is another approach to counter-based protection, and that assigns a unique identifier to all of the basic blocks inside the control flow graph. Then, it verifies the identi-

fier/signature by every control-flow transfer. Basic examples of hardware-based countermeasures are applied in different ways, such as utilizing passive shields to cover the vulnerable part of the chip and using specific logic such as autonomous frequency detectors that can detect the glitches in the clock signal [48].

The software-based countermeasures have significant performance overhead, and they cannot guarantee complete code integrity against fault injection attacks. However, in many non-critical cases, they can provide a good trade-off between hardware cost and security. Accordingly, hardware-level countermeasures can be applied for safety-critical embedded devices to protect the design against stronger FIAs. Moreover, hardware-based mechanisms have better performance than software solutions since fewer processor cycles need to be spent on performing the security checks. The main drawback of the hardware-based countermeasures is their cost for embedded devices.

1.2 THESIS STATEMENT AND MAIN OBJECTIVES

Embedded software evaluation against hardware-based security attacks has recently gained compelling attention in industry and academia. Although several assessment tools and platforms have been developed, choosing the right hardware platform is not always easy and depends on many parameters. In response, the first objective of this thesis is to review some of the existing assessment tools, specifically against non-invasive fault injection attacks, and to classify them according to their specifications and features. This study helps to determine the important factors of an hardware evaluation platform by considering the budget and the design effort.By budget, the aim is to use low-cost devices and chips and by design effort, the aim is to have a easily configurable platform. The way that IoT products are developed today - especially with their reduced" time to market"- does not always give time to perform a third-party hardware security evaluation. Nowadays, developers get more involved in performing the evaluations by themselves. The existing commercial evaluation tools have been simplified and matured considerably to satisfy the developers' needs; however, they are dependent on the use of specific hardware, closed-source, not accessible to deploy for the low-cost IoTs and not easy to exploit the software-level vulnerabilities. On the other hand, very little research has been conducted to determine whether the hardware security analysis instruments fulfill the embedded software developers need to secure their code. Accordingly, the second objective is to provide one open-source and low-cost hardware evaluation tool, which is highly important in the embedded software development stage.

When such a tool is available, the target users (e.g., non-security specialists or embedded software developers) must be guided properly to apply the practical experiments. Unfortunately, the previous research works have only focused on the in-depth evaluation of low-level specific instructions, and these isolated approaches cause inefficiency for a broad evaluation purpose. So, this research's next objective is to define comprehensive and function-level evaluation methods to be

executed in order to detect the embedded software vulnerabilities. Combining these scenarios and the open-source platform designed in this thesis can significantly simplify the evaluation process at the development phase.

The introduced hardware platform in this thesis, can efficiently apply fault injection attack into an embedded system; however, it still requires significant time and analysis effort to detect the vulnerabilities of a target. In practice, the evaluation time plays an essential role in the embedded software design step, and one cannot ignore or just accept it as "that is how long it takes." Likewise, this approach lacks information about the root causes of the revealed vulnerabilities and cannot guide the target user to fix these issues or add proper countermeasures. Accordingly, this thesis improves the evaluation process's critical factors (e.g., accuracy, coverage, and time).

The proposed experimental FIA platform along with its associated tools and the defined methodologies in this thesis work construct an evaluation framework,to be used by embedded developers. Accordingly this framework is verified in a case study to have a qualitative study. Among various attention-demanding case studies, this thesis focuses on a medical IoT application named Sec-Pump, which is also in the context of the SERENE IoT project. The SERENE IoT project (Secured EneRgy EfficieNt hEalth-care solutions for IoT market) aims at developing high quality smart e-health IoT devices in Europe. SERENE-IoT project is labeled within the framework of PENTA, the EUREKA Cluster for Application and Technology Research in Europe on NanoElectronics. The project contributes to developing high quality connected care services and diagnostic tools based on advanced smart health-care IoT devices. The revealed vulnerabilities from hardware security evaluation of the Sec-Pump can detect the real potential risks in similar critical applications for the end-users and the service providers.

In brief, the main objectives of this thesis are summarized in the following:

- To implement an open-source, low-cost and efficient hardware evaluation platform to apply clock glitching FIA. This hardware platform will be generic and can be used for various embedded targets.
  - To investigate and review the state of the art of the existing practical fault injection platforms that are used to attack various embedded systems
  - To analyze the cost overheads, the advantages and the disadvantages of the proposed hardware evaluation platform in front of other existing platforms
- To define a practical evaluation process for the developers and non-security specialists at the design stage to identify the potential security vulnerabilities
  - To evaluate the potential vulnerabilities at high level of the software and to narrow down the fault injection time intervals by using the results from the simulation-based experiments respectively

  - To improve the coverage, efficiency,accuracy of injected faults and to simplify the analysis process for embedded applications

- To apply our evaluation framework which consists of a clock glitching FIA hardware platform and our evaluation methodology into a practical case study (Sec-Pump)

  - To identify the security vulnerabilities of an IoT application (Sec-Pump as an example) by using our evaluation framework

1.3 THESIS CONTRIBUTIONS

The main contributions of this thesis are summarized in the following:

- **The Existing Fault Injectors are Reviewed**

  The first contribution of this work is to study the different non-invasive fault injection approaches and platforms. This helped to extract their important factors such as complexity, cost, and required usage expertise. The thesis focus has been on the clock, and voltage fault injection approaches due to their effectiveness and affordable equipment. Accordingly, this thesis first reviews the proposed clock and voltage glitch generators in the literature and categorizes them based on different essential parameters in the attack process.

  This part has been published in [6].

- **A Hardware Evaluation Platform for FIA is Designed**

  The reviewed state of the arts was challenging to adapt for general IoT designs and it lacked the proper configuration characteristics. The second contribution of this thesis is in Chapter 3, which introduces a practical evaluation platform to evaluate an MCU-based system running a software application against the Clock Glitching FIA. This platform is specifically focused on the clock glitching FIAs, and it is a low-cost and easy-to-use interface for non-security specialists. Then, the utility of this platform is demonstrated by applying FIA to an encryption algorithm and analyzing its results.

  This part has been published in [44].

- **Application-level Test Scenarios is Proposed**

  Another contribution of this thesis is to define high-level test scenarios to exploit and analyze the vulnerabilities of a target embedded software after injecting faults. This analysis is applicable for high-level patterns and standard C functions. After this analysis, a report will be generated, which can help the embedded developer to mitigate the vulnerabilities in the early developing stage of the application. The main focus of this thesis approach is on the analysis of the most prominent control flow and data flow integrity objectives.

  This part has been published in [45].

- **An Evaluation Framework for Software Developers is Provided**

  The fourth contribution is to provide a complete overview of the threat modeling for the critical assets of an embedded application against FIAs. The software developers should be guided correctly to identify the right critical assets of their application and then conduct an appropriate assessment against the potential attacks. Therefore, in addition to an evaluation platform, a methodized approach is defined to analyze the target code step by step.

  This part has been published in [7, 49].

- **The Timing Characterization of Clock Glitching FIA is Improved**

  The next contribution is to introduce a mixed simulation-experimentation methodology to detect the security flaws against the clock glitching FIA. First, the simulation operates on the defined fault models as fault injection attacks into the RISC-V micro-architecture. This can help to improve the FIA timing characterization in the experimental attacks. Therefore, the experimentation attack could give the analyzer more detail and more precise results.

  This part has been published in [50].

- **Symbolic Assertion and Code Shredding to Obtain A Global Vulnerability Factor**

  The last contribution of this thesis is to use the symbolic assertions and obtain a global vulnerability factor for the embedded software evaluation against the clock glitching FIA. Accordingly, a partitioning approach is applied to divide an application into various code blocks with respect to the functionality and the main variables of each code block. Then, the detection patterns are inserted into the code to report the successful attack and obtain the vulnerability factor. This approach is an efficient criterion to evaluate all the corner case vulnerabilities of software blocks against FIA. The final goal of this thesis is to verify that by using such an approach, one could show the potential risks of the Sec-Pump blocks.

1.4 ORGANIZATION OF THE DISSERTATION

The remaining part of the dissertation is organized as follows (Figure 1.6): Chapter 2 explains the basics of fault injection attacks and reviews the state-of-the-art of the existing fault injectors. Chapter 3 describes the details of a proposed evaluation platform against fault injection attacks. It will then show the experimental results and verify the usage of the platform. Chapter 4 presents the steps of a high-level evaluation approach for embedded software developers. To explain the introduced methodology, it has been applied in a case study (Sec-Pump). Chapter 5 studies the optimized evaluation methods to improve the assessment results. It is based on utilizing the simulation approach in order to improve the experimentation parameters. Finally, the conclusions and perspectives of this thesis are presented in Chapter 6 and 7.

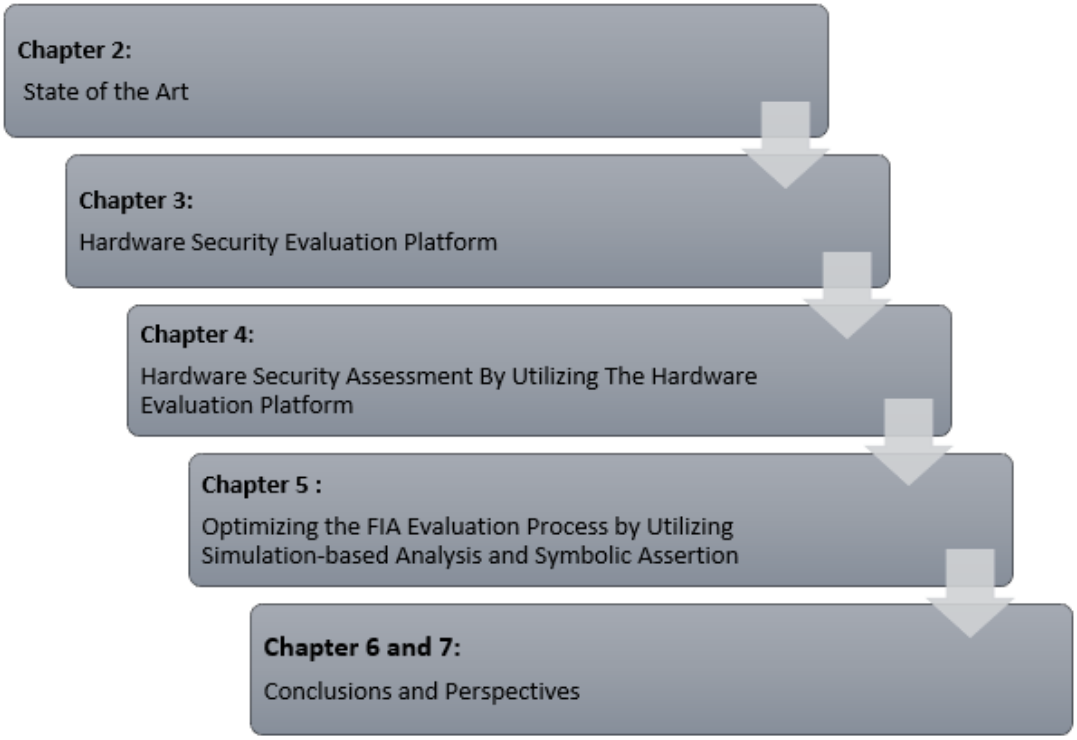

Figure 1.6: The road map for the dissertation

2 STATE OF THE ART

Contents

This chapter studies the fundamentals of FIAs and reviews the main fault generator parameters. These factors determine the applicability and efficiency of a fault injection platform. Then, the principles of clock and voltage generators are studied, and their important specifications are determined. Finally, it reviews the state-of-the-art clock and voltage fault generators and categorizes them based on their features.

2.1 A BASIC SETUP FOR A FIA PLATFORM

Figure 2.1 shows an experimental setup for a fault injection system. It includes a target board, a fault generator, and a controller computer. The fault generator is the crucial component of a fault injection system. In the following, the main characteristics of a fault generator, which affect the success rate of the attacks, are described.

Generally, in the hardware-controlled fault injection techniques, a separate external fault generator is used to induce faults in the running application. It is essential to produce well-controlled faults utilizing the generator in order to achieve the desired results. It is, therefore, a challenging

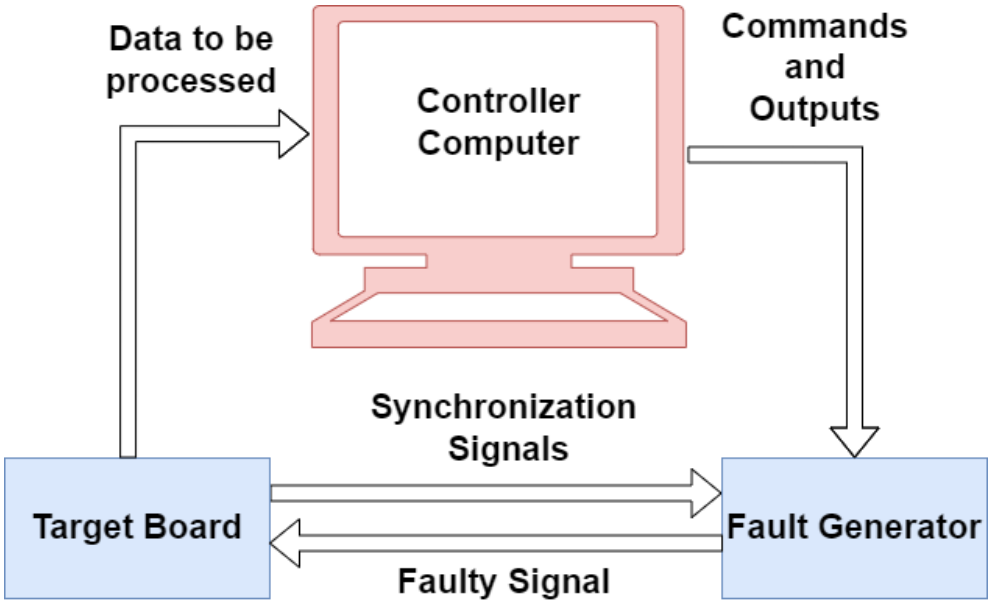

Figure 2.1: General Fault Injection Attack Setup

procedure to design a fault generator with a high level of accuracy and precision. Several factors determine the applicability and efficiency of a fault generator. Below, a list of features are identified, which are the main parameters for a hardware-based fault generator [38,51,52,53,54,55,56,57,58]:

- Accuracy in the generated faulty signal: The accuracy of the generated faulty signal can be defined as our control over the fault parameters. In fact, the more control one has over the generated faulty signals, the more achievement in successful attacks in terms of generating and exploiting the desired faults. Control over the time and location of injected faults can be achieved using the communication lines between the controller PC and the fault generator.

- Run-time fault configuration: In some evaluation platforms, the parameters of a faulty signal produced by the generator can be reconfigured during run-time. They are not limited to the design time and can be used to test a target system during the attack phase. This feature is available in some generators and is based on some kinds of FPGAs, allowing modification of some parameters during the run time [51].

- Reproducibility of the faulty signal: Reproducibility is the ability to obtain the same results in multiple repeats of the same test. Regarding this feature, the faulty signal should be reproducible and validated by a third party if the same hardware is used (for example, the same FPGA). This parameter becomes more important when the same evaluation platform is used multiple times to generate a faulty signal [38, 51, 52, 53, 56, 58].

- Randomness in faulty signal: Many of the faulty signal-generating methods presented, such as [59, 60, 61, 62, 63], are deterministic methods. To set up a fault generator and to apply an attack with high accuracy, all the output parameters must be prepared and calculated. However, as this requires complete and adequate information about the target, it is not applicable

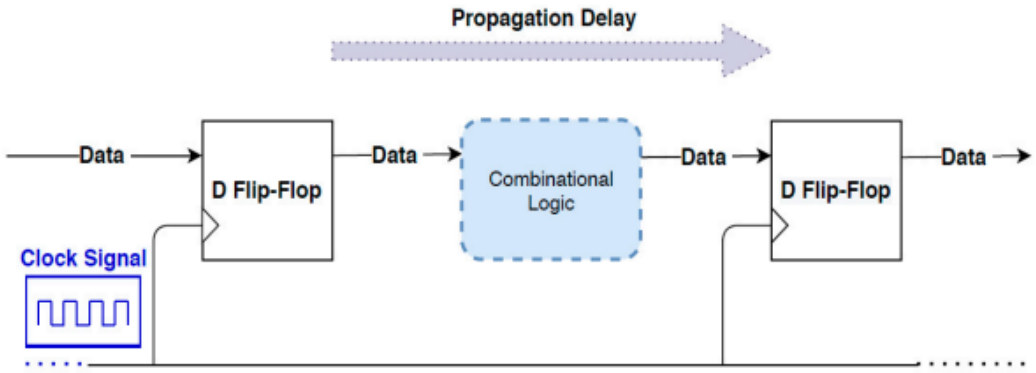

Figure 2.2: Synchronous Representation of Digital ICs [55]

in all real-world attack scenarios. In addition, in order to achieve an efficient attack, a large number of possible combinations of the fault injection parameters need to be covered. This makes deterministic methods intractable within a limited time. To address this issue, a fuzzy-based fault generator has been proposed. It has been reported that this method has a high attack success rate [54].

These factors for the clock and voltage fault attacks include accuracy, development expenses, the complexity of the system setup, and user expertise.

In the following, the basic concepts and significant parameters in clock and voltage fault injection methods are reviewed.

2.2 Clock-Based FIAs

Clock-based fault injection is a low-cost attack that can be applied by the attacker to devices supplied with an external clock such as smart cards. If the target uses an internal clock signal, this method is often not applicable. The fundamental concepts and related characteristics for this approach are provided in the following sub-sections.

2.2.1 Concepts of Clock FIAs

First of all, to explore the importance of proper timing in digital ICs, one has to understand the synchronous design concept [55]. This is the basis for simple to complex computing systems. Digital designs often consist of two main parts: 1) combinational logic for computational operations and 2) memories such as register banks to store the computation results after each clock cycle. Figure 2.2 shows the concepts of propagation delay and setup time as irrevocable delays for performing computational operations. They are important timing parameters and need to have valid stored values in the flip-flops.

The inequality in equation 2.1 should be satisfied in order to guarantee the correct behavior of the flip-flop. T_{clk} represents the clock period, $T_{critical}$ represents the minimum time needed to

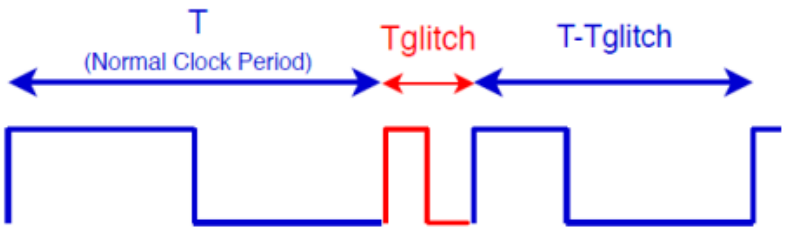

Figure 2.3: Violating Critical Path Delay by Insertion of Additional Positive Clock Edge

process the data through the combinational part, and T_{Setup} represents the setup time of flip-flops, specified as the minimum amount of time for which data input needs to be stable before the active edge of the clock:

$$T_{clk} > T_{Critical} + T_{setup} \tag{2.1}$$

One way to perform fault injection attacks is to violate these timing constraints. Violation of the time constraints induces faults in the target. $T_{critical}$ and T_{Setup} are two parameters that are dependent on the system logic design and the technology, therefore they cannot be manipulated to perform fault injection. Unlike the previous parameters, T_{clk} is a knob used for attackers to carry out their fault injections. Clock fault injection is applied by tampering with the clock signal temporarily or permanently. There are two different methods of clock fault injection: 1) Overclocking and 2) Clock glitching. Overclocking is a kind of timing violation attack in which one tries to apply a clock signal with a higher frequency than the nominal frequency for a specific time interval [55]. In fact, the overclocking method violates the timing constraint inequality (eq1) by decreasing the clock period. A clock glitch is regarded as an unwanted transition in the clock signal.

In the clock glitching method, the attacker generates glitches in the clock signal. The induced glitches produce extra edges in the clock signal, resulting in an erroneous output as the timing inequality has been violated. Figure 2.3 shows a typical clock signal in which a glitch is induced. In this figure, T represents the normal clock period, and T_{Glitch} is the width of the glitch signal. As it can be seen, an extra edge appears in the clock signal. Another important parameter is T_{min} , which is equal to the reciprocal of maximum frequency. In order to have erroneous behavior, T_{Glitch} should be less than T_{min} [55]. Moreover, in [52], the authors have considered "T-T_{Glitch}" as a "post-glitch" phase. This abnormal semi-clock part could be considered as another approach for fault injection. In this case, if "T-T_{Glitch}" is less than T_{min}, it leads to an erroneous behavior. Figure 2.4 shows how a clock glitch can inject faults in the system and how these faults can propagate through the next clock cycles [64].

Table 2.1 compares the accuracy of overclocking and clock glitch attacks. The first column, entitled spatial precision, refers to the level of accuracy with which the fault generator can inject a fault into a specific location. Temporal precision, the second column, is defined as the accuracy of the fault injection process in inducing a fault at a specific time. The third column reveals glitching parameters such as frequency, width, and amplitude. The last two columns show equipment costs

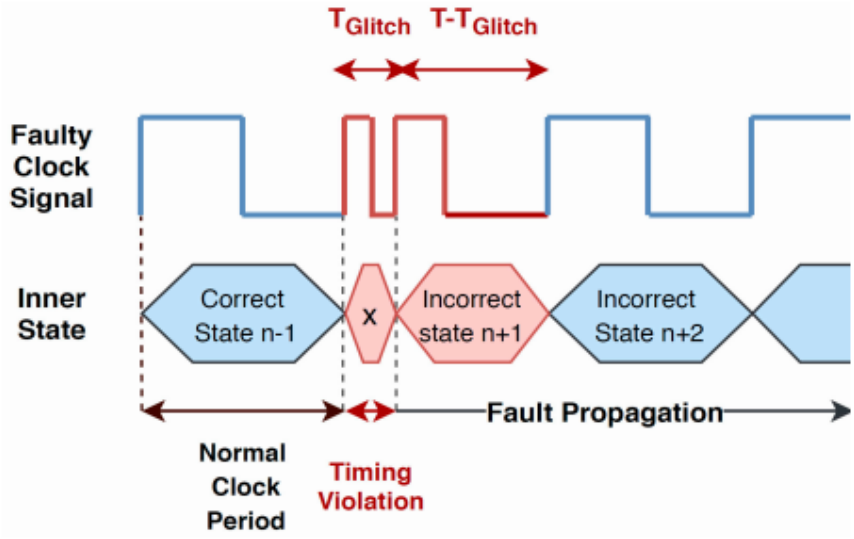

Figure 2.4: Representation of Clock Glitch and Fault Injection [64]

Table 2.1: Clock Fault Injection Techniques and Characteristics

Technique	Spatial precision	Temporal precision	Controlling the intensity	Equipment costs	Required expertise
Over Clocking	Low (global)	Not Applicable	Clock frequency	Low	Low
Clock Glitching	Low (global)	High (local)	Glitch parameters	Low	Moderate

and the required level of expertise for fault injection, respectively. Since the clock is a global signal, sufficient control is needed to induce a fault in a specific location in the system. Consequently, fault induction in the clock signal would not have high spatial accuracy with either the overclocking or clock glitching techniques.

In the clock glitching method, high temporal precision can be achieved by synchronizing the target and the fault generator circuit. Temporal precision is meaningless in the overclocking technique as the clock frequency is considered a fault. Briefly, the clock glitching technique is more desirable than the overclocking technique as it provides more accuracy and flexibility to manipulate clock signal parameters. In fact, the overclocking technique is not applicable for injecting a fault into a specific instruction at a specific time. In such cases, clock glitching is the only technique for fault attacks. For accurate fault attack using the clock glitching technique, especially for complex processor architectures employing the instruction pipeline, the fault generator circuit must be highly accurate in terms of injection time and location. In addition, the injection process should be carried out as fast as possible to avoid synchronization violations [65].

In this thesis, the main focus is on the clock glitching attack approach. Figure 2.5 shows a

typical circuit employed by attackers to conduct a fault injection attack. A clock glitch generator generally consists of different hardware components such as FPGA boards, pulse generators, and micro-controllers to enable the implementation of various glitch generating methods. The desired configuration is provided by a controller PC connected to both target and generator sides. Synchronization is done by using the trigger signal between the target and the generator.

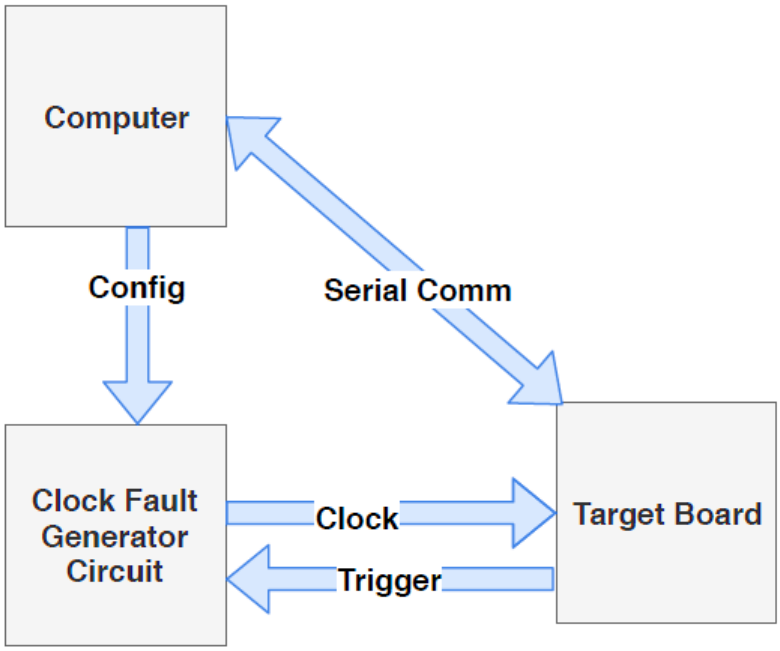

Figure 2.5: Experimental Setup of Clock Fault Attack

2.2.2 Clock Glitching Attack Examples

The clock glitching attack can significantly impact the critical parts of a running application, such as its encryption module and arithmetic or logical instructions. In several works, a clock glitching attack has targeted a specific round or operation of the AES algorithm to generate faulty cipher text [38, 44, 52]. These faulty outputs can be used to recover the encryption key. Other than AES, other cipher blocks, including DES, Camellia, CAST-128, SEED, and MISTY1, have been evaluated against the clock glitching in [58]. Five commercial and low-cost processors have been targeted by clock glitch injection in [52] to conduct a deep analysis of fault impacts.

2.2.3 Clock Glitch Generator Characteristics

As discussed in previous sections, fault generators, in general, should have specific characteristics, namely accuracy, run-time configuration, reproducibility, and randomness. In this sub-section, the clock glitch generator characteristics are explained. Below is a list of the main features that are

identified, and then for each category, the related articles are compared.

- **Clock Glitch Accuracy:** This feature includes the parameters such as minimum glitch width, gliding shift steps, standard deviation, clock glitching placement, and clock glitch frequency control. These features depend on the characteristics of the clock generator and the method which is used to induce the glitches.
  - **Minimum glitch width:** Most target systems contain protocols to protect their external clock input so that when an abnormal clock is observed and detected, it resets the system or produces an alert. Therefore, less glitch width results in a lower chance of detection. In addition, in many cases, a precise and short glitch is needed to target a specific instruction running on the processor or specific data being fetched. This makes it important to control the glitch width. The precision of the clock glitch width is on the time scale of picoseconds or a few nanoseconds. For example, in [58], a 2-channel pulse generator has been used to design an accurate 35ps glitch. [38,57] and [65] include glitches on the nanosecond scale.
  - **Gliding shift steps:** this parameter refers to the minimum value by which a clock glitch generator adjusts the glitch width. These steps are on the nanosecond time scale. For example, in [52,56], the gliding shift steps are reported as 1ns.
  - **Standard deviation:** The standard deviation of the reported glitch width depends on the glitch generation algorithm and the limitations of the hardware used (e.g., FPGA).
  - **Clock glitching placement:** It is important to induce a glitch into a specific position of the clock signal. For example, it is important to inject a glitch into a system at a certain round of encryption. [66] uses a counter to determine the location of the glitch injection.
- **Clock Glitch Run-Time Configuration:** faulty clock parameters, such as phase delay and frequency, can be reconfigured at the run-time to help the attacker. For example, in [57], a digital clock manager (DCM) of an FPGA is used to configure the glitch. FPGA run-time adjustment can be used to change the phase-shifted values, but this method is limited to a certain range of values. [51] presents an approach for partial configuration using the FPGA configuration capabilities, including reconfiguring the DCM blocks at the run-time. The "bitstream differential files" generated by FPGA tools are applied to substantiate the run-time configuration.
- **Clock Glitch Reproducibility:** this helps attackers to reproduce the faulty clock signal with the same characteristics as many times as they wish. It is important in some applications, which require the assessment of vulnerabilities and tracking of the clock glitch attack procedure. [58] includes a unique evaluation platform (SASEBO) for reproducible clock glitch generation.

- **Clock Glitch Randomness:** In most clock glitch generators, the glitch parameters such as frequency, width, precision, etc., are pre-determined, i.e., they are determined before run-time. This requires complete and adequate information about the target system, which would not be possible in all real-world attack scenarios. Furthermore, in order to have a highly efficient attack, one needs to cover a large number of possible combinations of the parameters mentioned, which would not be practical in a limited time. For this reason, in recent years, new techniques such as the fuzzy glitch generator [54] have been proposed for the generation of clock glitches. All of these techniques use ring oscillators instead of FPGA blocks and demonstrate the ability to achieve a highly successful attack rate. Details of the related works will be provided in the following section.

2.3 VOLTAGE-BASED FIAS

Voltage fault injection is another practical and low-cost attack that adversaries can use to conduct efficient attacks against targeted systems. They are used to attack embedded systems with one or multiple external power supply inputs. The fundamental concepts and related characteristics for this technique are provided in the following sub-sections.

2.3.1 CONCEPTS OF VOLTAGE FIAS

An important class of voltage glitch attack is manipulation of supply voltage [33, 56, 61, 62, 63, 67]. This is specifically applied in scenarios where the target system is fed from an external power supply [33]. Similar to clock fault attacks, a voltage fault attack is an approach that does not require extensive equipment or knowledge. Moreover, it can be used when access to the external clock input is not available on the target systems, i.e., those using their own internal clocks.

The non-equality in equation (2.1) points out the condition for correct circuit operation and correct value storage in memory. A standard approach to produce a timing constraint violation is to manipulate the clock period. The other approach is to increase the data propagation delay by tampering with the input voltage. In these approaches, equation (2.2) will not fulfill the timing condition for the correct circuit operation , which may result in erroneous data captured by memory cells and/or flip-flops. [68] discusses the relation between the input voltage and data propagation delay with a simple CMOS inverter as an example. Figure 2.6 shows that there are two propagation delays, named T_{pHL} and T_{pLH}, for the output variations from high to low and vice versa.

Equation 2.2 shows that these two delays depend on different factors, including the supply voltage. In this equation, V_{DD} is the power supply voltage, C_L is load capacitance, $V_{th,p}$ is PMOS threshold voltage, u_p is holes mobility, C_{ox} is gate oxide capacitance, and (W_p/L_p) is the aspect ratio of the PMOS. By replacing the parameters related to the inverter's PMOS by NMOS parameters (e.g. u_n, (W_n/L_n), $V_{th,n}$), T_{pHL} can also be derived [68, 69]. Equation 2.2 demonstrates that T_{pHL}

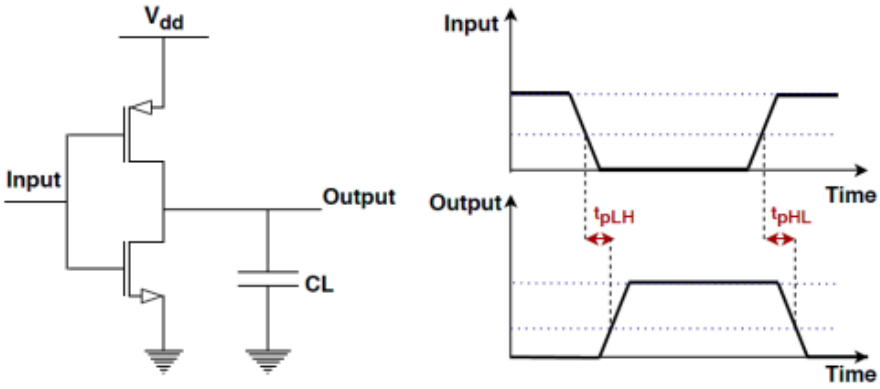

Figure 2.6: Inverter Circuit and T_{PHL}, T_{PLH} Parameters in Response Waveforms

and T_{pLH} have a direct relation with v_{DD}. In particular, reducing the power supply will increase T_{phl} and T_{plh} of the inverter. Therefore, manipulating the supply voltage (in long or short intervals) can be an efficient approach to inject a fault in order to violate the timing constraints [33, 70].

$$T_{pHL} = \frac{C_L([\frac{2V_{thp}}{V_{DD}-|V_{thp}|}] + ln(3-4\frac{|V_{thp}|}{V_{DD}})}{u_P C_{ox}\frac{W_P}{L_P}(V_{DD}-|V_{thp}|)} \tag{2.2}$$

There are two main types of voltage attacks: 1) Underpowering and 2) Voltage glitching [70]. Underpowering is a type of voltage manipulation in which the target's supply voltage is applied permanently (or for a relatively long period) to a voltage source out of its nominal range (the standard voltages defined by the manufacturer). Voltage glitching is a sudden and momentary change in the supply voltage. It has been shown that under-powering is not sufficient to achieve a high success rate because it impacts multiple instructions that are executed [36].

In Voltage glitching, the attacker attempts to feed the target with a power supply below or above the considered nominal value(V_{DD}), for a certain period. Voltage glitching is generally used to induce a fault via timing violation; however, an attacker needs to have high control over the attack procedure and precision. This sudden (usually negative) change of voltage, which is defined by V_{diff}, can induce transient faults. In this regard, a multiplexer can be used to switch the voltage between the two V_{DD} and $V_{DD} - V_{diff}$ values [63]. The value for V_{diff} strongly depends on the level of the protection mechanisms (e.g., glitch detectors) that exists in the target system. If it goes beyond a certain value, such unusual behavior may trigger a warning regarding target system functionality and reliability, and usually leads to a device reset (to prevent damage in the IC, burning of electronic components, etc.). Typically, injected voltage glitches are generated as a square or V-shaped signal [33]. Figure 2.7 depicts an example of a negative voltage glitch. However, the amplitude of such a glitch can be positive (i.e. above the nominal voltage value) [70].

The shape of the glitch is defined by the rising and falling time (t_f and t_r) and the pulse width (t_p). The sum thereof represents the total duration of the glitch, named T_{glitch} [71]. T_{glitch} is also referred to as the Attack window. This window is located between the falling and rising edges for

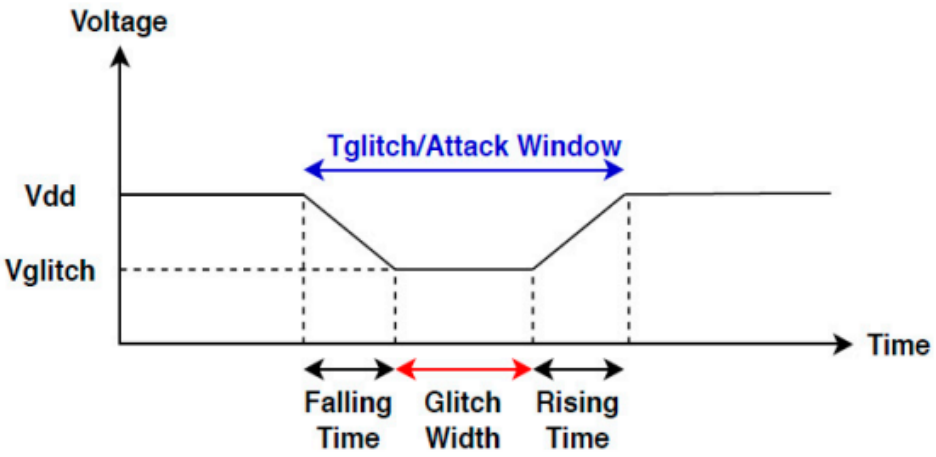

Figure 2.7: Negative Supply Voltage Glitch

negative glitch inducing and vice versa for positive. V_{glitch} (equal to V_{dd} - V_{diff}) is the voltage level that is fed at the target glitching time.

Table 2.2 compares the accuracy of under-powering and voltage glitching attacks.

Table 2.2: Supply Voltage Fault Injection Techniques and Characteristics

Technique	Spatial precision	Temporal precision	Controlling the intensity	Equipment costs	Required expertise
Under Powering	Moderate	Not Applicable	Voltage amplitude	Low	Low
Voltage Glitching	Moderate	Moderately synchronization	Voltage amplitude glitch width or delay	Low	Moderate

All the column headings are the same as those in Table 2.1. The first column compares the spatial accuracy of under-powering and voltage glitching. The advantage of voltage fault injection methods as compared to clock fault injection methods is higher spatial accuracy (moderate level). The spatial precision of voltage fault injection is due to the presence of multiple power islands in modern systems. These systems typically operate in multiple power islands in accordance with their performance requirements. However, the temporal precision of voltage glitching (column 2) is less than that of clock glitching because in the later one can synchronize the injected fault in respect to the specific cycle accurately. In addition, the precision of the fault injection process in inducing a fault at a specific time is not applicable to the under-powering and overclocking approaches. The third column compares the attacker's level of control over the voltage glitch parameters, such as glitch delay and width in voltage fault injection methods. The same level of control is not attainable for under-powering attacks. Finally, the last two columns show equipment costs and the required level of expertise for the two methods, which are very similar.

This thesis considers the voltage glitching attack approach from the voltage FIA techniques

based on the facts above. Figure 2.8 shows a common setup employed by attackers to conduct successful voltage glitching attacks. A voltage glitch generator generally consists of different hardware components such as power sources, FPGA boards, and high-speed multiplexers to enable the desired glitch shape(s) in the generated voltage signal. Similar to the clock fault injection setup, a controller PC is connected to both target and generator sides to determine and configure the parameters of the attack. Synchronization is done by using the trigger signal between the target and the generator. The glitch may be injected by using a trigger signal from the general-purpose I/O pins and set up by running the program on the target. In the next sub-section, the main characteristics of voltage glitch generators are classified.

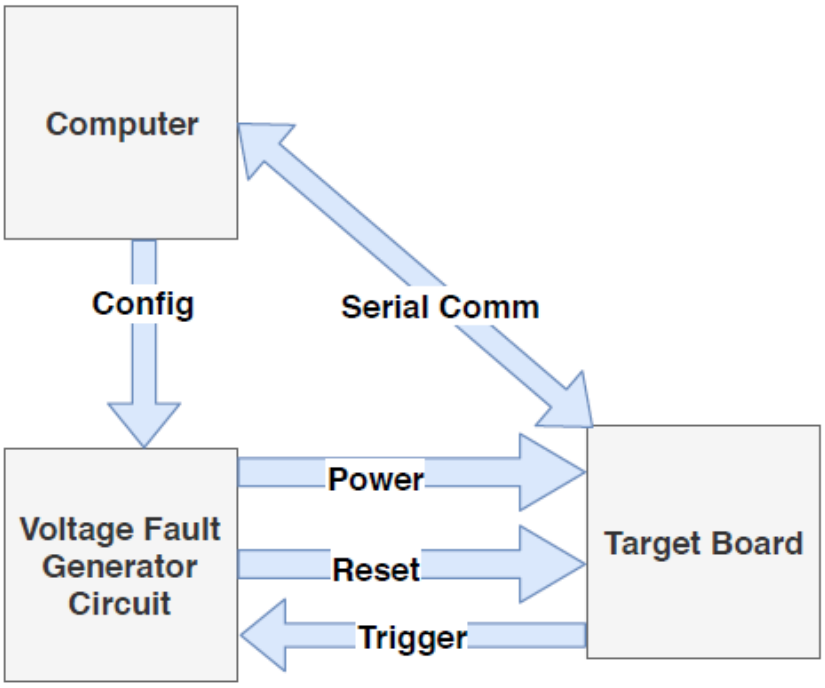

Figure 2.8: Fault Injection Setup for Voltage Fault Attack

2.3.2 VOLTAGE GLITCHING ATTACK EXAMPLES

This type of attack has been employed against various critical modules of embedded applications. One of the most critical examples is the encryption module, such as the AES algorithm which has been evaluated against voltage fault injection attacks. In this algorithm, applying voltage fault injection can result in a reduction of the number of encryption rounds [32]. The faulty cipher texts can then be leveraged to reveal the encryption key. Moreover, it has also been used by adversaries to induce faults and bypass different security features such as the authentication or secure boot of the embedded systems. The induced fault can help the attacker to load his/her arbitrary values in the PC (Program counter) register, which is very dangerous for secure systems [71]. Finally, [62]

has shown that the voltage glitching attack can lead to privilege escalation from the user to kernel mode in the Linux OS.

2.3.3 VOLTAGE GLITCH GENERATOR CHARACTERISTICS

Different characteristics, such as accuracy, run-time configuration, reproducibility, and randomness, are important in successful fault injection attacks. In this sub-section, these characteristics for the voltage glitch generators are discussed. There are many similarities between voltage and clock glitch generator characteristics. Here each voltage glitch generation feature is studied in more detail, and later the works related to these features are compared in Section 2.5.

- Voltage Glitch Accuracy: Various parameters can affect the accuracy of the generated voltage glitch, such as minimum glitch width, glitch delay, and glitch placement. These features are dependent on the characteristics of the voltage generator and also the method which is used to induce the glitches.
  - Minimum glitch width: Generally, a precise and short voltage glitch is needed to bypass the existing glitch detectors on the external voltage inputs of the targeted systems. These protection modules observe the external voltage inputs and can reset the target system or create an alert to avoid any malfunction. Furthermore, a short voltage glitch is needed to target a specific instruction of the running code on the processor. Therefore, the more accurate and tenuous the voltage glitch is, the more chances an attacker has to accomplish the desired fault attacks. Depending on the equipment employed to generate the faulty signal, the minimum value of the glitch width can be changed. For instance, in the generator presented in [70], a pulse generator is used to generate the optimized voltage glitch, and the minimum pulse width is reported as 10ns.
  - Glitch delay: This parameter shows the amount of time a voltage glitch takes to be injected after the setting up of the trigger signal [62]. To take advantage of the existing vulnerabilities in the running application, the glitch must be induced simultaneously with specific instruction(s) execution. According to the selected glitch width, the glitch delay should be considered to guarantee that the injected glitch hits the targeted instruction. For instance, to generate faulty cipher texts, it is important to inject a glitch into the target at a certain round of encryption [72,73]. The glitch injection time is decisive in other situations such as escalating privileges [62], skipping authentications, and misinterpreting an instruction [71].
  - Voltage glitch placement: this parameter is related to the spatial position of the glitch. Spatial precision specifies the exact locations of the targeted circuit affected by the faulty voltage signal. In advanced systems such as modern SoCs, there are different power domains or power islands. A power domain is a group of gates powered by the

same supplier. Depending on operational conditions, different voltage suppliers are connected to these domains. This parameter can help the attacker to provoke an erroneous behaviour in a specific part. For example, by targeting an area in the chip like register banks and inducing memory faults, the wrong values may be gathered from the memory bus [32].

- Voltage Glitch Run-Time Configuration: this parameter is defined as the possibility of reconfiguring the voltage glitch during the run-time. The parameters of generated glitchy voltage signals such as glitch width, delay, and amplitude can be reconfigured at the run-time to help the attacker to examine different attack scenarios. Run-time configurability depends on the specifications of the generator circuit. Reconfigurability can be applied through hardware inputs (e.g., by using sliding switches to adjust glitch width) or by communicating (e.g., UART) with the controller PC [51].

- Voltage Glitch Reproducibility: This feature helps the attacker to obtain the same voltage glitching attack results when the same glitch is injected into multiple runs. Typically, voltage glitch generations are reproducible when fixed values are identified for the glitch parameters.

- Voltage Glitch Randomness: Instead of pre-defining and pre-calculating all voltage glitch parameters in every attack scenario, an attacker can select them randomly from an interval of possible values. Using this approach, more combinations of the parameters mentioned can be covered within a short time, leading to more successful attacks. With some of the previously proposed approaches, such as [62], glitch width and glitch delay parameters are defined using a divide and conquer method with a randomized parameter approach. Now that the important parameters of glitches are categorized, some of the main works in the field are reviewed in the next section.

2.4 A Review of Fault Generators

In this section, some state-of-the-art platforms for clock and voltage fault injectors are reviewed. Then, these works are classified based on the defined parameters in the previous section. Then these parameters are compared in terms of operational constraints, accuracy in the generated signal, and impacts on the target system. Table 2.3 and 2.4 shows the details of our comparison.

2.4.1 Clock Glitch Generators in the literature

There are various practical techniques for generating a faulty clock signal [38, 51, 52, 53, 54, 55, 56, 57, 58, 64, 74, 75]. All of the proposed methods call for a high-frequency clock signal, the so-called "Nominal Clock" on the clock fault generator side. Nominal Clock generation can be done via several methods, including ring oscillator, Phase-Locked Loop (PLL), voltage oscillator coupled

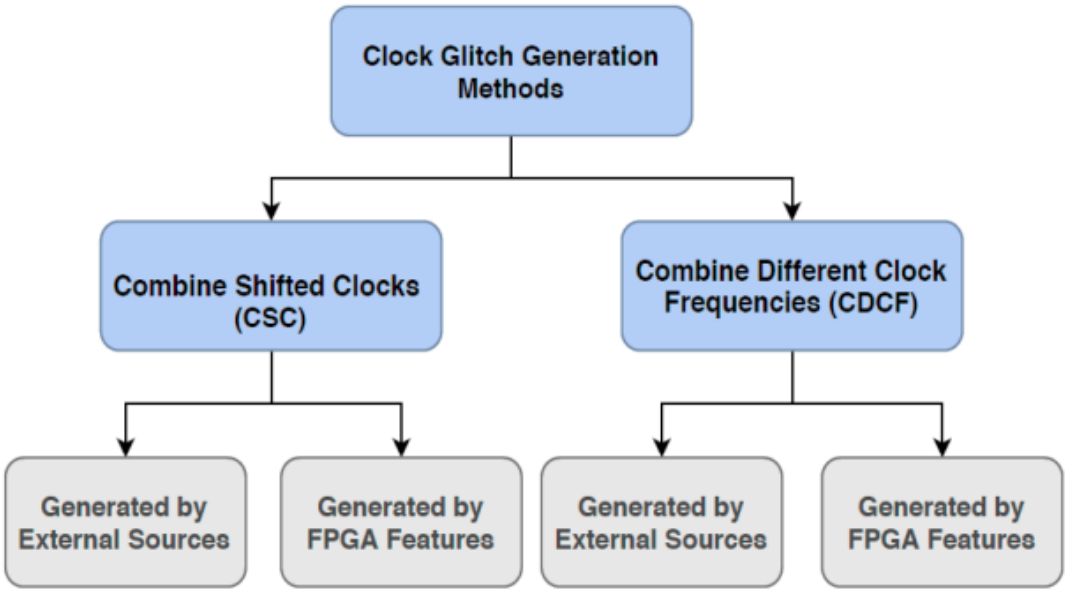

Figure 2.9: Different Methods for Generating Clock Glitch(es)

with PLL and clock frequency circuit, or crystal oscillators. Figure 2.9 summarizes the different solutions to generate a single glitch or even multiple glitches in the nominal clock signal. In this figure, the related works are divided into two broad categories: 1) Combine Shifted Clocks (CSC), 2) Combine Different Clock Frequencies (CDCF). Below, the works in each category are introduced. It should be noted that all previous works assume that the external clock of the target system is accessible.

2.4.1.1 Combine Shifted Clocks (CSC)

Combine Shifted Clocks (CSC) is based on multiplexing two different clock signals with the same frequencies and different phases at the appropriate time [55]. Figure 2.10 shows the point of multiplexing which is defined by the trigger signal ($Clock-Delayed1$ and $Clock-Delayed2$) [51,53, 56, 58]. In CSC, the glitch width and glitch period are controlled by the trigger and shifted clock signals.

In order to have higher accuracy and control over the glitch generation, [52] presents a method in which the output clock is created by multiplexing between three signals (rather than two) with equal frequencies and different phases. There are mainly two implementation groups to produce the shifted versions of the nominal clock signal and to combine them. The first method is based on using external clock sources with phase-controlling capability. Figure 2.11 shows an example that combines the two external clocks using a 2-to-1 multiplexer from a two-channel pulse generator [58]. In the proposed glitch generator, the trigger signal is set to be synchronous with the execution of the running code on the target. To this end, an oscilloscope detects the positive edge of the target's execution signal and then sends out the trigger signal with a constant delay.

The second method utilizes internal FPGA features such as the embedded DLL (Delayed Locked

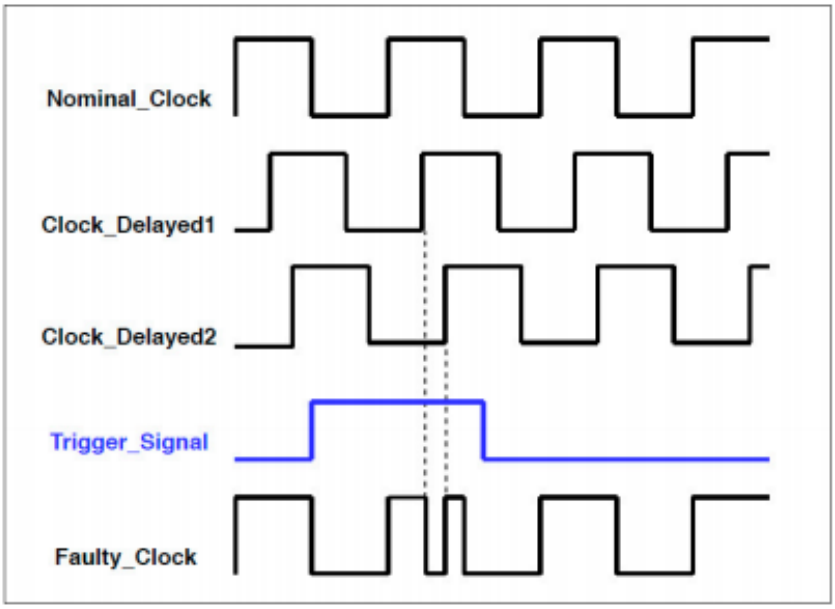

Figure 2.10: Faulty Clock Generation Using Two Shifted Signals

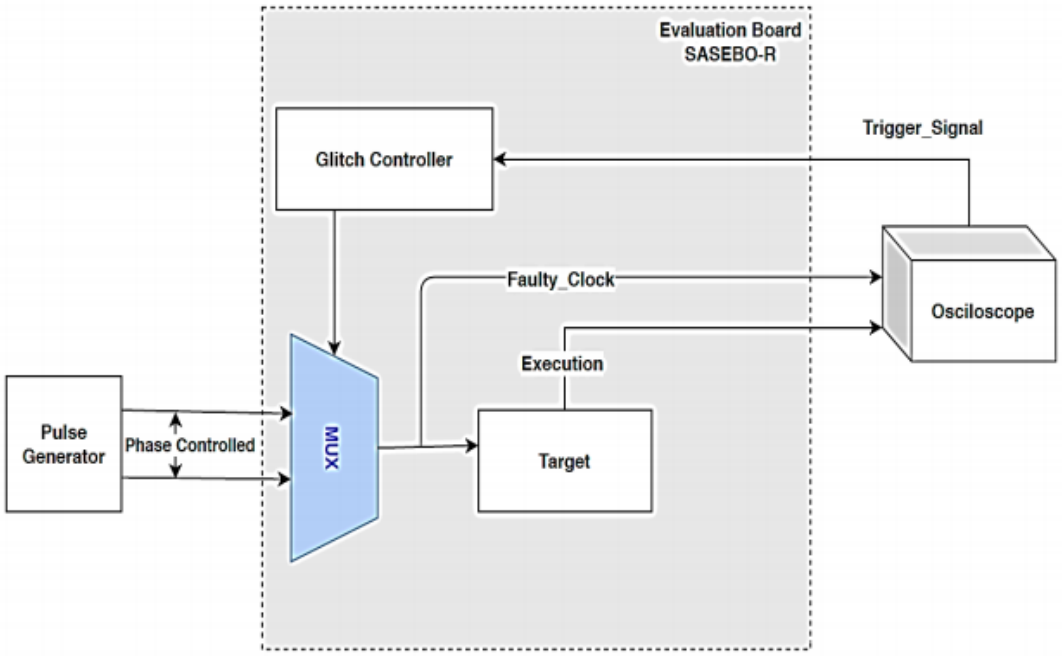

Figure 2.11: Experimental Environment

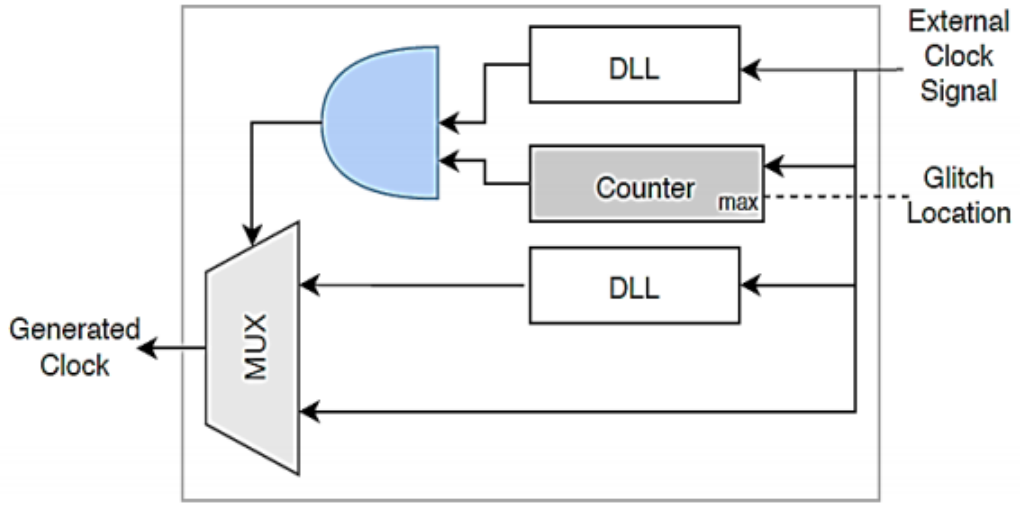

Figure 2.12: Glitch Generator in [57]

Loop) to generate the shift [51, 53, 55, 57]. In this method, the precision of the glitch is closely related to the smallest elementary delay of the DLL. Figure 2.12 illustrates a method in which two DLLs and a counter are used to control the glitch delay [57]. Compared to [55], this method can change the glitch period in just one cycle and can induce glitches into any cycle (higher configurability). Note that the frequencies of the generated clocks are limited to the DLL circuit of FPGA. To solve this issue, [57] used dividers to output the desired frequency.

2.4.1.2 Combine Different Clock Frequencies (CDCF)

Combine Different Clock Frequencies (CDCF) is based on multiplexing the nominal clock with a high-frequency clock of the same phase whose period is T_n and T_g, respectively [38, 52, 74]. Figure 2.13 demonstrates that a trigger signal is used for timely multiplexing between two clock signals in the CDCF-based method.

There are mainly two implementation approaches for CDCF. The first group is based on generating the desired clock signals using external sources [38, 54]. For example, [38] uses a wave generator to produce the nominal clock and the high-frequency clock. However, this approach cannot provide an acceptable level of randomness in clock glitching scenarios. [54] introduces an approach that includes randomness in the glitch generation procedure named Fuzzy glitching. This approach is based on the use of adjustable ring oscillators at various frequencies. In this work, instead of producing exactly timed glitches, random glitches are made by using ring oscillators. Figure 2.14 shows an application of fuzzy glitching where a multiplexer is used to inject the glitches into the system for a limited time.The second approach is based on generating the clock signals with different frequencies by using the interior FPGA features such as Phase Locked Loop (PLL). As an example, in [74], a precise and accurate external clock signal passes through the on-chip PLL to provide the appropriate nominal clock. The unchanged version of that external clock signal is used

to produce the faster clock. Multiplexing between these signals can be performed via Mixed Mode Clock Management (MMCM).

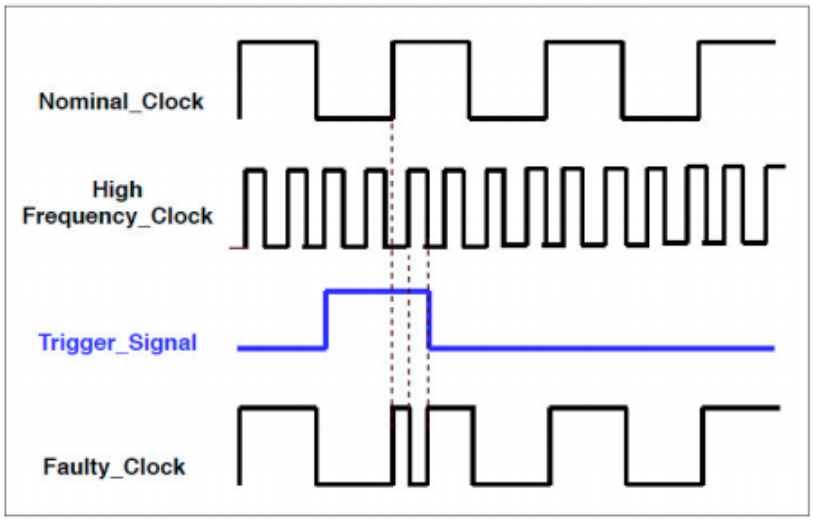

Figure 2.13: Glitch Generation Using High-Frequency Signal

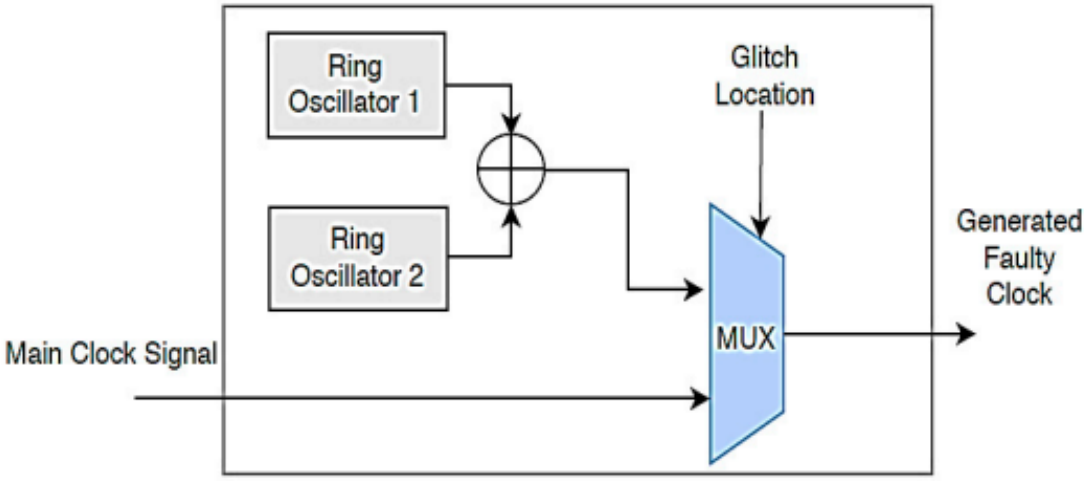

Figure 2.14: Glitch Insertion Circuitry Using Two Ring Oscillators [54]

Table 2.3 summarizes the technical attributes of the proposed CSC-based and CDCF-based clock glitch generators. Table 3 shows that in both CSC and CDCF, clock glitches are either generated using internal features of FPGAs or by external clock sources. FPGAs are the major platform for clock glitching experiments [51, 52, 53, 54, 55, 58]. A waveform generator is also a common tool used in CDCF to generate a high-frequency clock signal. Below, all of the parameters reported in Table 2.3 are elaborated.

Table 2.3 compares different approaches in terms of complexity and cost. It shows that the FPGA-based approaches are easy to implement and very cost-effective [51, 52, 55]. There are generators that require custom board design and need to be produced by more skilled engineers and perhaps at a higher cost. Moreover, there are specific evaluation boards such as SASEBO and Chip whisperer, which are semi-configurable and have a moderate cost [51, 59, 76].

Table 2.3 reports the minimum glitch width values and studies the fault clock cycle location among different approaches. It is observed that [51] from the CSC category and [38] from the CDCF category provide the most precise glitches among different glitch generators. It also shows that all the previous works, except the fuzzy generator [54], have the capability to inject glitches in the chosen clock cycle [51, 53, 55, 58]. Another important factor for fault generators is the run-time configuration. Table 2.3 shows the limitation of this factor in FPGA-based generators [52, 53, 55, 58]. The design in [51] uses a more advanced Xilinx board and can, therefore, support more run-time configuration parameters such as glitch delay and glitch width. Finally, Table 2.3 analyses the clock glitching approaches with respect to reproducibility and frequency adjustability. All of the glitch generators, except [54], can reproduce any faulty clock signal with exactly the same characteristics. Instead, the capability of providing random faulty clock signals is only considered in [54]. Clock frequency adjustments are also possible for all of the reviewed generators. However, FPGA's clock management units create another limitation related to the generated clock maximum frequency [51, 53, 55, 56]. This setting is less restricted using external clock resources [38].

Table 2.3: Review of Previously Proposed Clock Glitch Generators

Characteristics \ Methods		Fault Attack By Tampering Clock Input								
		CSC						CDCF		
		Using FPGA Features					Using External Clock Sources	Using External Sources		Using FPGA Features
		[55]	[51]	[53]	[56]	[57]	[58]	[38]	[54]	[52]
Evaluation Platform	Equipment	Xilinx Virtex5 FPGA	Spartan6 FPGA	Spartan6 FPGA	Spartan6 FPGA	FPGA on SASEBO-G VirtexII-Pro XC2VP30	Agilent 11152 Pulse generator and SASEBO-G	Agilent E4438C 6GHz Waveform generator DE2-115 development board.	Ring Oscillators on Spartan-3E FPGA	Virtex-II Pro XC2VP30 FPGA
	System Complexity	Moderate	Moderate	Moderate-High (PCB designing)	Moderate-High (PCB designing)	Moderate-High (evaluation platform)	Moderate-High (evaluation platform)	Moderate	Moderate-High	Moderate
	Cost	Moderate	Moderate- High *	Moderate	Moderate	Moderate- High * (evaluation platform)	Moderate- High * (evaluation platform)	Moderate	Moderate	Moderate
Minimum Glitch Width		No Info. (100Mhz)	3ns for (10MHZ)	7ns for (10 and 20MHZ)	5.9ns for (24Mhz)	4.9ns (24Mhz)	5.6ns (24Mhz)	8ns in (114Mhz) and 2ns (2GHz)	100 ns for 8mhz	27ns (8mhz)

Application	Target Platform	Desired Randomness	Reproducibility of Glitchy Clock	Capability to Control Generated Faulty Clock Frequency	Capability to Perform Run-Time Configuration	Capability to Inject Glitch into Specific Clock Cycle
128-bit AES	Xilinx Spartan 3AN fpga	No	Yes	Yes but Limited to DLL	No	Yes
128-bit AES and authentication	AVR,XMEGA,smart cart	No	Yes	Yes but Limited to DLL	Partial reconfiguration	Yes
Arithmetic instructions	AVR atmega 162	No	Yes	Yes but Limited to DLL	No	Yes
arithmetical/logical instructions, branch instruction, and memory instructions	ARM Cortex-M0, ATxmega 256	No	Yes	Yes but Limited to DLL	No	Yes
AES	SASEBO-G VirtexII-Pro XC2VP7)	No	Yes	Yes	No	Yes
AES	SASEBO-R	No	Yes	Yes	No	Yes
AES	Altera DE2-115 EP4CE115F29C7	No	Yes	Yes	Yes	Yes
Infinite-loop	STM32F030	Yes	No	Yes	No	No
Smartcard application SOSS	ATMEGA 163	No	Yes	Yes	No	Yes

2.4.2 VOLTAGE GLITCH GENERATORS IN THE LITERATURE

In this section, the existing voltage fault injector designs are reviewed. The parameters discussed before are used to compare them in terms of operational constraints, accuracy in the generated signal, and impacts on the target system. The comparison results are then presented in Table 2.4. The input voltage of a target system can be manipulated to perform fault injection if the system is connected to an external source. Reducing the input voltage increases the propagation delay and can result in timing violations [68]. Voltage fault injection can be performed either by permanently decreasing the voltage supply (under-powering) [63, 77] or by multiplexing between different voltage levels for a limited time (Voltage glitching) [62, 68, 70, 71, 72, 78].

For the under-powering method, adjusted and pre-planned parameters are not required. Instead, in order for an attack to be performed, the target simply needs to be connected to a constant voltage below the nominal voltage. However, this value depends on the target system specifications and occasionally calls for many trials and errors in order to be found [72]. For example, to perform differential fault attacks (DFA) using this approach, the attacker must find a particular voltage at which the faulty output would appear. Furthermore, the applied voltage should be above a threshold value at which the target system is functional, and communications to the controller computer must operate in order to collect the faulty outputs [63, 68, 77].

Although under-powering is a low-cost and simple method to implement, it does not provide the required time and location precision in many attack scenarios [60]. For more effective attacks, voltage glitch fault injection is employed. To obtain the desired voltage signal and to induce faults in the target systems accurately, different implementations are employed. Figure 2.15 summarizes the main techniques for voltage glitching and the classification of the related works into three categories: 1) Voltage Multiplexing (VM), 2) Voltage Short-Circuiting (VSC), and 3) Voltage Glitching using Pulse Generators (VGPG). It should be noted that for a more effective attack, some changes may be made to the experimental set-up before voltage fault injection is performed. For instance, the main power supply from the target chip should be disconnected. This calls for the removal of the corresponding resistor bridge, which helps to control the voltage supply of the target system. Moreover, all existing bypass capacitors on the target PCB should be detached since their role is to enhance stability, and they are not needed to carry out the attack process. In the following, the works in different categories are introduced.

2.4.2.1 Voltage Multiplexing (VM)

VM is based on the use of a high-speed multiplexer to switch between Vdd and a lower voltage value during trigger Signal activation [60, 79]. In [60], one high-speed multiplexer on an FPGA is used. The multiplexer includes two adjustable inputs, and its output is connected to the target system in order to switch between different voltage levels (Vdd and 0 volts). Different specifications must be considered to select a proper multiplexer for the fault injection set-up, including input voltage

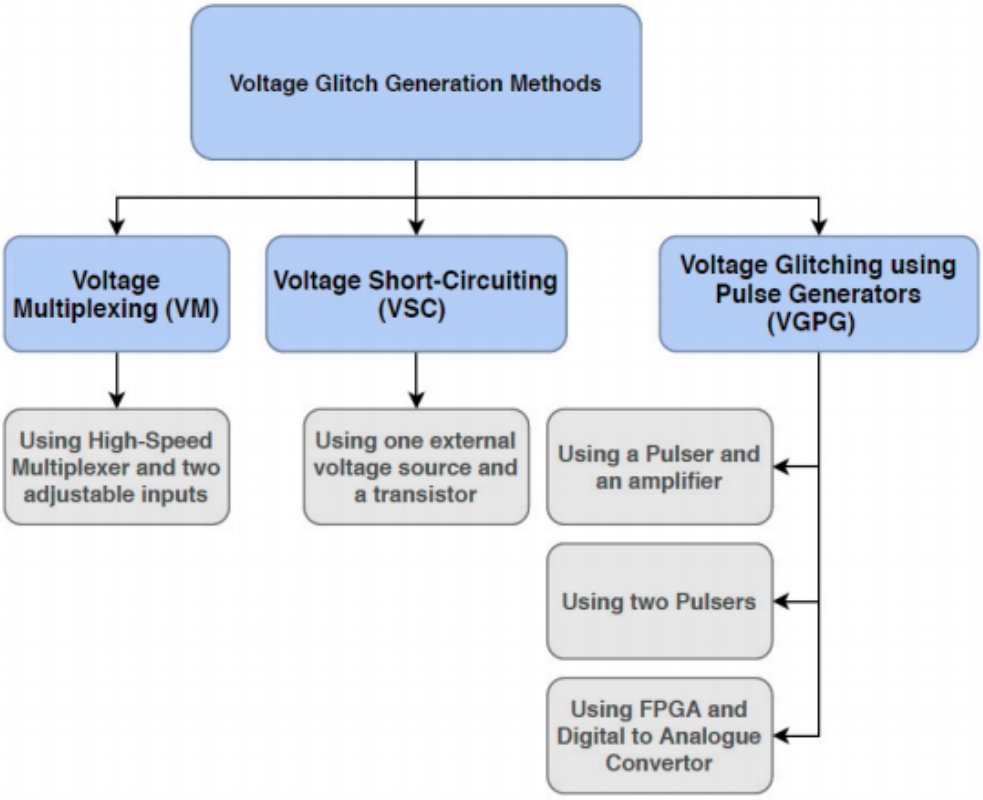

Figure 2.15: Different Methods for Generating Voltage Glitches

limits, switching speed, output leakage, existing capacitance, and charge injection. Depending on the required voltage ranges on the output, different multiplexers are designed and optimized. Consequently, the proper multiplexer should be used in order to avoid performance degradation.

2.4.2.2 Voltage Short-Circuiting (VSC)

VSC is one of the most commonly-used fault injection approaches for generating one or multiple glitches in the voltage signal. Figure 2.16 presents the related set-up in which a transistor is placed in parallel to the power supply line in order to create a short-circuit between the Vcc and the ground [51]. In this method, additional equipment is required to control the voltage levels and the timing of glitch induction. The generated glitch can be erratic due to process variations and the target system's electronic properties. [33] has shown the oscillations in the generated voltage signal by using the VSC set-up and has proposed a method for a more precise and optimized voltage glitch shape.

2.4.2.3 Voltage Glitching using Pulse Generators (VGPG)

The idea behind VGPG is to utilize a pulse generator as a DC voltage source to generate and inject arbitrary glitches (usually square shapes) into the supply path [33, 68, 70, 71, 78]. The pulse generator can produce the equivalent analog signal from a digital source [33]. A logic level change in the

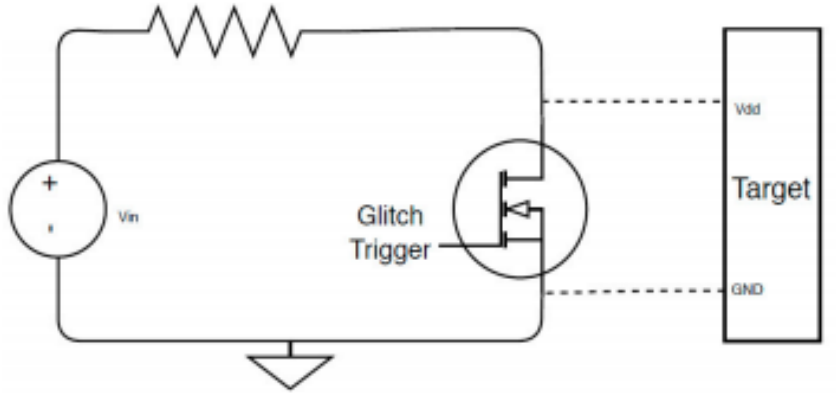

Figure 2.16: Voltage Generator Set-Up in [51]

external input trigger leads to a transition from the constant supply voltage to the glitch waveform loaded in the generator. In [33], a pulse generator is used to generate arbitrary wave forms in which a set of parameters can be defined at the software level and saved in the internal memory of the pulse generator. [68] presents a solution for using two pulse generators (instead of one) to achieve higher precision. The second pulse generator can improve the generated glitch shape, and it brings the voltage value to the normal level at a higher rate.

Figure 2.17 depicts the expected and altered glitches with one and two generators in this work. A high-speed FPGA can also be used to generate the desired pulse. In [71], an FPGA and an amplifier are used to produce the faulty voltage signal. In this case, glitch parameters are easily configured by loading the bitstream file into the FPGA.

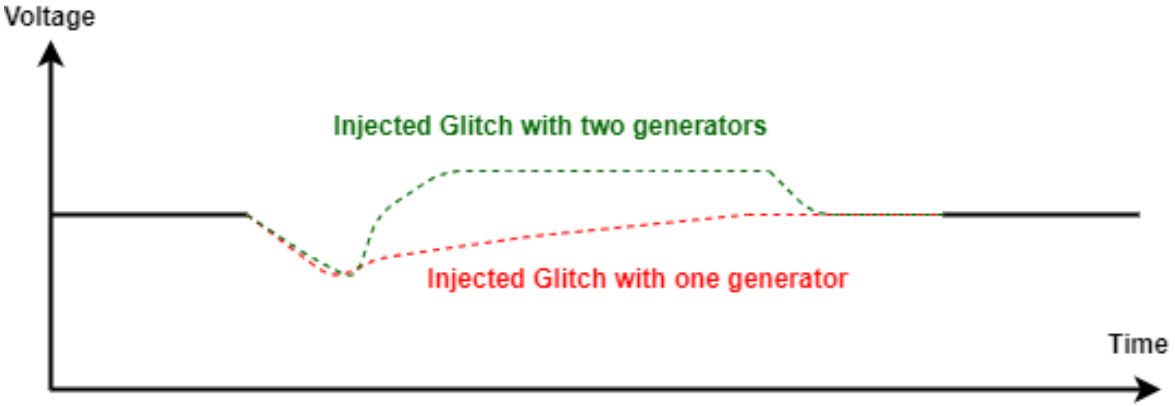

Figure 2.17: Comparing Generated Voltage Signal with One and Two Voltage Generators [68]

Table 2.4 reviews voltage glitch generators and summarizes their technical features. The table compares the VM, VSC, and VGPG approaches and presents their implementation platforms (FPGAs, transistors, and pulse generators) [33, 51, 60, 68, 70, 78, 79].

Various targets and applications are also listed in the table and have been examined with respect to voltage glitch generators. Complexity and cost are the first two parameters in our comparison shown in Table 2.4. It can be seen that most voltage glitching platforms can be implemented with moderate knowledge and expenses. The minimum glitch width is the next parameter to consider in different voltage glitch generators. It depends highly on the equipment used and its ability to shape the precise voltage glitch. The VM-based approach could ideally generate very short glitches by internal FPGAs; however, due to the limitations on I/O pins or other connectors used, higher values are obtained in practice [60, 79]. In the VGPG-based approach, the minimum glitch width is reliant on the capabilities of the pulse generator. For example, in [70], it has been reported as 10ns. The next main parameter of voltage glitch generators is run-time configuration capability, which is limited to the MUX used in VM-based generators [60, 79]. However, this setting is less restricted by the use of external voltage resources in VSC and VGPG approaches [33, 51, 68, 71, 78]. This table also shows that in all studied works, synchronization between the voltage glitch generator and the target is applicable by using a trigger signal.

As a final point, Table 4 analyzes the voltage glitching approaches concerning reproducibility. It shows that all of the glitch generators can reproduce any faulty voltage signal with the same characteristics. Note that these voltage generators cannot provide any random faulty voltage.

Table 2.4: Review of Previously Proposed Clock Glitch Generators

Characteristics \ Methods		Fault Attack by Tampering Voltage Input							
		VM		VSC	VGPG				
		Using FPGA Features		Using a Transistor	Using a Pulse Generator				
		[60]	[79]	[51]	[71]	[33]	[68]	[78]	[70]
Evaluation Platform	Equipment	XILINX Spartan-6 XC6SLX45	Standard Evaluation Board (SASEBO) that includes two Xilinx VirtexTM-II Pro devices	Chipwhisperer evaluation platform	Dedicated high-speed hardware	Arbitrary Waveform generator and glitch amplifier	Agilent 814A AND Picosecon d 10,3000B	PCB and a pulse generator	Agilent 8114A pulse generator
	System complexity	Moderate	High (evaluation platform) + RF measurements	Moderate-High (evaluation platform) And Open source	Moderate	Moderate-High (stm32 to controlling and I/O)	Moderate	Moderate	Moderate
	Cost	Moderate	Moderate-High (evaluation platform)	Moderate-High (evaluation platform)	Moderate	Moderate	Moderate	Moderate	Moderate
Minimum Glitch Width		-	-	-	-	-	-	-	10ns
Voltage Glitch Placement		Synchronous with Trigger	Synchronous with Trigger	Synchronous with Trigger	Synchronous with Trigger	Synchronous with Trigger	Synchronous with Trigger	Synchronous with Trigger	Synchronous with Trigger
Capability to Perform Run-Time Configuration		Limited to MUX (on the FPGA)	Limited to MUX (on the FPGA)	Limited to the voltage source	Limited to the FPGA	Limited to the voltage source	Limited to the voltage source	Limited to the voltage source	Limited to the voltage source

Reproducibility of Glitchy Clock	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Desired Randomness	No	No	No	No	No	No	No	No
Target	FPGA Xilinx Spartan3 and Spartan6	Tag Chip	AVR – STMF103	ARM	Different targets form ST, Texas instrument and Rennes	FPGA Xilinx Spartan 3A	Arduino Board	FPGA Xilinx Spartan3
Application	STRNG	Radio Frequency Identification (RFID) tags	Authentication/ boot loader	Secure boot attack/ secure runtime attack	Different Fault models	AES	The Ed25519 and EdDSA signature schemes	AES

2.5 CONCLUSION

Chapter 2 reviewed most of the state-of-the-art clock and voltage glitching platforms. It then defined their important characteristics and compared them based on these factors.The review points out these platforms are challenging to adapt for general IoT designs, and they lack the proper FIA configuration characteristics, such as clock/voltage glitch shape and injection timing parameters. An ideal platform makes it easy to set these variables for the software developers. Moreover, most of the proposed platforms were used to inject fault at lower levels (e.g., assembly or binary levels), making it difficult for the software developers to interpret the vulnerabilities.Therefore, they need an extra work to first configure the platform, then identify the most important/ vulnerable points, and then to analyze the fault effects. Respectively, this thesis benefits from the reviewed and already existing features to design an efficient fault injection platform based on clock glitching. This platform will be explained in next chapter. .

3 HARDWARE SECURITY EVALUATION PLATFORM

Contents

This chapter presents a practical hardware FIA platform designed to evaluate an embedded software against clock glitching attacks. This platform consists of various components, including 1)Fault Configuration Interface, 2)Fault Generator, and 3)Fault Effects Analyzer. Each of these units is presented with their related characteristics. Nevertheless, this chapter mainly focuses on the Fault Generator part. Accordingly, the implementations of two different clock glitch generators are first presented. Then, the capabilities and accuracies of these two clock glitch generators are compared. These two generators are experimentally evaluated by applying a clock glitching attack against an encryption algorithm. Finally, a simplified high-level fault effects analysis approach is proposed.

3.1 THE FRAMEWORK OF A PRACTICAL EVALUATION PLATFORM

This section presents the framework for a practical evaluation platform against the clock glitching FIA. The characteristics of this platform have been defined according to the studied works in Chapter 2. Figure 3.1 illustrates the high-level diagram of the proposed platform, which consists of

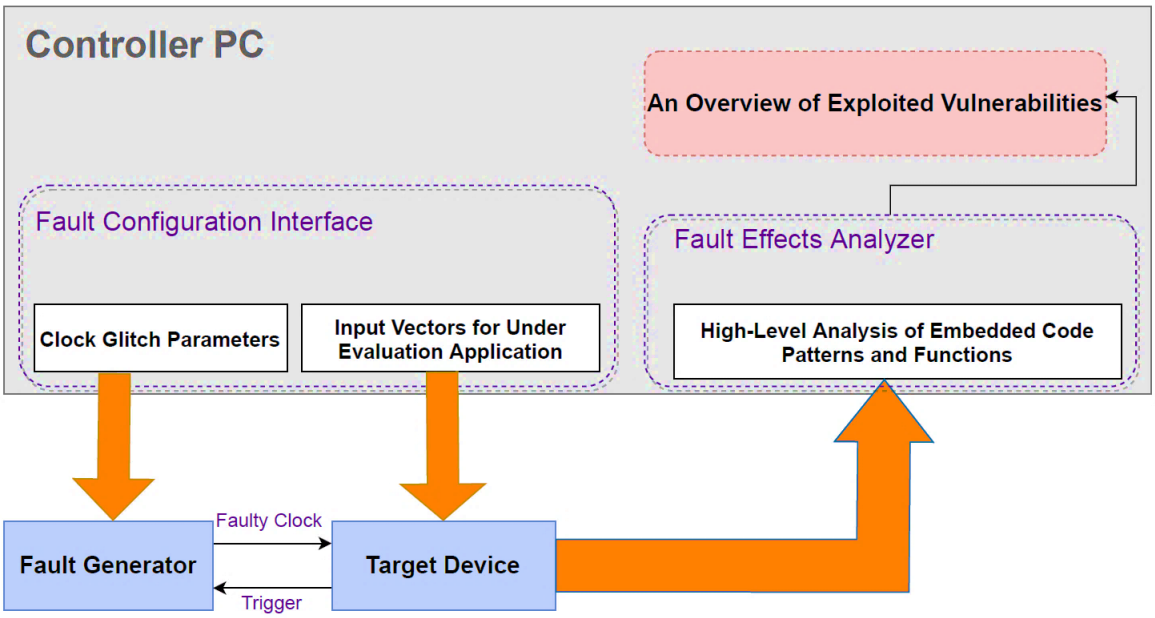

Figure 3.1: The architecture of the proposed evaluation platform

three main components: 1) Fault Configuration Interface, 2) Fault Generator, and 3) Fault Effects Analyzer. The configuration and the analyzer are accessible through a Controller PC. The following process details the way that the platform is used to perform a fault injection

- Via the configuration interface the user adjusts the fault generator with proper parameters and initializes the target processor.
- The fault generator then sends a ready signal to the configurator interface when it is completely programmed.
- The configuration interface sends the proper inputs to the target under attack and starts its execution.
- The target processor sets the synchronization signals (e.g., a trigger signal) to reach the predefined point in its execution flow.
- The fault generator generates the faulty signals with the parameters specified in step (1) and applies them to the corresponding target.
- The analyzer examines the individual attack results and gives an overview of vulnerabilities that the evaluator can use (e.g., software developer or hardware security specialists). It depicts the prone to attack sections of the system, which the developer can consider improving.

In the following sub-sections, each unit is studied in more detail, and its characteristics are explained.

3.2 FAULT CONFIGURATOR INTERFACE

To increase a platform's evaluation performance and to build practical clock glitching attack scenarios, a configurator interface is required. This can help the user (evaluator here) to control and adjust the faulty signal parameters. This configurator can automatically or manually generate the fault parameters, modify them, and send them to the fault generator circuitry. In the following, the important configuration parameters for clock glitching FIAs are explained. Thereafter, a search strategy is introduced to gather the configuration sets that lead to a successful fault attack.

3.2.1 KEY PARAMETERS FOR CLOCK GLITCH CONFIGURATION

In this thesis, the focus is on the clock glitching FIA. A clock glitch will temporarily shorten the clock cycle period from T_{clk} to T_{glitch}, cause timing violations and faulty outputs or malfunctions in the target processor. There are multiple clock glitch parameters that must be tuned, such as:

- Glitch Delay: This parameter shows where to insert the glitch after the positive edge of a clock cycle.

- Glitch Width: This parameter describes the width from the point indicated by Glitch Delay to the right.

- Glitch Temporal Location: This variable shows the clock cycle (i.e., number of cycles) to insert the glitch after the trigger signal's positive edge.

In order to perform clock glitching attack, one needs to run the application and wait for the respective clock cycle to apply an efficient fault. It is difficult and time consuming to test all the combinations of clock glitch parameters. Therefore, one needs to apply optimized search strategies for clock glitch configurations. The first assumption is that the glitch delay is equal to the Glitch Width (Glitch Delay= Glitch Width), and their sum equals to Glitch Period (Glitch Delay +Glitch Width= Glitch Period which is named as T_{glitch}). This assumption can help to have only two glitch variables to be configured including Glitch Period and Glitch Location. Figure 3.2 demonstrates the considered clock FIA parameters including Glitch Temporal Location, Glitch Width and Glitch Delay. One can also consider the additional parameters such as the Glitch Repeats, which is the number of times one aims to repeat the glitch in successive clock cycles, resulting in more advanced scenarios.

In the following, the approach to develop the configuration interface strategies for clock glitch configurations is presented.

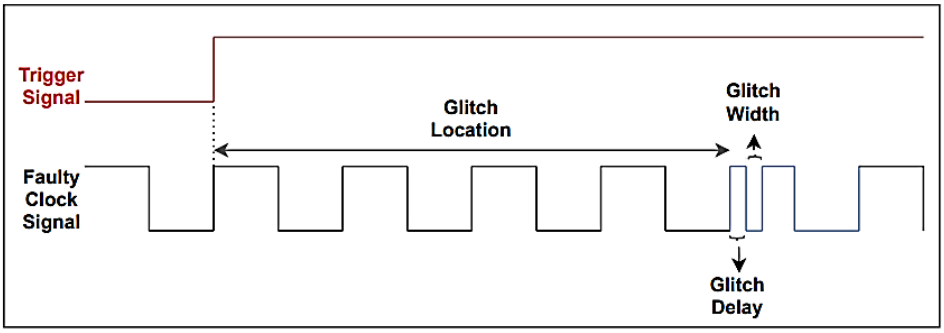

Figure 3.2: Clock Glitch FIA Parameters

3.2.2 Clock Glitch Configurator Interface

To have successful FIA experiments, one needs to improve the platform and build up an example database for each of the target processors' instruction sets with different sets of glitch parameters, namely Fault Settings. The first step in developing the configurator interface is to define a search space for different FIA parameters. In the search space of the clock glitch generator, the two fault variables are T_{glitch} and Glitch Temporal Location, and every combination of these values represents a specific configuration. The goal is to define the appropriate configurations, yielding a successful FIA. Therefore, one needs to apply an approach to obtain the proper values. However, choosing a proper algorithm that can give the best combination of FIA parameters in the search space is difficult. One can randomly sample this search space as the first possible solution. Note that an accurate range for the parameters is necessary for an evaluation process, and a lousy estimation of these ranges leads to spending much time to test the different parameter combinations. Regarding this, the idea of random sampling can be used as a basic approach in which there is no guarantee to find the minimum precision for the evaluation process.

To apply improved approaches and to obtain more optimized results than a random way, the search space needs to be narrowed down within respective bounds. This makes the evaluation process easier and more time-efficient. According to this, one can study the datasheet of the target processor and find the operating clock information. After that, based on that information, one can define the specific bounds for the T_{glitch} parameters:

- $Period_{min}$: A lower bound for the T_{glitch}, if the T_{glitch} is set to this value or lower, the device will ignore the glitch and mask it as a normal noise; therefore, the device response will be as regular operation.

- $Period_{max}$: A higher bound for the T_{glitch}, and if the glitch width is set to this value or higher, the device is affected by the glitch; however, the protection protocols try to tamper with it, and the device goes into reset mode.

The new bounded search space for T_{glitch} is in the ranges of ($Period_{min}$, $Period_{max}$). Figure 3.3 depicts the new search space for clock glitch parameters: Note that the glitch periods in (a) and

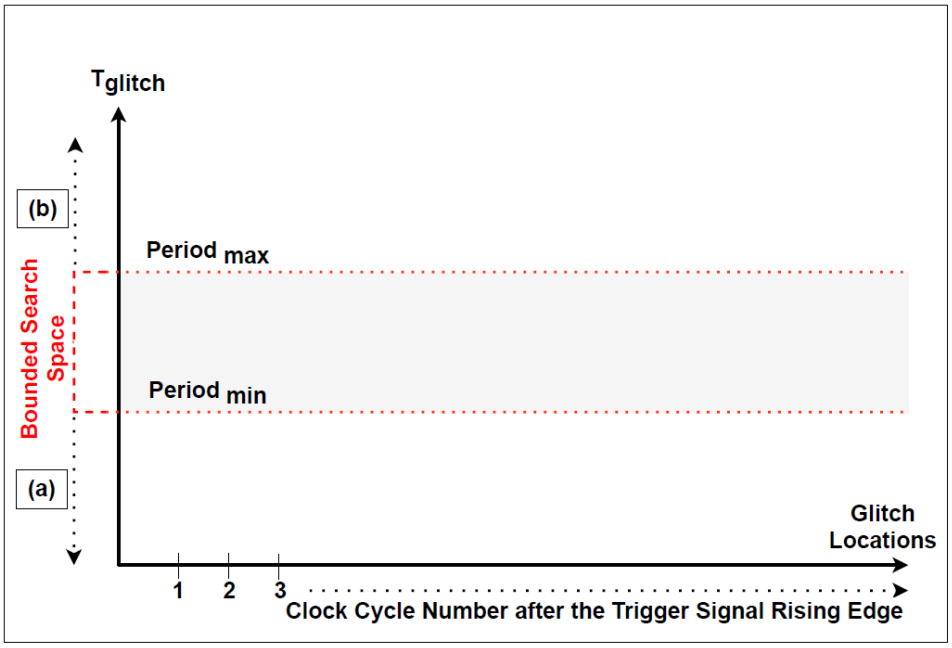

Figure 3.3: Glitch Period's Search Space Bounds

(b) are not considered in the new search space.

Now, having a limited bound, the aim is to find the best values for T_{glitch} in the defined interval to have a successful attack. We apply here a basic method to obtain the best T_{glitch} values. Regarding that, one can divide the interval into "N" equal spaces and obtain "N-1" respective value points. Depending on the required accuracy, one can increase the number of value points. For instance, if the interval is divided into two sub-intervals, then one T_{glitch} is obtained. As another example, if the interval is divided into eleven sub-intervals, then ten values are produced. Dividing the interval by more sub-intervals can provide a more accurate estimation of the proper T_{glitch} for the FIA configurator. Finally, these values are tested iteratively in each one of the Glitch Locations, and their respective results are analyzed. If a successful fault injection is reported, the related parameters will be marked and stored as the proper glitch parameters.

Figure 3.4 shows the framework of this platform in which the Fault Settings (e.g., related glitch parameters) can be stored for each specific embedded software instruction (e.g., beq, addi, etc.) running the target microprocessor. Since the fault effects are dependent on the target microprocessors' architecture, only the developed settings and experiments for specific targets are explained. However, according to this platform's flexibility, further models could be added based on the evaluation process targets and needs.

3.3 Fault Generator

This section presents a fault generator unit based on the clock glitch FIA. First, the requirements to implement the clock glitch generator designs are explained. Then, a guide is given to select the

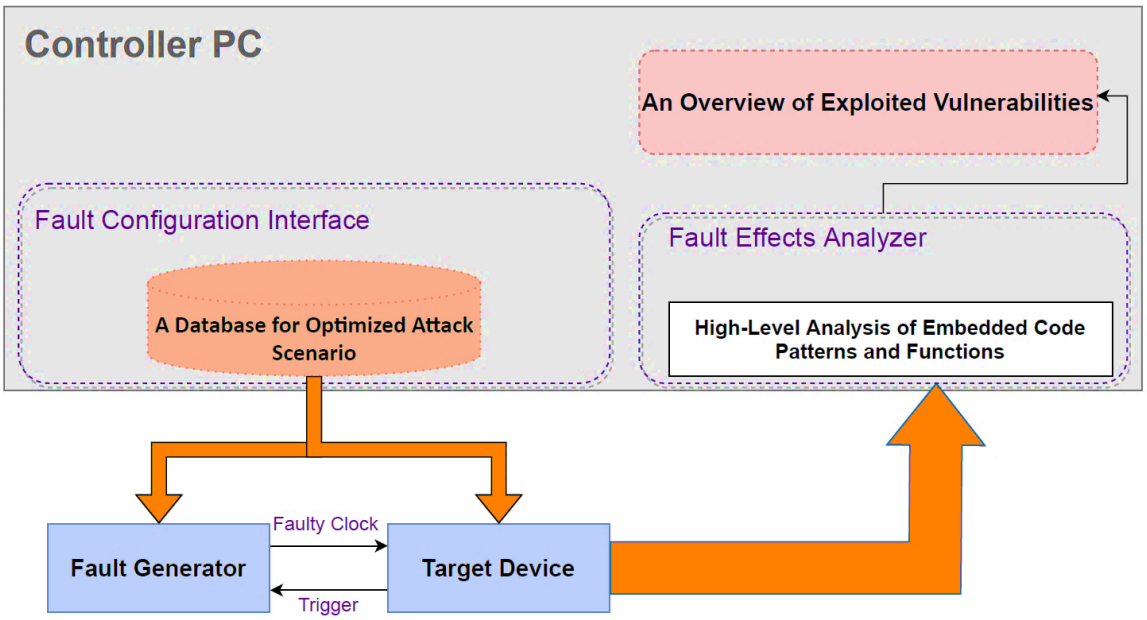

Figure 3.4: The improved architecture of the proposed evaluation platform

right architecture depending on different considerations, such as the used components, the design complexity, and the cost. Finally, two glitch generators are implemented, and their experimental results are discussed to verify their performance.

3.3.1 FPGA Implementation of the Clock Glitch Generator

As discussed in section 2.4, in order to design a clock glitcher, one can select between CSC (Combined Shifted Clock) and CDCF (Combined Different Frequency Clocks) methods and implement the design on a hardware platform. Among different platforms, FPGA development boards are easily accessible to implement glitch generator designs. Depending on the features that are important to the user, one can choose from the boards and implement the clock glitch generator design. The first important parameter is the price of the FPGA board, which fortunately does not impose a limit. For example, low-cost FPGAs such as Spartan-3 and Virtex-4 have a desirable feature named Digital Clock Management (DCM) that can be used to generate different versions of the nominal clock and create a glitch. A very similar feature can be found in the more powerful models such as Virtex-5 and Spartan-6 that have Phase Locked Loop (PLL) to generate glitches [80]. Moreover, advanced and more expensive FPGAs such as Virtex-6 and the seven series FPGAs have the Mixed-Mode Clock Manager (MMCM) in [81], including both the DCM and PLL features. The advantage of using MMCM is that it can generate multiple accurate clock signals with defined shifted-phases or divided/multiplied frequencies.

Another parameter to consider for fault generators is the run-time configuration. This feature allows the user to modify the glitch characteristics without re-programming the whole system and can be specifically useful in a testing scenario with various parameters. The authors in [44] present

an example where different clock glitch generator parameters, such as output frequency, phase, and duty cycle, can be dynamically reconfigured effortlessly in the run-time. This utility is present in the Xilinx 7 series, $UltraScale^{TM}$, and $UltraScale+^{TM}$ and is named the dynamic reconfiguration port (DRP).

In the following, the two fault generator designs based on the CSC and CDCF methods are explained. Then, the experimental results of an example target application are discussed to show their capabilities to exploit the desired errors. Afterward, the results of these two designs and their capabilities and performances are compared.

3.3.1.1 Clock Glitch Generator Design Based on the CSC Method

In order to design a glitch generator based on the CSC method, a Xilinx FPGA (Arty-S7-50) is selected. Figure 3.5 shows an architecture of a clock glitch generator that is implemented in this FPGA. It contains two phase shifting modules which are implemented by using the DCM, and then the signals are combined by some logical operations (e.g., XOR or MUX). Once there is a rising edge of the trigger signal, the glitch with the configured parameters can be injected. Using this method, one can control all the important glitch parameters such as Glitch width, Glitch Location, and Glitch delay. The run-time configuration for the glitch width parameter is related to the DCM block specifications, and partial reconfiguration makes it possible to change this parameter with some restrictions. The glitch delay inside the affected clock cycle is specified by the phase of the first clock signal.

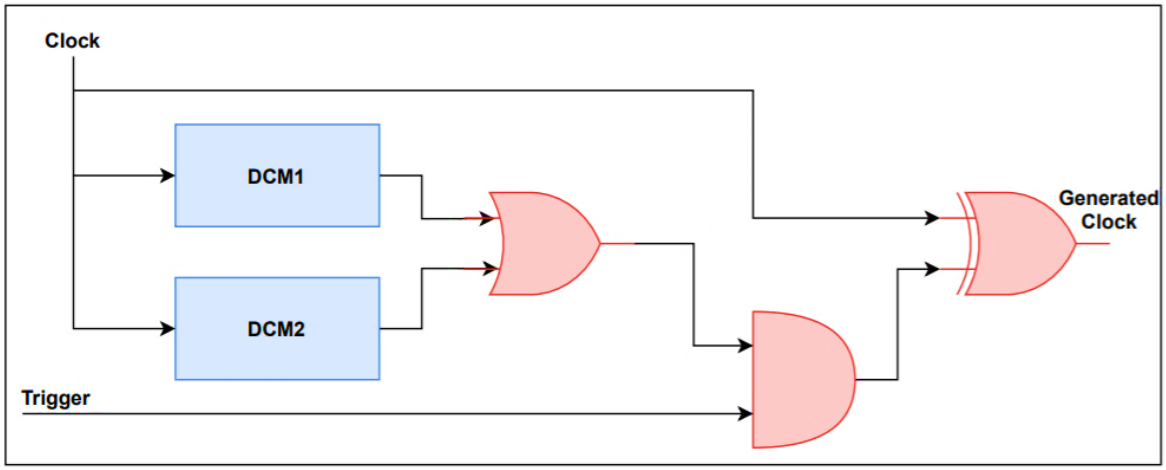

Figure 3.5: A Clock Glitch Generator Based on the CSC Method

3.3.1.2 Clock Glitch Generator design based on the CDCF method

The CDCF method is implemented with a Kintex 7 FPGA (Digilent Genesys-2). Figure 3.6 shows the two clocks that are fed into a specialized clock multiplexer and are available for Xilinx FPGAs (BUFGMUX). The multiplexer is used in an "asynchronous" mode to switch the slow clock to the

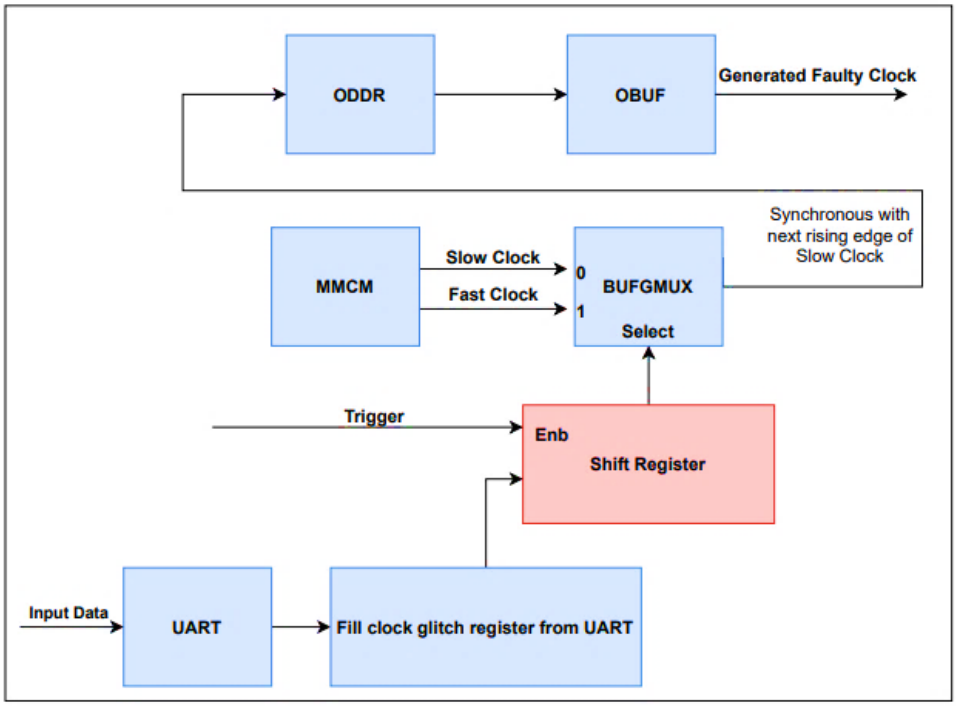

Figure 3.6: Glitch Generator based on the CDCF Method

fast clock when one aims to inject the clock glitch. The glitch injection is therefore realized by controlling the selection of the multiplexer. This selection is connected to a large shift register which is configured by the computer through a UART connection. Each bit inside the shift register corresponds to one potential clock cycle of the fast clock. Thus, when the shift register's output equals zero, the multiplexer outputs the slow clock, while it is equal to one, so it outputs the fast clock. In order to control the fault injection, a state machine in the FPGA is implemented, which initially configures the shift register. Then the system waits for a trigger from the target and synchronizes the shifting of the register to the select of the BUFGMUX on the "slow clock" edge, following the trigger. An example of this process is shown in Figure 3.7. In this process first, the fault injection setup is equipped with the clock generator. Then the software which controls the device under evaluation activates the trigger signal. Whenever the output value of the shift register equals to one, a single glitch will be injected. The output of the BUFGMUX is connected with an ODDR and an OBUF element to reduce the jitter and improve the driving capabilities for the output clock

This methodology can inject a clock glitch on multiple instants within a normal clock cycle according to the ratio of "fast clock"/" slow clock," which in our case was 208/16 = 13. Therefore, one is able to supply a clock glitch to the target in 13 different time divisions of each clock cycle of the normal clock. This setup is very flexible since it allows to configure any combination of single or multiple glitches during the computation under evaluation. The main drawback of the current setup is that if the glitch control shift register is very large, then it takes time to fill the register before every fault injection.

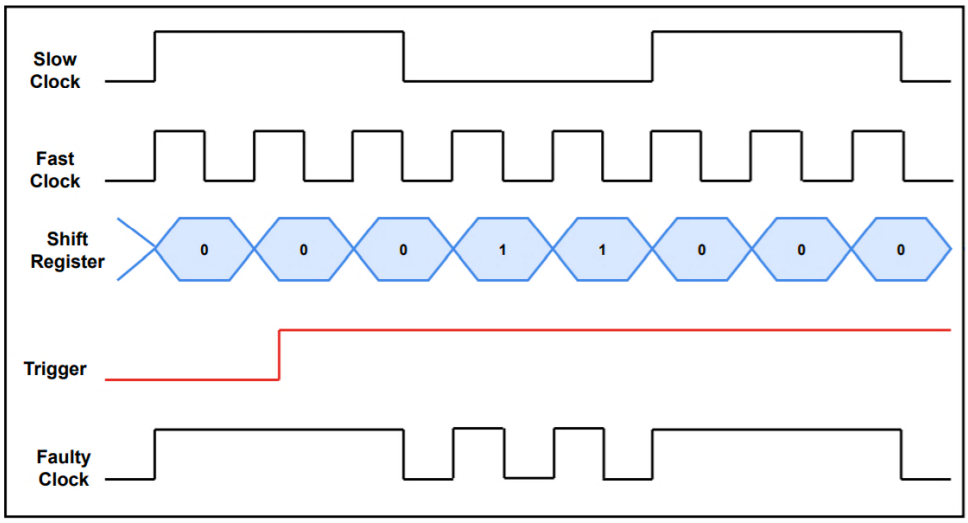

Figure 3.7: An example of faulty clock generation based on the CDCF method

3.3.2 Experimental Comparison of Clock Glitch Generator Designs, CDC vs. CDCF: Attacking AES Algorithm

In this section, an example of clock glitching FIA on the AES algorithm is presented to validate the efficiency of implemented fault generators. These glitch generators are then validated on an off-the-shelf ARM-Cortex-M3 32 bit micro-controller (MCU) target. Clock glitching FIA can cause an erroneous behavior of this algorithm and result in faulty cipher texts. The main goal of this experiment is to characterize the types of injected faults. It is important to perform an accurate clock glitching FIA and have a single bit faulty value in generated AES cipher text. The AES algorithm is applied to encrypt or decrypt data blocks of 128 bits by using a secret key of 128, 192, or 256 bits [82, 83]. The key length decides the total number of encryption rounds. Except for the last round, each round consists of four transformations: SubBytes, ShiftRows, MixColumns, and AddRoundKey. Compared with other rounds, the last round does not execute the MixColumns function. A structure of AES 128-bit has been shown in Figure 3.8.

Here, a short explanation of the main steps of the AES-128 algorithm is presented:

1. SubBytes is a nonlinear byte substitution in which another value replaces every byte from the 16-byte state. In the SubBytes step, each byte in the state is replaced with its entry in a fixed 8-bit lookup table.

2. Shift rows are circular shifts on the four rows of the state. More precisely, row i is transformed by a circular shift on bytes by i positions to the left.

3. A mixed Column is a linear bijection on the four columns in parallel.

4. AddRoundKey consists of an XOR operation of a generated schedule from the original 128-bit key utilizing a key expansion, which means that a different, unique key is generated and added to the state for every round.

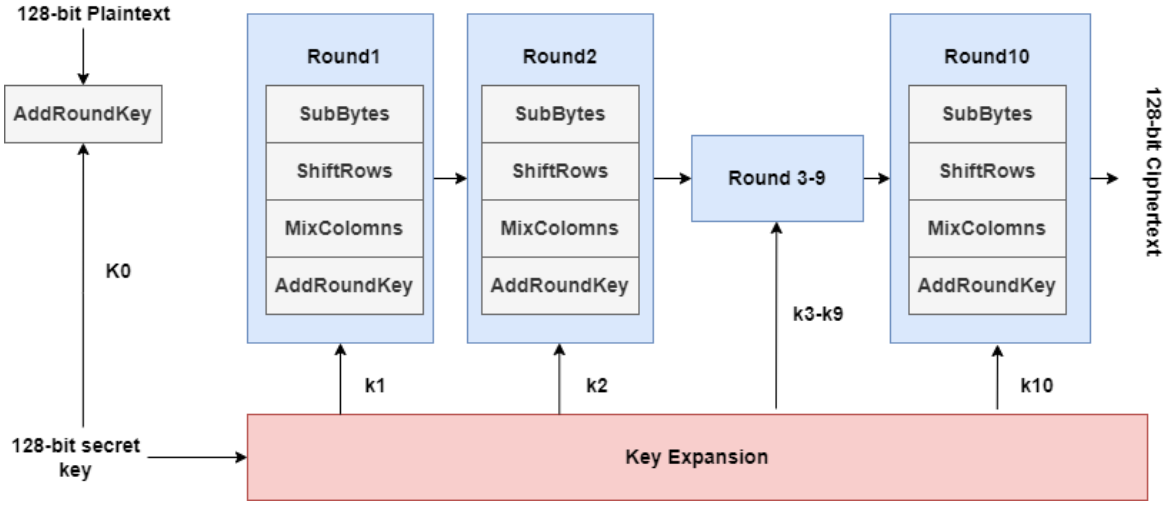

Figure 3.8: Structure of AES [83]

All the steps are illustrated in Figures 3.9, 3.10, 3.11, 3.12.

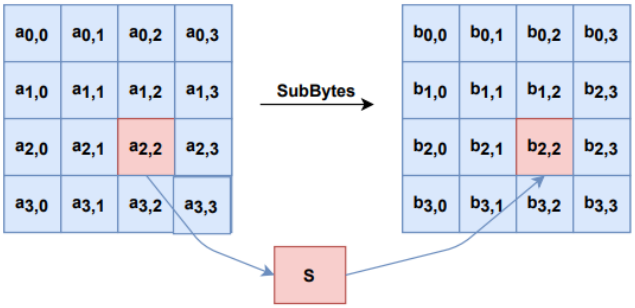

Figure 3.9: SubBytes in AES 128

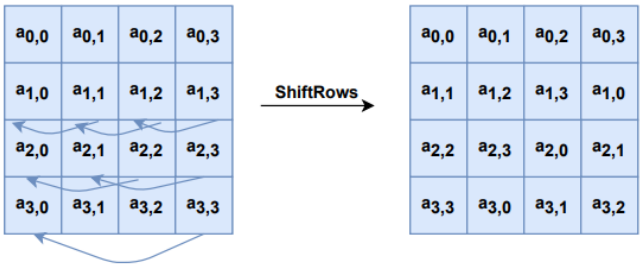

Figure 3.10: ShiftRows in AES 128

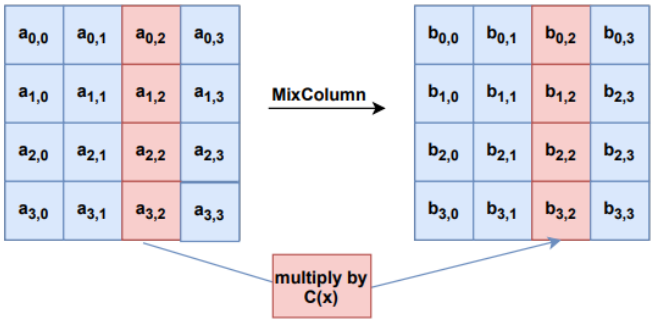

Figure 3.11: MixColumn in AES 128

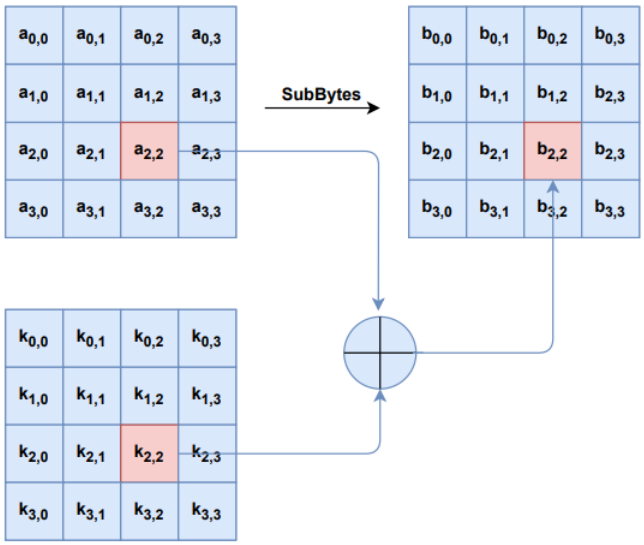

Figure 3.12: AddRoundKey in AES 128

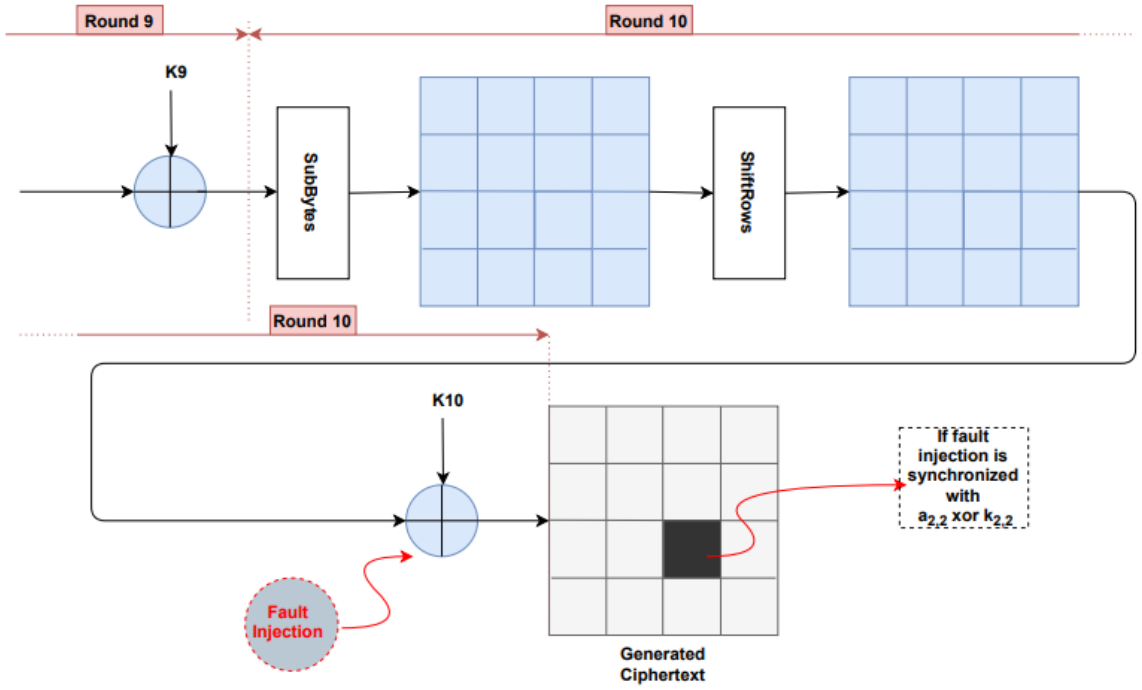

Figure 3.13: Fault Injection on a single byte of an AES-128 bit

In the following, a clock glitching FIA on an AES-128 running on a target MCU is performed. As an example, the AddroundKey function of the last round (10th round) is selected. Moreover, the glitch generator is configured to examine the effect of injecting a single clock glitch with proper parameters on each of the 410 clock cycles needed for executing this function. Fault Injection on AddRoundKey of the last round of AES-128 bit is illustrated in Figure 3.13. In order to characterize and compare the capabilities of the clock glitch injection platforms, a fault injection experiment on an AES algorithm running on an ARM-CortexM3-32bit MCU is performed.

For the setup, the evaluator needs to follow the high-level diagram of Figure 3.14.

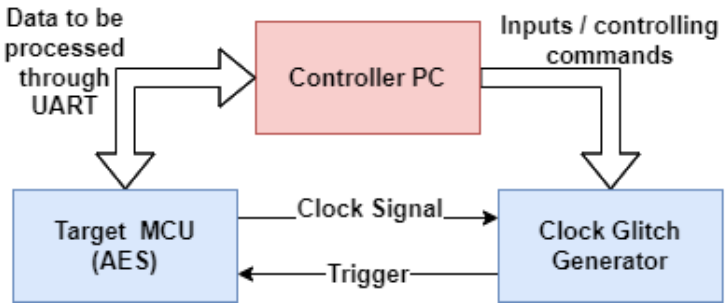

Figure 3.14: The Clock Glitching FIA Setup

The setup consists of a control PC, a target board, and a fault-injection module. One can control and configure both the fault-injection module and the target board by using the PC. UART performs the communication between the MCU and the PC. Moreover, this PC acts as a user interface and configures the clock glitch generators via a second UART interface.

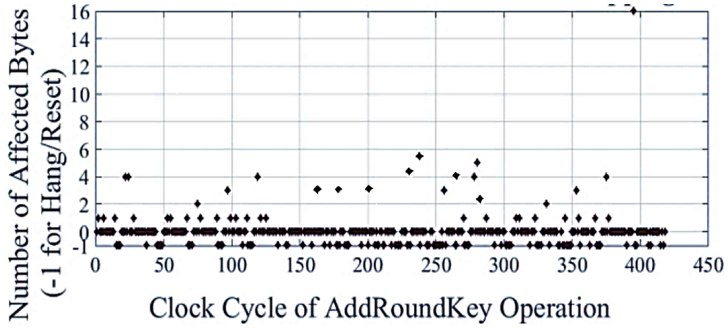

Figure 3.15: Fault Mapping, CSC generator

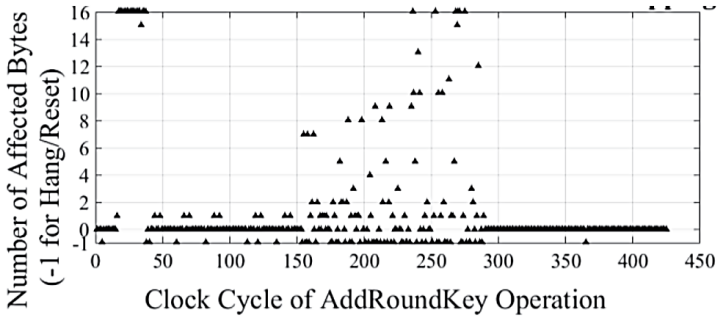

Figure 3.16: Fault Mapping, CDCF generator

In this work, the AddRoundKey operation of the last round (10th round) is attacked. After applying the glitch, the computed cipher text of the AES has been saved and sent to the computer. In this work, 4100 fault injections were performed. Furthermore, in order to be able to verify the behavior of the fault injection with different data being processed, for every different setup of glitch attack parameter, ten fault injections have been performed with random plain texts. Thus, ten fault injections have been performed in each of the 410 clock cycles of the last round AddRoundKey operation, while all 4100 encryptions are performed with random plain texts and the same key.

Figures 3.15 and 3.16 depict the cartography of the injected faults for the two injectors. The horizontal axis contains all 410 clock cycles of the AES operation under attack.

On the vertical axis, the number of affected bytes of the AES's state register due to each of the 410 depicted attacks has been plotted (with random plain texts and the same key). When a fault injection results in a reset/hang of the MCU, the value of minus one is assigned to that. Such cartography is very useful because it can show the clock periods of the computation, which can lead to a fault of a specific impact. The obtained patterns for the ten repetitions of the attacks are very similar. This shows consistency during the fault injection with different plain texts concerning the number of affected bytes. Such cartography can also be used later to focus on a more thorough evaluation of specific clock cycles. Comparing the two figures, one can notice that the use of the CSC glitch generator led to fewer successful fault injections than the CDCF, which shows that the overall setup and the glitch location used for fault injection play an important role in the acquired results.

Table 3.1: Glitch Generator Comparison of Affected Bytess

	Number of affected bytes & Hang/Reset percentages (%) out of 4100 injections																
	Hang Reset	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
CSC	25.1	66.4	3.9	0.5	0.7	1.1	0.4	0	0	0	0	0	0	0	0	0	0.1	2
CDFC	13.4	64.5	6	2.8	0.6	0.1	0.6	0.4	0.6	0.8	0.6	0.7	0.3	0.2	0.2	0	0.3	8

Table 3.2: Fault Multiplicity of Single-Byte Faults

	Number of affected bits in single byte faults							
	1	2	3	4	5	6	7	8
CSC (161 total 1-byte faults)	1.2	1.2	2.5	26.1	30.4	26.1	12.4	0
CDFC (255 total 1-byte faults)	0	11	27.1	29	19.2	9	4.3	0.4

In Table 3.1, the glitch generators are compared in terms of the number of affected bytes. In this table, the results of all 4100 injections are illustrated with the corresponding percentages of injections, which led either to a fault-free operation or to a Hang/Reset of the MCU or to a successful injection. For successful cases, the number of bytes that were affected are also mentioned. This can confirm reproducibility and also help us to figure out if there is a relation between plain text and the number of faulty bytes or not. The faults which led to a Hang/Reset were 13.4% for the CDCF generator while 25.1% for the CSC generator. Furthermore, the CDCF glitch generator led to considerably more errors affecting one or two bytes, and at the same time, it caused more faults affecting all 16 bytes of the AES.

Table 3.2 provides the results regarding the number of bit flips injected in the subset of injections affecting a single byte of the AES. This time it is observed that the Clock CDCF generator did not lead to any faults of one bit, while the CSC generator led to 1.2% of errors of a single bit. On the other hand, the CDCF glitch generator led to a higher amount of 255 single-byte faults versus 161 for the Phase Shift glitch generator.

Although the results of the two glitch generators are different, they show that both glitch generators are capable of injecting well-controlled faults into the MCU. However, the CSC approach has more parameters to control than the CDCF method does. For instance, with the CSC method, the glitch delay can be manipulated inside any single clock. Moreover, much thinner glitches can be produced by applying the CSC method due to the existing DLL constraint of generating higher frequencies for the CDCF method.

3.3.3 Design of an Efficient and Automated Clock Glitch Generator

As discussed in the previous section, the minimum glitch width with the CSC method is less than the CDCF and the glitch location is controllable inside any of the single clocks. Therefore, the CSC method has been selected for the rest of the experiments in this thesis work. In order to automate the clock glitch generation and to induce the glitch in all clock cycles of the targeted software, an advanced FPGA (Arty S7-xc7s50) board has been selected. Figure 3.17 illustrates the high-level architecture of the clock glitch generator. This clock glitcher consists of three sub-modules, including a Frequency-Convertor, a MMCM-Dynamic-Phase-Shifter, and a Glitch Injector.

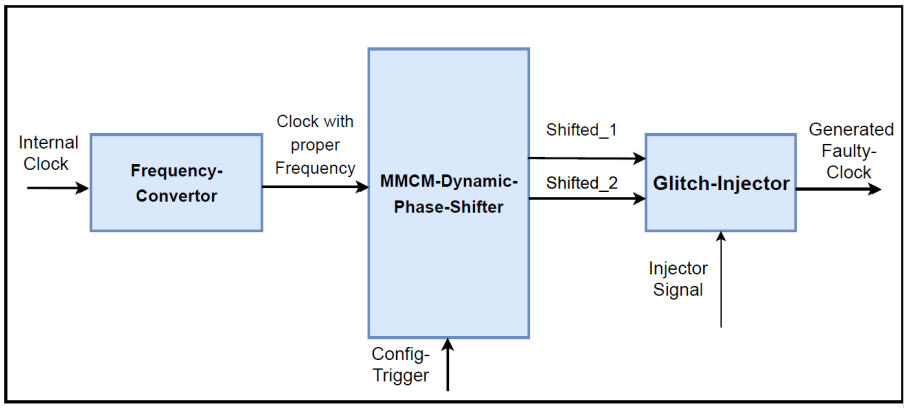

Figure 3.17: General Architecture of the Clock Glitch Generator

The Frequency-Convertor module is used to convert one of the available internal clock signals in the FPGA (12 and 100 Mhz for Arty S7-xc7s50) to the desired frequency. Then, the MMCM-Dynamic-Phase-Shifter utilizes the Dynamic Reconfiguration Port (DRP) feature to generate shifted versions of the converted clock signals. The shifted phase values (shifted_1 and shifted_2) are determined in a state machine shown in Figure 3.18. They can change from 0 to 90 degrees and this can cause to generate different glitches with various parameters (glitch width and location inside a single clock). Then, another state machine shown in Figure 3.19 is used to insert the generated glitches in specific clock cycles.

Figure 3.20 elaborates one example of a generated faulty clock in which the glitch is inserted after 1,2 and 3 cycles after the rising edge of injector signal.

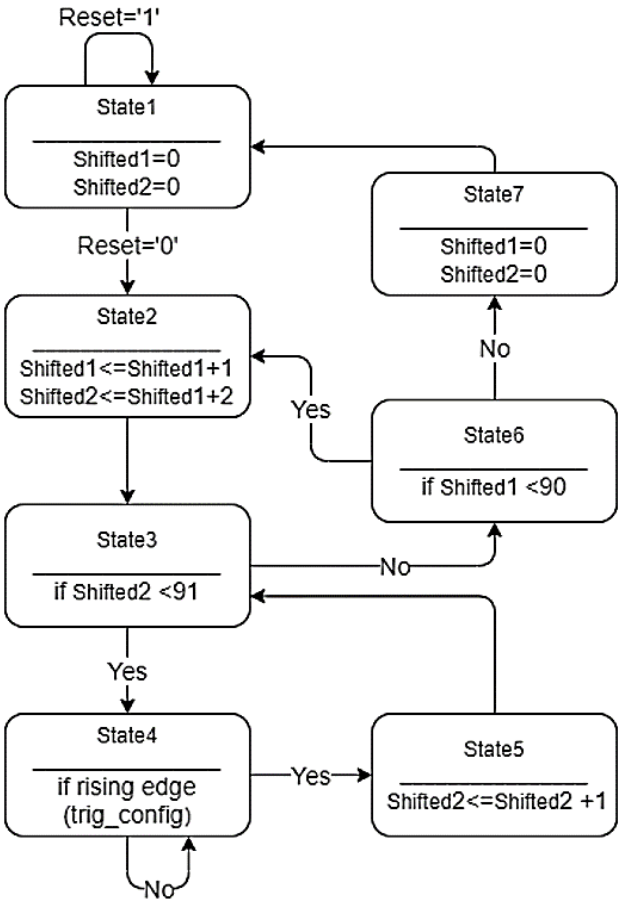

Figure 3.18: The State Machine for Updating the Phase Shifts

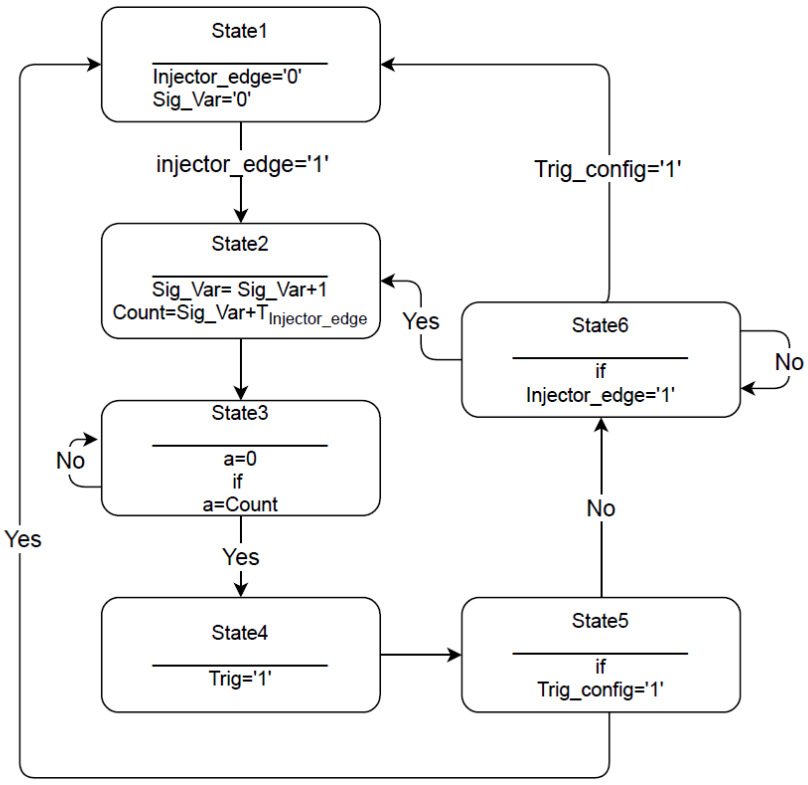

Figure 3.19: The State Machine for Updating the Glitch Location

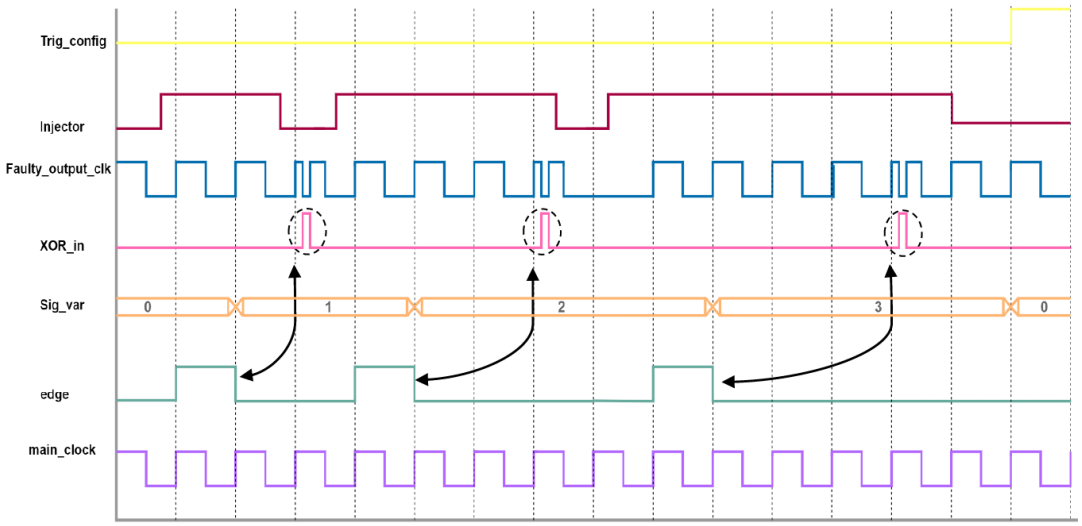

Figure 3.20: An Example of the Generated Faulty Clock

3.4 Fault Effect Analyzer

This section defines the high-level analyzer of the evaluation platform and explains the proposed test scenarios and approaches. The goal is to exploit and analyze the high-level vulnerabilities of a target embedded software after injecting faults. After this analysis, the evaluator should have information about the security vulnerabilities in the early developing stage of the application.The problem is that it is not trivial on an application to understand in details which faults might occur. Therefore, the goal is to instrument the C code and to be able to catch the fault effects at higher software levels. The following focuses on analyzing the most prominent patterns in the program control flow and common functions.

3.4.1 Main Control Flow Patterns and Their Evaluation Methods

In this part, in order to present the evaluation approach, the important control flow statements are categorized into three main classes, namely: 1) Unconditional Branches, 2) Decision Makings, and 3) Iterative Controls. Table 3.3 shows these statements with some C code examples.

Table 3.3: Important Control Flow Statements

Control Flow Statements	**Type**	**Examples**
Branching/Skipping	Unconditional	Continue/Break/Go
Decision Making	Conditional	If/If-else
Iterative	Conditional	For/Do-While/While

In the following, in order to illustrate our method, different graphs are utilized in which the

nodes plus the edges represent the code segments and the control flows, respectively. To illustrate the vulnerability of control branches one or two check points (CP1 and CP2) are assumed. In fact, a check point is an added variable in the software which its value can show the execution of a certain path.

- Branching/Skipping evaluation: Figure 3.22(a) shows the unconditional branch as a basic control flow that contains an outgoing edge from a node in a control flow graph. Figure 3.21(b) shows an extension of a control flow evaluation with the inserted check points where the program runs in the presence of the fault injection. When CP1 was activated, the injected fault did not affect the correct execution of the branch, and when the CP2 was set, it showed the branch was corrupted. When none of the checkpoints are activated, it implies that the PC register contains an incorrect instruction memory address (represented as X).

- Decision-making evaluation: Figure 3.22 (a) shows a basic decision-making control flow graph (if-else) that contains a decision node with two control branches. The conditional control branches are merged after executing the statements of the selected branch (white circles). Figure 3.22(b) presents an evaluation example of decision-making statements by first setting the condition to a state which leads to a known result, and then CP1 and CP2 are inserted to monitor the consequence. In this example, it is expected that the condition is false, so when the fault injection is unsuccessful, the CP2 is activated. When the CP1 is set, one can detect the skipped conditional test.

  To evaluate the nested conditional branch (branches in branches), more checkpoints are needed. Figure 3.24 illustrates the inserted checkpoints in a nested branch. In this case, the conditions are set to give false results, and therefore, it is expected that the CP3 to become activated. This indicates the error-free execution of this statement. The fault injection can result in the activation of CP1/CP2, and one can detect the vulnerability of branches.

- Iterative control evaluation: Figure 3.24 presents the while loop evaluation as one of the iterative control statements. First, it is assumed that an always true condition such as while (1), then the CP1 as a watchdog flag is defined to detect the skip from the loop. In this case, if CP1 becomes active, it demonstrates that the fault injection has manipulated the correct execution of the loop.

  To evaluate a finite iterative control flow statement (e.g., for-loop), using the check point is not efficient to detect the fault execution. Instead, on can monitor its number of iterations by using a counting variable in presence of fault injection. For this, the same for loop is run twice (one in the presence of fault injection and the other is executed in a normal situation). At the end of the loop, the final values of counting variables are compared. If these two values are not equal, it indicates that fault injection has caused an error in the execution of the loop.

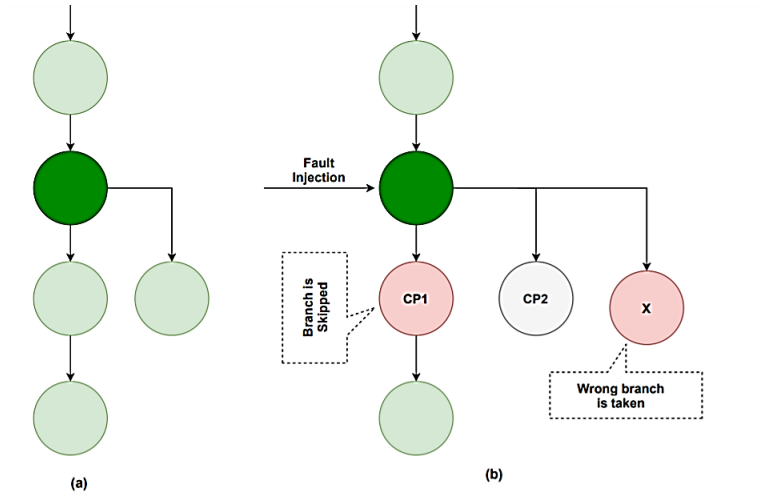

Figure 3.21: Control Flow Evaluation for Unconditional Branch

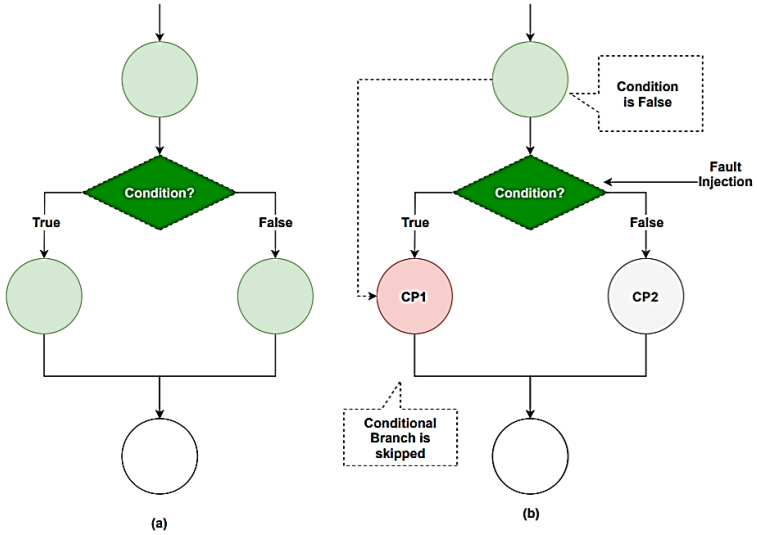

Figure 3.22: Control Flow Evaluation of Single Conditional Branch

Next, in order to cover more evaluation scenarios, some standard C functions are studied against fault injection attack.

3.4.2 Main Standard C-Functions and Their Evaluation Methods

This section aims to explain more general evaluation scenarios to exploit the vulnerabilities of standard high-level C functions. First, the standard embedded C-functions are categorized, including:

- Type Casting Functions perform data type conversion from one type to another. Two important examples are *atoi* and *itoa*, which convert string to int and int to string, respectively.
- String Manipulation Functions can modify the strings. There are various functions in this class, including *strcpy* and *strncpy* to copy a string to another.
- Memory-Based Functions manipulate the data inside the memory and are specifically vital

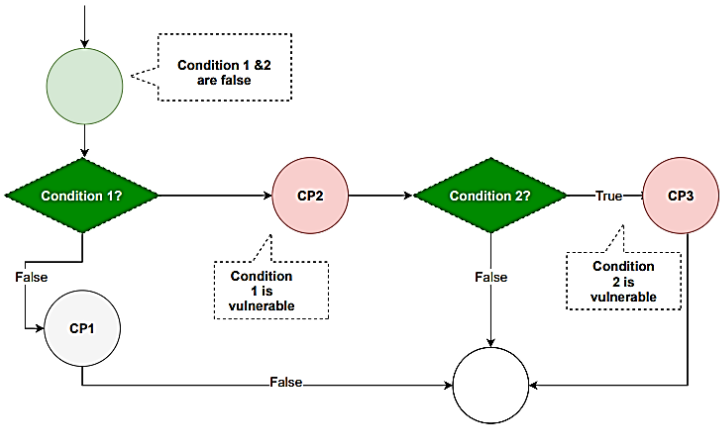

Figure 3.23: Control Flow Evaluation for Nested Condition

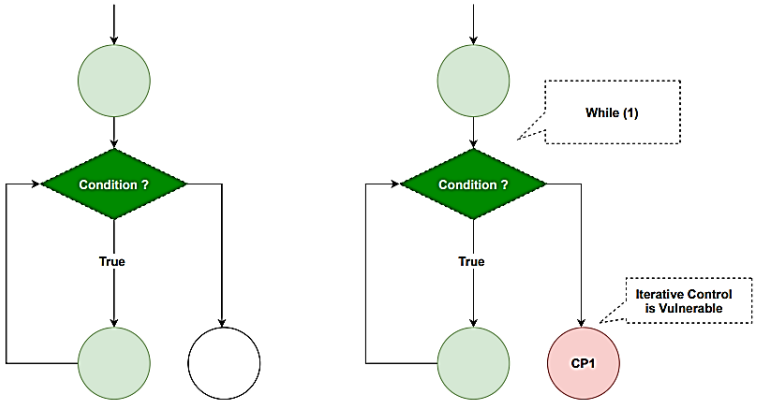

Figure 3.24: Control Flow Evaluation of an Iterative Control

for a system's initialization. Important examples of these functions are *memset* and *memcpy*, which are used to set all the bytes in a block of memory to a particular value and to copy a block of data from a source address to a destination address.

- Searching and Sorting Functions include examples such as *bsearch* and *qsort*. A *bsearch* sorts an array and then searches the desired record based on the binary search tree algorithm. A *qsort* function sorts an array of numbers. To evaluate it, each element is weighed at the output array.

There are different evaluation scenarios for different C-functions in the analyzer interface. For example, to test the Type Casting Functions, an (input, output) pair is selected and the function runs in the presence of FIA. If the generated result differs from the expected one, a successful attack is reported. For the Sring Manipulation Functions that transfer or copy data from a source to a destination, the results are compared with *strcmp* to detect any possible mismatch. The Memory-Based Functions are assessed by feeding them with known values and checking the specific memory location(s) related to the operation. A fault-affected function will result in an incorrect memory

address or value. The output of the Searching Functions can be observed when a known array is given to the function, and when they return a null value, it means that the attack was successful. To evaluate the Sorting Functions, each element of the output array is weighed. Then, when the sorted array is generated, the sum of all the multiplication of weights and related elements of the arranged array are compared. If these two are not equal, it means wrong sorting. Table 3.4 summarizes different categories of the standard library functions with the normal and their faulty behavior in front of FIA.

Table 3.4: The behavior of different high-level C-functions

C-Functions	Normal Behavior	Faulty Behaviour In Front of FIA
atoi	Converts an ASCII array to an Integer value	Corrupted integer value
itoa	Converts integer value to an ASCII presentation	Corrupted ASCII value
memset	Saves value in memory	Corrupted memory value or manipulating neighbor memory blocks
memcpy	Compares the values in memory	Faulty comparison results
strcpy	Copies from one-character array to another	Corrupted or incompleted copied array
strncpy	Copies portion (n-bit) of contents of one string into another string	Corrupted or incompleted copied array character
strchr	Finds the first occurrence of a character in a string	Finds the wrong position or does not find
strtod	Convert string to double value	Corrupted double value
qsort	Sorts an input array	Change the position of values in an array
bsearch	Searches an array to find value	Does not find the value

3.5 CONCLUSION

This chapter of the thesis proposed a practical hardware evaluation platform to analyze the embedded software vulnerabilities against the clock glitching FIAs. The architecture and the main components of this hardware platform were studied. Then, in order to implement an efficient clock glitcher, the architecture of two different clock glitch generator were compared by targeting an example (running 128 bit AES algorithm) and the best architecture was chosen for this work. One of the limitations of this experimental platform is that it can take a long time to consider all of the possible glitch parameters.

In the following of evaluation platform, different high-level analysis methods were proposed to help the software developer to detect the existing vulnerabilities of different patterns and standard functions against these hardware-based security attacks. Regarding this, another point to consider is that the test scenarios are applied at functional level and it might get difficult to use them to evaluate an overall embedded application. Therefore next chapter will use this hardware platform and guide the software developer step by step to evaluate an embedded application.

4 HARDWARE SECURITY ASSESSMENT BY UTILIZING THE HARDWARE EVALUATION PLATFORM

Contents

Securing IoT applications against hardware-based attacks is an increasingly important concern for embedded software developers. To answer such a need, a hardware evaluation platform along with high-level analysis methods have been proposed in the previous chapter. However, having an implemented fault injection hardware platform and using only basic evaluation approaches are not sufficient to assess a complex application which has different modules. In other words, the software developers need to be guided properly to identify the critical assets of their application and then to conduct the appropriate assessments against the experimental FIAs. Therefore, this chapter shows how to utilize an evaluation platform efficiently to assess an embedded software against clock glitching attacks. It uses a divide and conquer approach that includes four steps (Identification, Classification, Evaluation and Mitigation), named as ICEM. Accordingly, this approach reveals the procedure of how one developer can identify the critical assets and find the potential points of interest (in terms of security) of the underlying application. These points can affect the credential information, the data/control flow integrity, and the availability of the critical services when targeted by the FIA.

In this chapter, a medical IoT application is being analyzed by using our evaluation platform and the ICEM approach as a case study along with a proof of concept. This example shows how to discover the vulnerabilities, and presents a few software-level examples to mitigate the mentioned FIA impacts on an application's security level. The results and analysis of this chapter show that by using ICEM approach, one can easily detect the system susceptibilities. Thereafter the system security can be enhanced by adding proper countermeasures at the most security sensitive units.

In the following, the ICEM hardware security assessment methodology is described. Then, each sequential step of this assessment approach is explained in detail.

4.1 ICEM Assessment Methodology

The ICEM assessment approach consists of four main steps for the embedded IoT applications: 1)**Identification** of sensitive assets, 2)**Classification** of the assets based on their security properties, 3)**Evaluation** of the assets, and 4)**Mitigation** of software-level vulnerabilities. Figure 4.1 shows the flow of this approach that a developer should apply, and each step is explained in the following parts of this chapter.

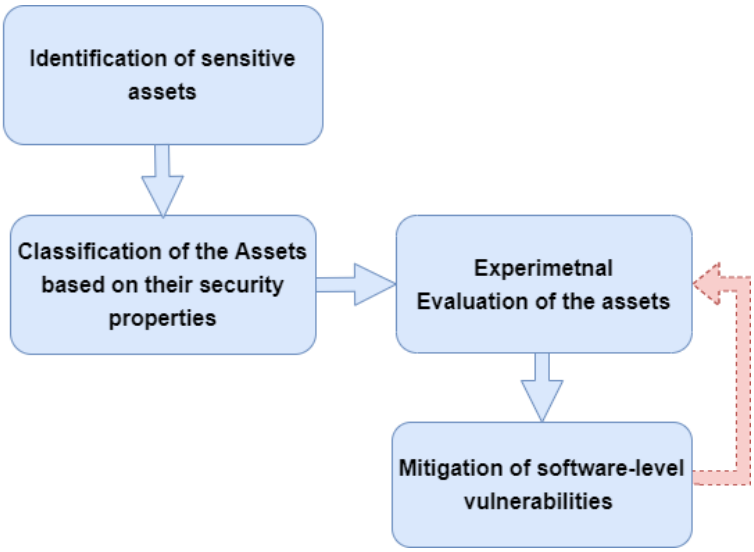

Figure 4.1: ICEM Assessment Methodology

4.1.1 Identification of sensitive assets

Initially, all the application's sensitive assets which have high value for the service provider or the end-user must be identified. These assets can take various forms, including the device's intentional operational flow, the firmware, the user IDs and passwords, the encryption keys, the entered information from the user or sensors, the stored system logs and libraries. By making a list of critical assets and prioritizing them, software developers can effectively decide the required testing coverage level for each one of them.

4.1.2 CLASSIFICATION OF THE ASSETS BASED ON THEIR SECURITY PROPERTIES

Each asset has some associated security properties. Categorizing the mentioned assets in an IoT application is critical because all subsequent steps rely on this step. Assets can be classified into three main groups: 1) Confidentiality-related, 2) Integrity-related, and 3) Availability-related assets and they are explained in the following:

- Confidentiality-related assets: these assets are the ones that contain various sensitive user data and should only be accessible by authorized people. Confidentiality-related assets should be highly secured to avoid any leakage of the user or the device information. Password encoding and determining different access levels are typical solutions in the software development stage. Moreover, the hardware developer's dedicated protected storage for confidential and important data is common [84].

- Integrity-related assets: this category includes the information that an embedded application may record or process in an intended manner. Only authorized parties can modify the modules' functionality. Therefore, the targets' critical functional modules are considered as integrity-related assets, and they must be protected against any unintentional modification by an attacker. In other words, the accuracy and consistency of a functional unit over its life cycle must be guaranteed in a secure embedded device.

- Availability-related assets: these assets are mostly related to the user interface for an embedded application. They enable communication between the service providers and the end-users. They collect the data from the physical entities and update the firmware according to its requirements. These units must always remain operational. Hence, a complete analysis must be performed to evaluate the service loss risk under different physical attack scenarios.

4.1.3 EXPERIMENTAL EVALUATION OF THE ASSETS

To perform an experimental evaluation of different assets and to discover the existing vulnerabilities in an application, one can split the application into multiple modules. Based on the targeted module properties, the right attack configuration and analysis scenarios from the evaluation platform can be applied. For example, the evaluation of a module that performs the computing task is different from a module that authenticate the users. In this work, the focus is on the clock glitching FIA, which can subvert the confidentiality, integrity, or availability of the vulnerable assets. For example, the attacker can skip the privacy controls and authentications (confidentiality), enforce the target to write erroneous data in unintended locations (integrity), perform corrupted read operations from the memories (integrity), manipulate the functionality by corrupting a control flow (integrity-availability).

4.1.4 Mitigation of Software-Level Vulnerabilities

After evaluating an application against the fault injection attacks, the software developer can conclude the to be employed security level based on the discovered vulnerabilities; and can determine the security objectives which are defined at the application level. At this point, a list of required mitigation patterns for a secure application can be proposed to protect it against similar FIAs [85, 86, 87]. Furthermore, considering target devices' power and memory limitations, the developer can apply the appropriate mitigation based on the asset's priorities.

4.2 Evaluation of a Medical Embedded Application against Clock Glitching FIA: A Case Study

This section aims to apply the ICEM methodology to an example of a medical embedded application. Generally, medical IoTs do not always offer a high level of security and lack the hardware-security standards. Moreover, the applications of these examples are written in unsafe languages such as C. The criticality and lack of a clear evaluation approach of the embedded application in this domain was the reason to apply this methodology to a medical IoT case study. An excellent example of a medical IoT device exists as an infusion pump installed in hospitals to deliver doses of drugs to patients and monitor their health status. Figure 4.2 shows an infusion pump's physical architecture connected to the network, which different users with different credentials can configure. The pump's generated logs and data are stored and sent to the central service provider, such as the hospital.

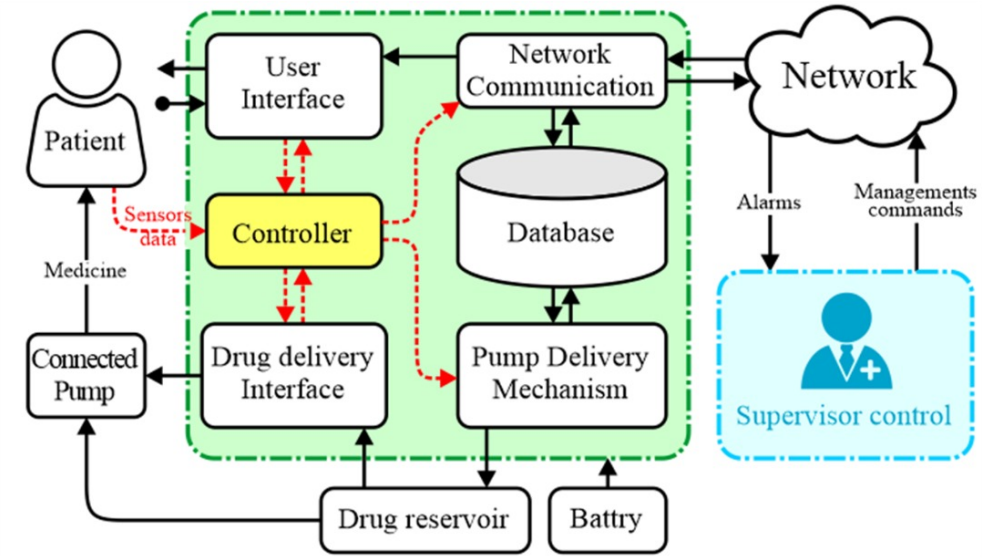

Figure 4.2: Infusion Pump Physical Architecture

The lack of openness to research and posed commercial constraints of these infusion pumps made it impossible to assess a real commercial example. Accordingly, an open-source and security-

oriented medical application was selected to model the behavior of a life-critical infusion pump named as "Sec-Pump" from the SERENE-IoT Project [8]. This example has been designed to be used for both research and teaching activities related to embedded system security and runs on both ARM and RISC-V microprocessors [88]. Sec-Pump can demonstrate the potential security breaches and vulnerabilities of other existing infusion pumps [49].

4.2.1 IDENTIFYING THE SEC-PUMP ASSETS

Figure 4.3 shows the important modules of the Sec-Pump application and identifies the critical assets within each module,

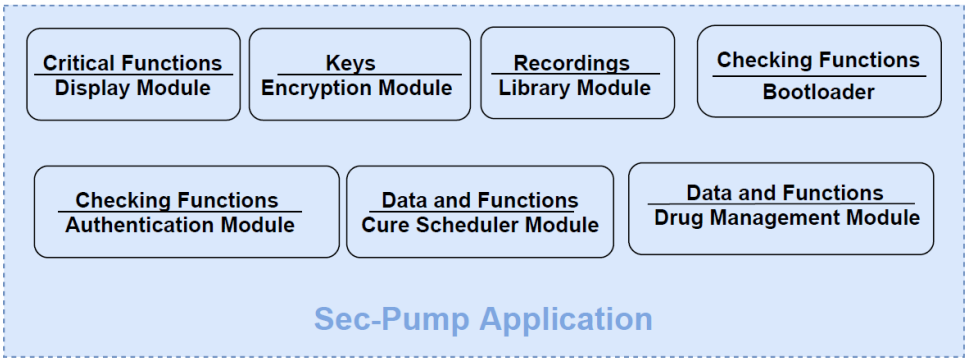

Figure 4.3: Critical Assets in Sec-Pump

These modules and their assets are briefly explained as follows

- Display module: It is responsible for showing different information, including the time, injective drug, patient name, etc. Since this module is easily accessible and important for information transfer, its crucial functions need to be secured against related physical attacks.
- Network and Cloud Communications: It configures the Sec-Pump and sends/receives the administrative commands and data packets. This module should always be connected to transfer commands and to monitor the running status.
- Bootloader: This module calls the application and runs an interface to display the infusion pump's current version and functionalities. It configures the Sec-Pump set values and memory addresses which plays an important role in the program flow.
- Encryption module: This module encrypts the critical user information and the variables used by other modules. The encryption keys and the flow of important instructions should be secured against any physical attack.
- Authentication module: This module requests a password from the user to provide a session with privileges. If an attacker can reveal the passwords or bypass the checking instructions, he/she will have the highest level of access to the system.

- Library: The secure pump applies much information from its library to secure functions at runtime. The attacker may target the proper functionality of this module.
- Clock Scheduler: This module enables the doctor to choose a drug from the library and inject it into the patient's body at a specific time and with proper dosage. If the attacker can modify the drug dosage or disrupt its schedule, a critical situation will happen for the patient.
- Drug Management: This module allows the doctor or medical authorities to modify the boundary parameters, delete the drugs, or add new medicines. These values should be protected from any non-authorized person.

4.2.2 Classifying the Sec-Pump's Assets based on their Security Properties

The second step is to define some functionality level categories and to classify the assets in the related categories. Regarding this, Table 4.1 presents the different asset categories of the Sec-Pump application.The first group presents the confidentiality-related assets, including encryption and authentication, that contain various sensitive user data and should only be accessible by authorized people. Then, all of the critical functions in the boot loader, clock scheduler, and drug management modules are placed in the Integrity-related assets, and their accuracy and consistency are necessary for Sec-Pump's secure functionality. Finally, the network and display modules are linked to the availability of the device because they enable the communication between the service providers and the end-users.

Table 4.1: Different Asset Categories of Sec-Pump

Category	**Module**	**Assets Examples**
Confidentiality-related assets	Authentication	Username and passwords
	Encryption	Encryption keys
Integrity-related assets	Bootloader	The Program Counter (PC) register
	Clock Scheduler	Functions for setting the values
	Drug Management	Functions that affect the main outputs
Availability-related assets	Network	Essential functions to maintain the connection
	Display	All the necessary functions

4.2.3 Experimental Evaluation of the Sec-Pump

In the following, two of the Sec-Pump application's integrated modules, including Authentication and Drug Manager, as case examples are evaluated. The experimental setup to apply clock glitching FIA into Sec-Pump application and to debug its faulty behaviour is shown in Figure 4.4. The

Sec-Pump application is being executed on an RISC-V Rocket Core which is implemented in an Arty-A7 FPGA [89]. An Olimex debugger is used with JTAG interface and is supported by OpenOCD (Open On-Chip Debugger)to program/ debug the target board. OpenOCD is an open-source software that interfaces with a the JTAG port of the hardware debugger [90]. The target device's clock signal has been modified in three steps: 1) a clock-core signal has been defined and connected to the clock source in the FPGAChip.scala file. 2) a clock port is created and connected to this clock-core signal in the ArtyShell.scala file. 3) The clock-core signal is connected to one of the PMODs in the arty constraint file (arty-7.xdc) [91]. In this way one have access to the clock core of the system to apply the clock glitching FIA.

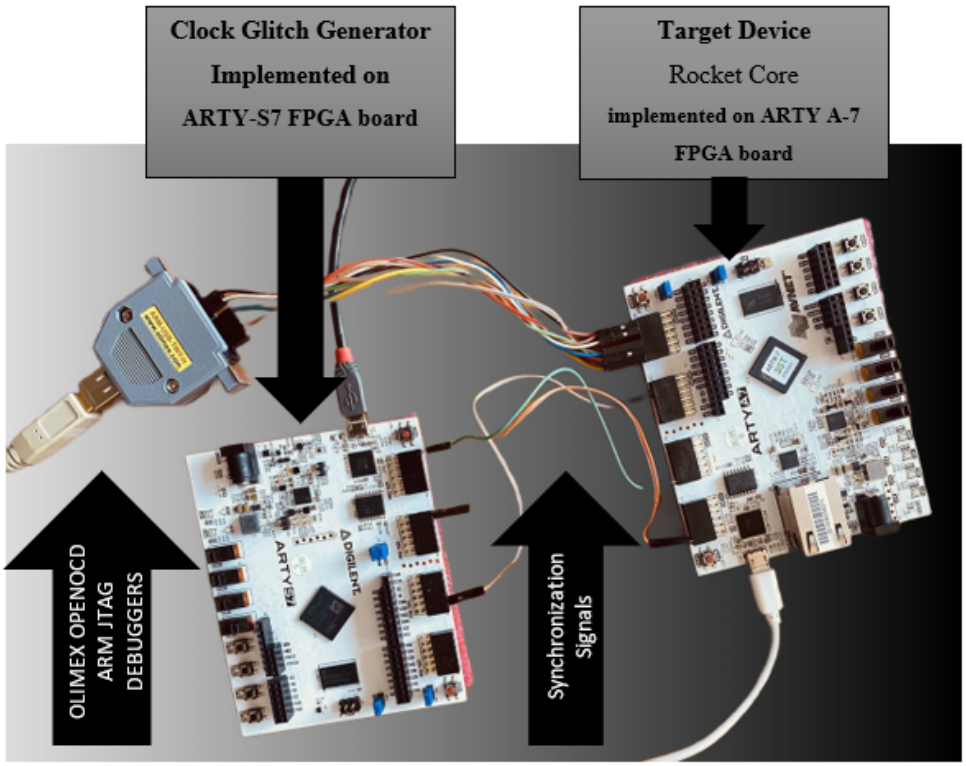

Figure 4.4: The Experimental Setup

Table 4.2 describes the security properties of Authentication and Drug Manager modules and a brief explanation of the potential threats against them. In the following, the vulnerabilities of these modules' certain operations are investigated. In the first scenario, the Authentication module and its related operations that authorize individuals to the system are evaluated. Then, the Drug Manager module is analyzed in the second scenario.

4.2.3.1 Evaluation of the Authentication Module

Sec-Pump has a single-step authentication process. When the Sec-Pump boots, it enters an infinite loop for password entry. If an attacker can bypass this password checking step, he/she can access the entire system. In the following, this module is evaluated against the clock glitching FIA. Listing

Table 4.2: Potential FIA Threats for Sec-Pump

Targeted Module	**Security Properties**	**Under Evaluation Functions against Clock Glitching FIA**
Authentication	Confidentiality-related assets	The Conditional Branch fo user password checking
Drug Manager	Integrity-related assets	Critical Functions to convert or save important values (Cure name and duration)

Listing 4.1: Authentication Module in Sec-Pump Application

```
bool PassCheck (char* r_pass, char* i_pass)
{
  if( strcmp(r_pass, i_pass) == 0)
    {
        printf("welcome to the Sec-Pump\n");
        InsulinController(FloatBuffer);
        return true;
    }
  else
    {
        printf("Password Wrong!\n try again!");
        return false;
    }
}
```

4.1 illustrates a single-step authentication in which the user needs to enter his/her password to enter the application environment. This conditional branch sets the authorized value and jumps to the beginning of the critical control flow of the Sec-Pump with a high level of accessibility. This branch as a critical part in the code has been identified and effort has been performed around it to explore fault injection scenarios to fail the correct execution of it. After synchronizing the target and the clock glitch generator using a trigger signal, in order to evaluate this single-step authentication process, different glitch widths, glitch offsets, and glitch delays have been examined. As explained in Chapter 3, a checkpoint (CP) in Figure 4.5 is used to identify the vulnerability of the conditional branch of this module in the presence of FIA; When the CP equals one, it shows this conditional branch has been corrupted.

The evaluation process has been performed using different glitch widths and glitch offsets. This experiments has been performed 100 times, on different glitch delays which were selected randomly. The glitch map for this attack is illustrated in Figure 4.6, in which red points are the

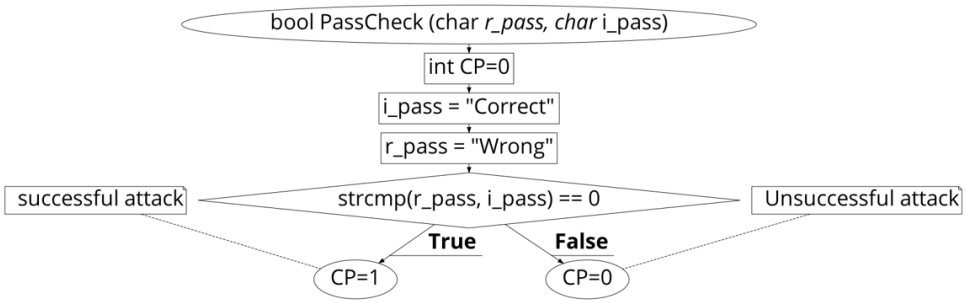

Figure 4.5: Sec-Pump Authentication Evaluation Process

successful faults, and blue points are the configuration that leads to the target reset. The best configurations are obtained when the glitch width = 2,3,4,20,21,22 ns and the glitch offset is between 2 to 11 ns. The results show that the narrow glitches (glitch width < 5 ns) which are located in the beginning of the clock cycles have more success rate. Moreover, the wider glitch widths (20 ns < glitch width < 23 ns) lead to small post glitches (T - T_{glitch}) that cause erroneous function of the processor.

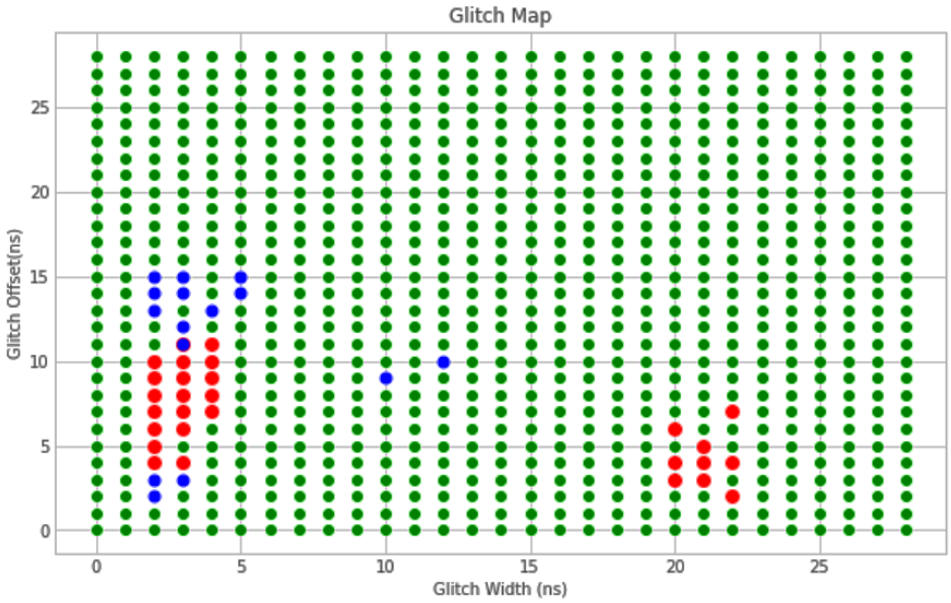

Figure 4.6: Glitch Map for Single-Step Authentication (RED: Successful FIA, BLUE: Target Reset)

4.2.3.2 Evaluation of Drug Manager Module

Drug Management is another critical module that manipulates the central data/control flows of the Sec-Pump application. To evaluate this module, first, the main functions and variables in this module are determined. The Drug Management module's core functions are 1)Create-Cure, 2)Modify-Cure, and 3)Delete-Cure. These three functions operate on three variables: 1)Cure-Name, 2)Cure-Volume, and 3)Cure-Duration. Cure-Volume and Cure-Duration directly impact the Sec-Pump's critical functionalities because they have a direct relation with the time and amount of injected

Listing 4.2: CreatCure Process in Drug Management Module of Sec-Pump Application

```
uint8_t CreateCure(uint8_t * name, uint8_t * volume, uint8_t * duration)
    {
        strcpy(cure.name, name);
        cure.initVolume = atoi(volume);
        cure.volume = atoi(volume);
        cure.initDuration = atoi(duration);
        cure.duration = 0;
        if(cure.initVolume == 0 | cure.initDuration == 0)
          {
            printf("NULL VOLUME AND DURATION\n");
            cure.valid=0;
          }
        else
          {
            cure.valid=1;
          }

      printf("[*] CURE CREATED\n");
        return 1;
    }
```

medicine.Consequently, they need to be protected against any kind of attacks, including FIAs.

First, the Create-Cure part is evaluated and shown in Listing 4.2. It forms a flexible mechanism to initiate a cure with a Cure-Name to receive the inputs (Cure-Volume, and Cure-Duration) and to initialize them.

In this work, the *strcpy* and *atoi* functions are evaluated against clock glitching fault injection attacks by dynamically monitoring the output of each operation. Because there are multiple *atoi* functions in the code, the one that operates on the Volume variable is taken as an example to assess. In the following, these two examples are explained in detail.

- The *strcpy* function copies the string from the name to the Cure-Name variable and returns the copied string. This function has been evaluated with different glitch configurations and as a result, the copied string was either incomplete or wrong. Figure 4.7 shows the evaluation process of a *strcpy* function using a boolean variable named as FAULTSUCCESS and an extra *strcmp* function to compare the copied string. The faulty behavior of *strcpy* results in various problems in the Sec-Pump's expected functionality. For instance, an inappropriate drug may be utilized, which is a dangerous action. Figure 4.8 shows the glitch map of the evaluation of *strcpy* function in which red points are the successful faults (wrong copied string), and blue points are the configuration that leads to the target reset. The best configurations are obtained when the glitch width = 4,5,6 ns and the glitch offset is between 5 to 10 ns. One can

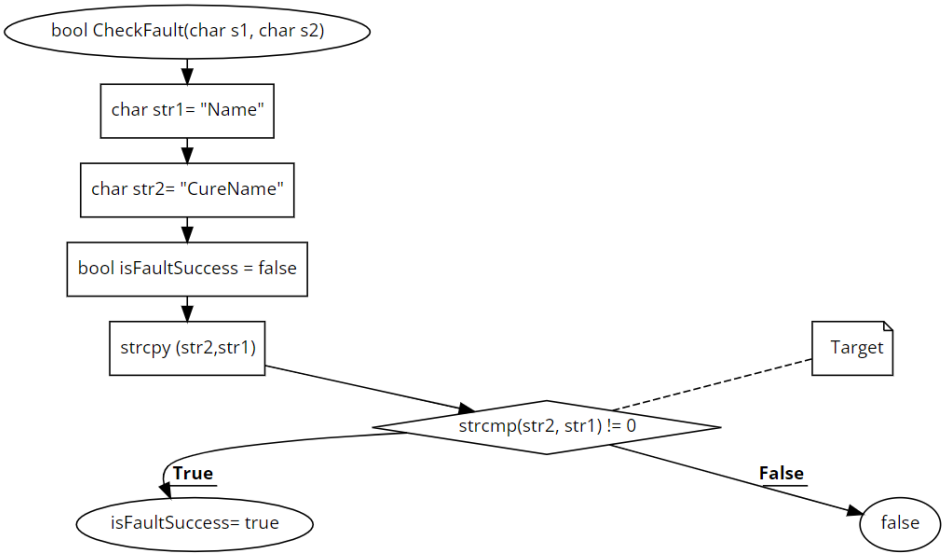

Figure 4.7: Sec-Pump Drug Management Module Evaluation Process (strcpy function)

notice that this function is less vulnerable than the conditional branch in the Authentication module.

- The *atoi* function in the Create-Cure module converts a string, such as entered volume and duration, to a number (specifically an integer). After the evaluation of *atoi* function with different glitch configurations,wrong volume/duration of the drug have been saved inside the system. This vulnerability comes from the repetitive processes that are terminated early by a clock glitching FIA. Figure 4.9 shows the evaluation process of an *atoi* function for volume equal to 65535 using extra *strcpy* and the comparison functions. The best configurations are obtained when the glitch width = 2,3,4 ns and the glitch offset is between 8 to 10 ns. *atoi* function is less vulnerable than *strcpy* function but it can has a more dangerous effect (e.g., wrong drug value) on the Sec-Pump critical functionalities. Figure 4.10 shows the glitch map for the atoi function in which red points are the successful faults (wrong integer value), and blue points are the configuration that leads to the target reset. The vulnerability of atoi is critical in the SecPump application because this function is also used to update and replace the current Cure-Name, Cure-Volume, and Cure-Duration values. As the reference, it takes the value from the timer of the system.

- An additional vulnerability of the Drug Management module originates from the Delete-Cure Function. Delete-Cure, as it is shown in Listing 4.3, is responsible for eliminating a cure and its related data (Cure-Name, Cure-Volume, and Cure-Duration) from memory. Erasing the Cure-name is done by filling its memory block with zeros. Accordingly, a C library function named *memset* has been called to copy 0x0 to the 32 first characters of the memory block where the Cure-Named is pointed to. The *memset* function can be evaluated by following the process in Figure 4.11.The faulty behavior of the *memset* also has been observed

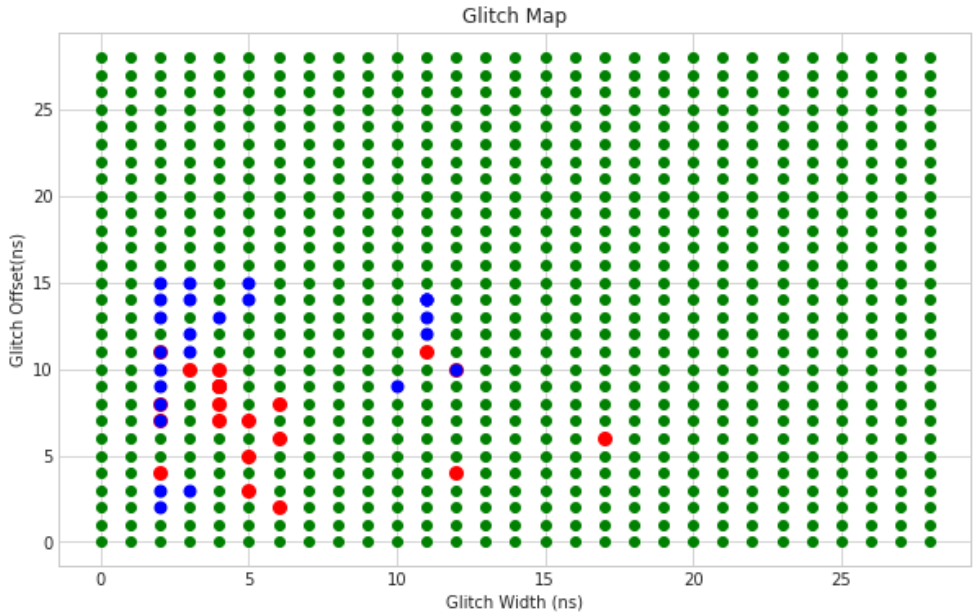

Figure 4.8: Glitch Map for *strcpy* function in Drug Management Module (RED: Successful FIA, BLUE: Target Reset)

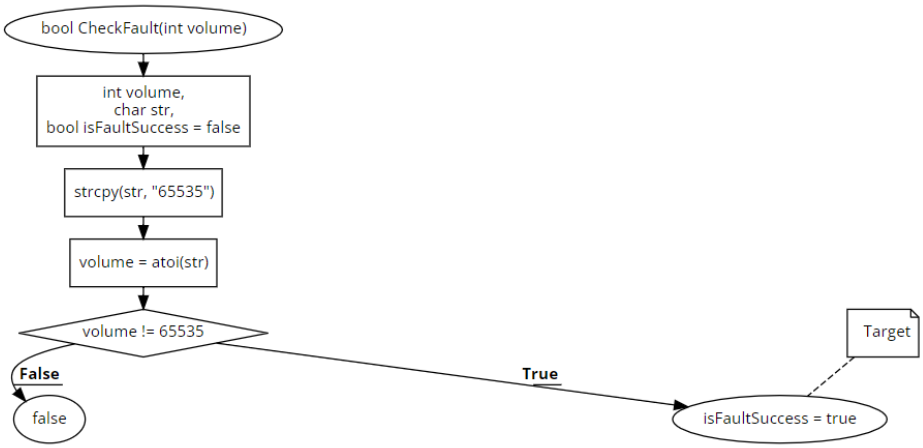

Figure 4.9: Sec-Pump Drug Management Module Evaluation Process (*atoi* function)

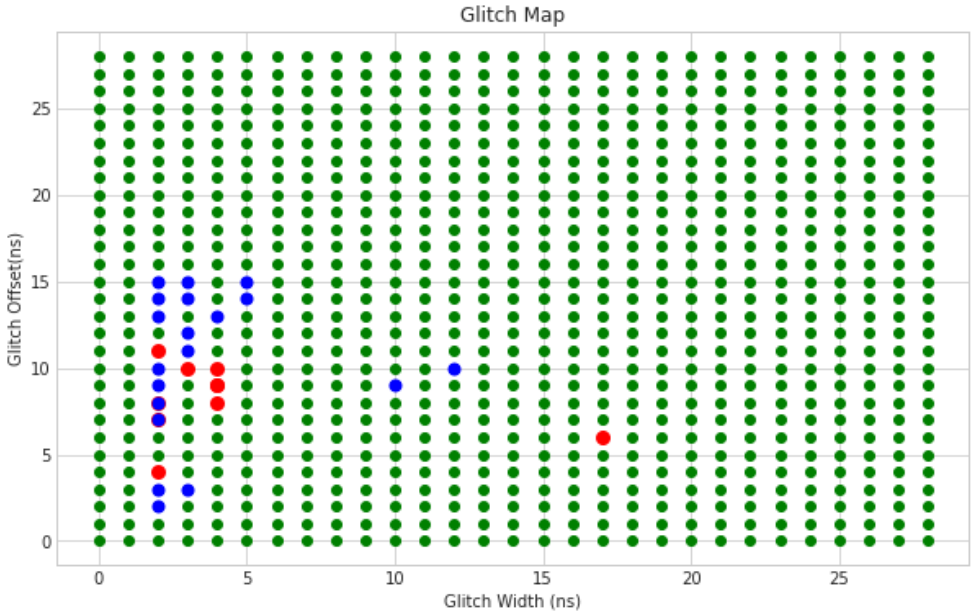

Figure 4.10: Glitch Map for *atoi* function in Drug Management Module (RED: Successful FIA, BLUE: Target Reset)

by loading the memory values from the application's storage and comparing them with the expected results. Figure 4.12 shows the glitch map for the *memset* function. This function is not very vulnerable to the clock glitching and the only configuration for successful attack is obtained when glitch width = 4 ns and the glitch offset= 9 ns. This is because the *memset* function needs access to the memory and the probability of having synchronized fault with the critical assembly instruction level is too low.

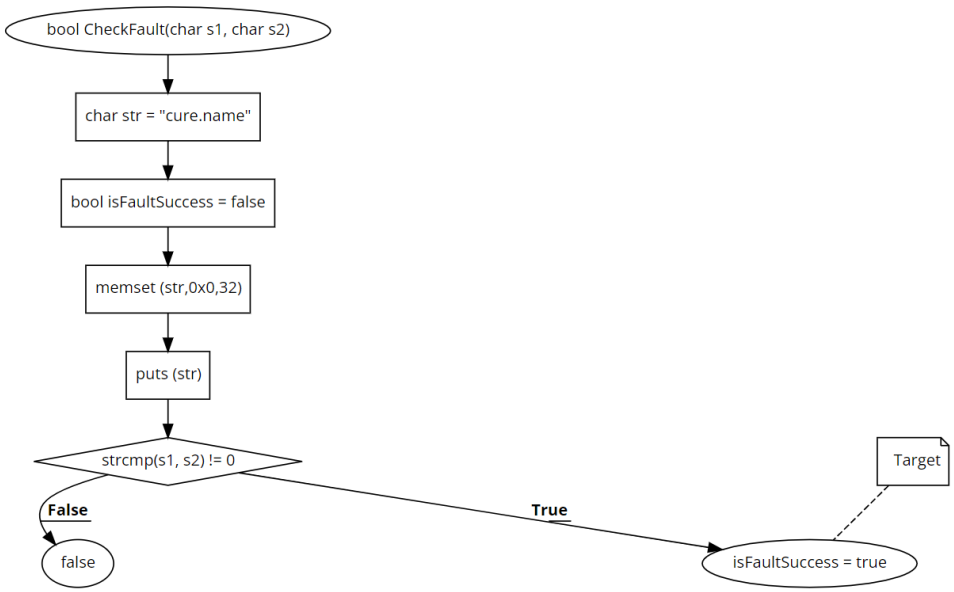

Figure 4.11: Sec-Pump Drug Management Module Evaluation Process (*memset* function)

Listing 4.3: Delete Cure Process in Drug Management Module of Sec-Pump Application

```
uint8_t DeleteCure()
{
  memset(cure.name, 0x0, 32);
  cure.volume = 0;
  cure.initVolume = 0;
  cure.initDuration = 0;
  cure.duration = 0;

  cure.valid=0;

  printf("[*] CURE DELETED\n");
  return 1;
}
```

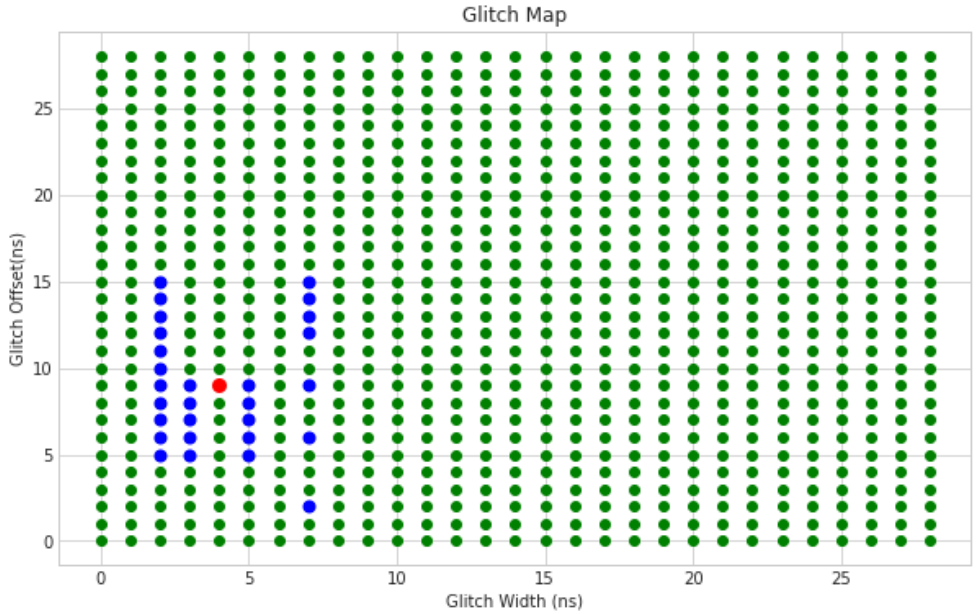

Figure 4.12: Glitch Map for *memset* function in Drug Management Module (RED: Successful FIA, BLUE: Target Reset)

Listing 4.4: The Comparison For Current Cure Duration and Initial Cure Duration In Drug Management Module of Sec-Pump Application

```
if(cure.duration >= cure.initDuration)
  {
    printf("END OF CURE\n");
    DeleteCure();
    return;
  }
```

In addition to the mentioned vulnerabilities of the Drug Management, this module also has single conditional branches to compare the injected drug volume with their final quantity. This comparison has been shown in Listing 4.4 The evaluation of the authentication module shows the vulnerability of these conditional branches that may cause erroneous behavior.

4.2.4 VULNERABILITY MITIGATION FOR THE SEC-PUMP APPLICATION

Previously it was shown that how one can apply FIA and exploit the different Sec-Pump application vulnerabilities. This section addresses some programming patterns to mitigate the revealed vulnerabilities. These patterns assist the embedded developers to reduce the risk of this kind of FIAs. These patterns mainly focus on critical data or program flows of Authentication and Drug Management modules.

4.2.4.1 Fault impact mitigation for the Authentication module

It was shown that the single-step authentication module is vulnerable and may be bypassed by a simple clock glitching FIA. Therefore it is needed to use an alternative for it. Nested conditional statements are more secure than the single step conditions for the authentication process and contain nested decision-making conditional statements. The glitch map in Figure 4.13 Shows that these kind of statements are not vulnerable to a single glitch FIA.

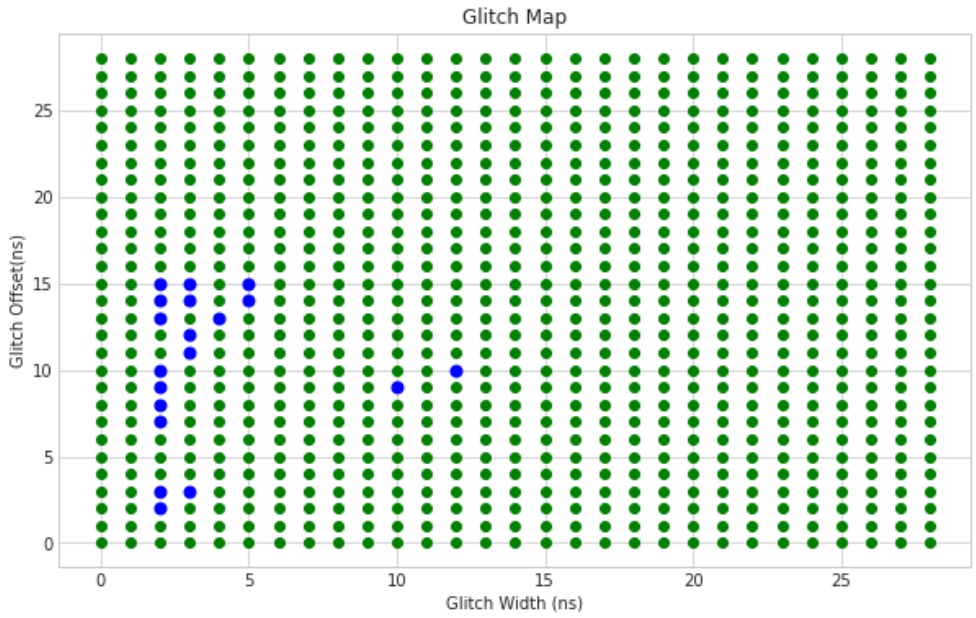

Figure 4.13: Glitch Map for Nested Conditional Authentication (RED: Successful FIA, BLUE: Target Reset)

4.2.4.2 Fault impact mitigation for the Drug-Management module

In this part, some robust software-level alternatives are proposed to mitigate the impact of existing vulnerabilities in different high-level instructions of the Drug-Management module.

One of the vulnerable functions in the Drug-Management module is the strcpy function. Generally, it is used to copy a string from the source to the destination with a null character termination. This operation does not specify the length of the copied string. The strcpy function in the Drug Management module is used to copy "name" to a destination "cure.name". As described in the previous section, the strcpy operation is vulnerable against the clock glitching FIA, meaning that

the copied string in the destination was not equal to the source. This vulnerability originates from the fact that the strcpy operation continues to copy into the destination location until it reaches the null character. In the presence of a clock glitch, the processor may not have enough time to detect the transferred NULL character and continues to copy the string. This is the reason behind the unexpected copied string in the curename of the Sec-Pump. There are different solutions/alternatives to have more secure strcpy type operation, which are described in the following:

- *strncpy* function: *strncpy*(destination, source, size n): This function is similar to the *strcpy* process, except that the first n bytes of the source must be copied. According to our experiments, most of the time, this function was not susceptible against single or multiple clocks glitching FIA. It is less vulnerable than *strcpy* in front of FIA because it is not just based on a single null termination but counts the copied bytes. However, injecting faults that are synchronized with the counting process of *strncpy* can be targeted. Thus, it can not guarantee a complete protection.

- *strlcpy* function: *strlcpy*(destination, source, size n): This function copies a string to a destination buffer, and it takes the destination size as a parameter like what *strlcpy* does. However, it writes a single null byte to the end of the destination. This approach can also guarantee the cases in which *strncpy* copies a null-terminated string to the destination.

The second vulnerable function in the Drug-Management module is *atoi*. This function converts a string into its integer numerical representation. The *atoi* function has been used in the Drug Management module to convert the related ASCII string as an argument (e.g., Volume and Duration values) to integer form. At some point, in the presence of clock glitching FIA, this function returns an undefined integer or zero, which does not correspond to the received argument. This faulty behavior can originate from the fact that this function works iteratively (convert characters from left to right), and in the presence of a clock glitch, it may miss one character and return the first valid number that can be converted from the received string. Also, *atoi* expects a null-terminated string as an input, and a clock glitch can affect the observation of the null character.

There are different solutions/alternatives to have secure type *atoi* operation, which are described in the following:

- Redundant function call: for sensitive data, one can call *atoi* twice and compare the generated integers. This must be done before passing the value to the other functions in the application.

- Adding self-verification code: The *itoa* function complements *atoi*, which converts an integer to an ASCII string. The *atoi* function's result can be sent to an *itoa* function which is called right after that to generate the related ASCII string. Then using a *strcmp* function, the two ASCII strings are compared.

The third vulnerable function is *memset*. This function fills a memory block using a particular value. The starting memory address, the value to be loaded, the number of bytes to be filled must be determined. The *memset* operation is used in the Delete Cure function of the Drug Management module to clear memory blocks. Invalid memory overwrites in the destination's neighbors' blocks are caused by clock glitching FIA, and the copied zeros exceed the assigned memory size. The reason behind it is that *memset* blindly writes to the specified address for the number of specified bytes, regardless of what it might be overwriting. This can distract the other important memory blocks, affecting the standard behavior/data of the Sec-Pump. Regarding *memset*, it is up to the programmer to ensure that only the valid memory is written. There are different solutions/alternatives for more secure memset type operation, which are described in the following:

- making the execution timing unpredictable: in this case, the programmer needs to consider a loop with random delays. Then within the loop, the same functionality for zeroing the destination can be called. Like so, the probability of successful synchronization for clock-glitching FIA is too low, and the other memory blocks are protected against any overwriting.
- Adding self-verification code: In this solution, the programmer can add some self-verification code for *memset* to make sure about the overwriting concerns in the presence of FIA. This can be done by copying the value of the "destination+length" pointer and compare it after the *memset* execution.

4.3 CONCLUSION

This chapter utilized an evaluation approach to identify the most important assets and to assess an embedded application against the clock glitching FIAs. This approach can help to partition a complex application into smaller and more accessible units to evaluate. It can detect the critical security vulnerabilities and highlights the piece of the code that needs to become more robust against the attacks. As a descriptive example of the this methodology, a medical embedded device (Sec-Pump) was studied, and some parts of its code were analyzed. Then, the evaluation approach and the assessment strategy of this chapter was utilized to find the Sec-Pump's vulnerabilities. Using this method, one could perform a practical security risk assessment of existing software-level vulnerabilities and reduce their impacts. In the following, it was demonstrated that this methodology could help to pick up the proper software-level patterns to boost the overall system's security. Regarding this, a few alternatives for high-level C functions and software pattern examples were introduced, which can help the software developer to make the application more resilient against FIA.

The main limitation of this approach is its difficulty in comparing and prioritizing the exploited vulnerabilities inside an application. Moreover, it is burdensome for the software developer to consider all of the FIA parameters to apply an attack for each vulnerable pattern and function. Furthermore, one cannot observe the global impact of the attack in an embedded application and can only notice its local effect. Therefore it is needed to narrow down the FIA parameters and to have a global estimation of the potential vulnerability by FIA. Finally, it is required to have an approach to help the developer to know when to inject the faults, because otherwise, even if one finds the important assets, there are too many possibilities. The discussed limitations and the way to tackle them are the topics for the next chapter.

5 Optimizing the FIA Evaluation Process by Utilizing Simulation-based Analysis and Symbolic Assertion

Contents

This thesis has previously assessed the embedded software against the clock glitching FIAs by utilizing some simplified test routines. Then the vulnerability of some common C-functions and patterns against experimental FIAs was demonstrated. Using the assessment approach one could exploit the software-level vulnerabilities and could observe their impact on a target embedded application (e.g., operational/data flaws, system output/behavior errors, or broken security features/privileges). However, this approach does not consider all of the FIA parameters to apply an attack for each vulnerable pattern or function. Furthermore, by using only constrained input vectors for an embedded software, one is not able to find the corner case vulnerabilities. Regarding this, a pre-injection analysis is needed to optimize the experimental evaluation against the clock glitching FIAs, to narrow down the search space for experimental attack's configurations and to find the most critical points of an embedded software. Accordingly, this chapter considers two

techniques for premeditated vulnerable C-functions and high-level software patterns inside a target application.

The first technique uses the simulation-based results which studies the defined instruction-level fault models, to find the root cause of the security weakness of high-level C-functions. In particular, this approach examines the vulnerabilities of the target embedded software which run on a RISC-V system. Respectively, an open-source and cycle-accurate simulation framework called RIPES performs a simulation-based fault injection campaign on the identified sensitive c-functions. The simulation results are then further exploited to fine-tune the experimental fault injection campaign parameters to reveal the more detailed vulnerabilities within an embedded IoT application.

The second technique in this chapter utilizes the LLVM compiler and its add-on named KLEE tool to evaluate the vulnerability of high-level patterns by using symbolic execution against FIAs. This can benefit the security assessment approach to obtain an overall vulnerability factor of different software blocks of the target application. This methodology has been applied on a Sec-Pump as an example of a secured embedded application.

The proposed approaches in this chapter can help to improve the experimental evaluation by giving a bigger picture of potential vulnerabilities, and consequently can increase the system's security. Therefore, they can result in having better hardware security assessment scenarios for non-security specialists and embedded software developers.

5.1 Enhancing the Experimental FIA Through Simulation-based Pre-Injection Analysis

The evaluation platform was utilized to exploit the potential vulnerabilities of some functions of the Sec-Pump application in chapter.4. Here, a more general test is considered for different C functions from main categories including Type Casting, String Manipulation, Searching and Sorting and Memory based functions.

5.1.1 Non-Exhaustive Experimental Evaluation of C-Functions

The first evaluation approach is to exhaustively test all of the possible discrete combinations of clock glitch parameters for each function. Although our platform can automatically generate all of the possible glitch parameter values, but assuming that a glitch has 2 parameters (glitch width, glitch offset) and each can take 90 values, this exhaustive approach takes too much time. Moreover, depending on the execution time of each function, glitch delay as the third parameter, can take different values (e.g., 100 different cycles). Therefore, the exhaustive evaluation is almost impossible because the target application may have a large number of sensitive functions. Instead of performing a complete test, a subset of glitch parameters combinations were generated. The search space for glitch width and glitch offset is divided into n (here 11) sub-intervals, and then the n-1 (here 10)

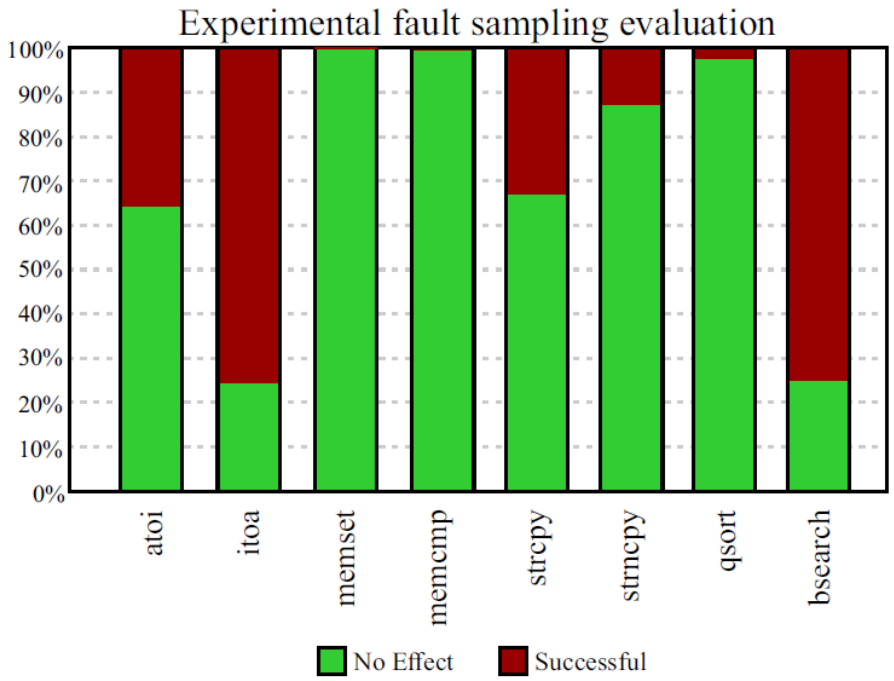

Figure 5.1: Experimental-Based Evaluation Results With Combinatorial Glitch Parameters

produced points were used. In the next step, the generated glitch can be inserted on the random clock cycles during the trigger signal. These selected combinatorial parameters can dramatically reduce the number of experiments.

The experiments in this section have been performed on the same target as chapter.4 including a RISC-V (Rocket Core) processor implemented on ARTY-A7 FPGA. The successful faults are reported in Figure 5.1. These functions have been executed on the same target design as chapter.4 (RISC-V Processor). From Type Casting Functions *atoi* and *itoa* were selected to be evaluated, and they show significant vulnerabilities against experimental attacks. Both *strcpy* and *strncpy* from String Manipulation Functions showed vulnerability by copying the corrupted string to the destinations. *strncpy* is less vulnerable than *strcpy* because it is not just based on a single null termination but counts the copied bytes. In Searching and Sorting functions, the vulnerability of the *bsearch* function has been exploited by the defined fault configurations.

The discussed evaluation process, is highly desirable to exploit a function's vulnerabilities; however, by applying a pre-injection step to find the most vulnerable intervals for the glitch delay parameter, can make the attack more efficient. In the following, the focus is on the fault effects on a RISC-V micro-architecture. A cycle accurate simulator has been utilized to model the fault effects at micro-architectural level and to observe their propagation at the application level.

5.1.2 Fault Effects on A RISC-V Micro-Architecture

The focus in this section is on the RISC-V processor. First, the main characteristics of the basic implementation of this processor are reviewed. Then, an approach is presented to model the ex-

pected fault effects at the micro-architectural level.

5.1.2.1 An Introduction to RISC-V Processor

RISC-V is an open-source Instruction Set Architecture (ISA) developed by the University of California, Berkeley [92]. This thesis targets the Rocket Core implementation of RISC-V [93]. It is composed of a 5-stage pipeline which is demonstrated in Figure 5.2 In this pipeline, there are sequential stages including Instruction Fetch (IF), where instructions are fetched from instruction memory; Instruction Decode (ID), which decodes the instruction, drives control signals, and reads data from the register-file; Execute (EXE), where operations are executed by the ALU; Memory (MEM), which undertakes memory reads and writes; and finally Write-Back (WB), where the results from the previous stages are written into the register-file.

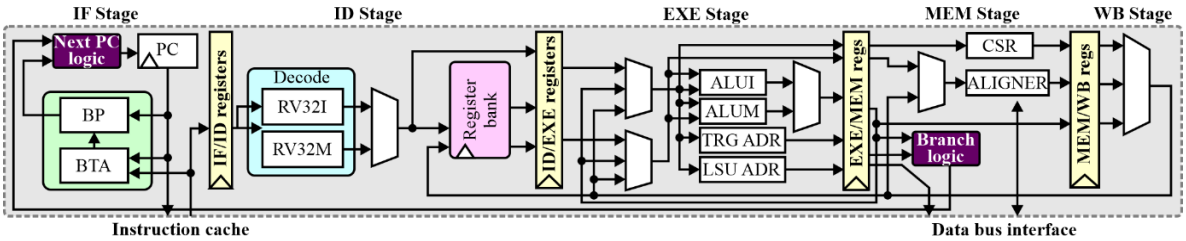

Figure 5.2: A 5-stage RISC-V CPU implementation

5.1.2.2 RISC-V Instruction Formats

There are six core of instruction formats in RISC-V ISA, as shown in Figure 5.3 They all have a fixed 32 bits long and must be aligned on a four-byte boundary in memory.

31 25	24 20	19 15	14 12	11 7	6 0	
fun7	rs2	rs1	fun3	rd	opcode	R-type
immediate[11:0]		rs1	fun3	rd	opcode	I-type
immediate[11:5]	rs2	rs1	fun3	immediate[4:0]	opcode	S-type
immediate[12|10:0]	rs2	rs1	fun3	immediate[4:1|11]	opcode	SB-type
immediate[31:12]				rd	opcode	U-type
immediate[20|10:1|11|19:12]				rd	opcode	UJ-type

Figure 5.3: RISC-V based instruction formats

These instruction formats are explained in the following:

- R-Type instructions perform computation on the value of the two source registers and store the result in the destination register. Examples are arithmetic/logical operations such as xor, add, mul, div.
- I-Type instructions are instructions having an immediate value (12-bit) as one of their source operands. load, jalr, slli, and addi are few examples of I-Type instructions.
- S-Type instructions such as sw and sb are used to store an operand into a destination register by using the base addressing mode. This type of instruction has a source register, an

immediate value as the base address, and a destination register.

- SB-Type instructions are branch instructions based on PC-relative addressing mode such as beq, bge. They need to specify a destination address and two registers to compare.
- U-Type instructions have been designed in RISC-V ISA, in which 32 bits in the instruction format have been dedicated to the immediate value. This format is specifically used for two instructions, namely lui, and auipc.
- UJ-Type is related to the unconditional jumps in which one may jump to anywhere in code memory, such as jal.

5.1.2.3 Instruction-Level Fault Effect Modeling

In the following, the aim is to model the instruction-level fault injection impacts by utilizing the clock glitcher designed in this thesis. The faults originating from the clock glitching attack may either alter the result of combination logic or flip the storage elements (e.g., flip-flops) [6]. These faults can propagate to the micro-architectural level and manifest themselves as registers with random values. One can observe all of the possibilities for micro-architectural level injected fault effects; however, the focus here is on the specific cases in which the content (32-bit) of Instruction Register (IR) is affected.

According to the variations in the detailed architecture of each pipeline stage, the fault impacts on IR will vary from one stage to another. This work concentrates on the fault effects that occur in ID, where a fault can be captured in ID/EXE registers and propagate to the following levels. The fault propagation consequences are modeled as high-level errors at the instruction level depending on the instruction types in Table 5.1. In this Table, different instructions types have been studied with some examples. According to the variations in the detailed architecture of each pipeline stage, the fault impacts on IR will vary from one stage to another. This work concentrates on the fault effects that occur in ID, where a fault can be captured in ID/EXE registers and propagated to the following levels. These fault propagation results in high-level errors at the instruction level depending on the instruction types and include:

- *R-Type*: Wrong RS1 and RS2 data may be generated and propagated as a faulty result (e.g. ADD X2 X3 X4 with wrong X3 and/or X4 value)
- *I-Type*: Wrong RS1 data or immediate value propagate as a faulty result (e.g., ORI X10 X15 50 becomes ORI X10 X15 54)
- *S-Type*: Wrong data in the wrong memory address may be store (e.g., SW X10 -24(X2) becomes SW X8 -16(X6))
- *SB-Type*: Either Wrong RS1 and RS2 data may cause wrong decision in the branches (e.g., BGEU X6 X15 16 becomes BGEU X2 X11 16) or the value of next PC may be corrupted (e.g.,

BGEU X6 X15 16 becomes BGEU X6 X15 24)

- *U,UJ-Type*: The value of the next PC may be corrupted (e.g.JAL X0 0x10504 becomes JAL X0 0x10564)

Table 5.1: Propagated fault effects based on different instruction types

The location of fault manifestation	The fault effects (micro-architectural level)	Ins. types	Propagated fault effects (ISA Level)	Example
Instruction Fetch Stage	The faulty stored PC value	All	Fetching another instruction/invalid data from a wrong address due to PC corruption	Instead of fetching the instruction with the address 110, the instruction with the address 114 is fetched.
	Faulty bit values of the fetched instruction in IF/ID registers		Faulty and unpredictable instruction fetch	Change the instruction operands, for example ADD X10 **X15 X1** becomes ADD X10 **X14 X3**
	Faulty valid-bit		Not executing because of the disabled valid bit	
	Wrong branch predictor decision			
Instruction Decode Stage	Faulty bit values of the register bank's data loaded to store in ID/EXE registers	R	Loading wrong RS1 and RS2 data and propagating a faulty result	One or all of the RS1/RS2 bits cannot be stored, therefore the data will be invalid in the registers.
	Faulty bit values of immediate bits of instructions	I	Loading wrong RS1 data and propagating a faulty result Propagate a faulty result because of the wrong immediate value	One or all of the RS1/RS2 bits cannot be stored, therefore the data will be invalid in the registers. ORI X10 X15 **50** becomes ORI X10 X15 **54**
		S	It is possible to store the wrong data to the wrong memory address	SW **X10 -24(X2)** becomes SW **X8 -16(X6)**
		SB	Loading wrong RS1 and RS2 and making the wrong decision in the branches Corrupted next PC	BGEU **X6 X15** 16 becomes BGEU **X2 X11** 16 BGEU X6 X15 **16** becomes BGEU X6 X15 **24**
		U, UJ	Corrupted next PC	JAL X0 **0x10504** becomes JAL X0 **0x10564**
Execution Stage	Faulty bit values of the result in EXE/MEM registers	R, I	Calculating the wrong memory address for load instructions The calculated data is not stored properly in the registers	In LW X14 **0(X8)**, if X8=10 then the relative address value may be incorrectly stored as **11**. If the result of SUB X2 X4 X2 is **15**, it is stored in the registers incorrectly as **7**.
	Faulty computing in multiplication and division	S	Calculating the wrong relative memory and finally the data is stored in the wrong address.	In SW X10 **-24(X2)**, if X2=24 then the relative address value may be incorrectly stored as **4**.
		SB	A wrong decision in the branches and jump to wrong PC address Corrupted next PC	It is not taken but marks as taken in the register. The next calculated PC is stored incorrectly.
Memory Access Stage	Faulty bit values of the result in MEM/WB registers	except for S &SB	Corrupted final results captured by the register addressed by RD	The final result of AND X2 X4 X2 is not stored correctly in the registers.
Write Back Stage	Write the wrong bit values in the register bank	except for S &SB	Corrupted final results captured by the register addressed by RD	The final result of SLT X2 X4 X2 is not stored correctly in the registers.

5.1.3 SIMULATION-BASED EVALUATION RESULTS

In this sub-section, the goal is to utilize a simulation-based evaluation to detect the micro-architectural fault effects that can be propagated into the application level. This can help to narrow down the fault injection intervals during the execution of high-level functions. Accordingly, one needs a simulation-based analyzer that can mimic the real fault effects in a software environment. Then, one can easily monitor the application state, memory values, and outputs after the fault injection process. Existing simulators are usually based on modeling some or the whole parts of the hardware stack that executes the embedded software. The fault injection attack in a simulation-based environment can be performed by: 1) Tampering with the target hardware model, at compilation time, in order to reproduce a real experimental FIA [94], and 2) Altering the state of the targeted software model at execution time [4].

In this work, RIPES [95] has been selected as an open-source hardware simulator, written in C++ and developed on QT cross-platform. RIPES is based on RISC-V ISA and simulates the execution of each instruction cycle accurately. Using this simulator, this thesis aims to induce faults into the selected instructions and monitor the behavior of high-level functions running on a RISC- V processor. The mentioned instruction-level fault effect models are integrated into the instruction types in Section 5.2.3. At each run, the simulator replaces one correct instruction with the faulty version and executes the function to report the potential high-level vulnerability in the whole function. This work does not consider the other affected instructions in the pipeline and only examines one faulty instruction propagation at a time. The simulation steps to evaluate a function are as follows:

1. The simulation environment must be prepared, and the objective function must be defined (with predefined input values).

2. Immediately after the function execution, a Check Fault function must be considered to evaluate the function's output(s) or the affected values.

3. A trial execution is also needed to specify the starting and ending clock cycle of the function execution. This helps to locate the fault injections cycles accurately. After these three steps, one can run the simulation in which the predefined fault models are injected at a lower level.

The simulation environment helps to monitor the execution process more precisely than the experimental evaluation by having access to the internal registers of the processor, their state, and execution time. Therefore, there are more possible results for the simulation-based evaluation which can be categorized as:

- **No effect**: when the function executes correctly Time out: when the function does not end due to a fault injection, and one has to stop the simulation

- **Target meet**: when the fault injection has reached the target of the attack and is considered as a successful attack

- **PC out of** : when fault injection causes the PC processor to move to an address outside the authorized area of the program

- **Disruption of run time**: when the execution time becomes longer or shorter

The results for the simulation-based evaluation of the example functions are shown in Figure 5.4 The results show that functions such as memscmp and qsort are robust to the proposed instruction-level fault model. On the other hand, atoi, itoa, bsearch are highly affected by fault injection, but the attack success rate is widespread in all time intervals of the function execution time. Moreover, the vulnerabilities of the memset, strcpy, and strncpy functions are exploited at specific intervals. To have optimized successful attacks, one can bound these intervals and choose the glitch parameters according to them.

5.1.4 FINE TUNED EXPERIMENTAL ATTACK

Having observed the results in the previous sub-section, one could notice that functions could be more vulnerable when they were attacked with the proper glitch parameters. This highlights the importance of choosing the attack time intervals. For example, functions such as memset, strcpy, and strncpy are more vulnerable at their initial execution clock cycles (clock cycles 0-40). Therefore the goal has been to re-perform the experimental attack for them by considering this information. Note that, the same (T_{glitch}) configuration has the same value as before, but this work targets all the clock cycles inside these specific time intervals. Figure 5.5 presents the results for such an evaluation, and it verifies the simulation results.

The results show that while the vulnerability of the memset function was not detectable in the previous experiment, now it is reported as 22 %vulnerable. Similarly, it has become easier to exploit the vulnerability of the strcpy (before 32 % vulnerable, after 73 % vulnerable) and strncpy (before 13 %vulnerable, after 34 % vulnerable) functions.

This section has analyzed the FIA vulnerability of some C-Function examples running on a RISC-V processor. This approach improves the timing characterization of the experimental attacks. According to the detected vulnerabilities, one can propose robust alternatives and countermeasures at the instruction level to mitigate the risk of existing vulnerabilities in different C functions. Besides standard C-functions, different high-level patterns should be assessed against such attacks. In the following part of this chapter, a complement approach is introduced to have an overall view of vulnerable patterns in the targeted code.

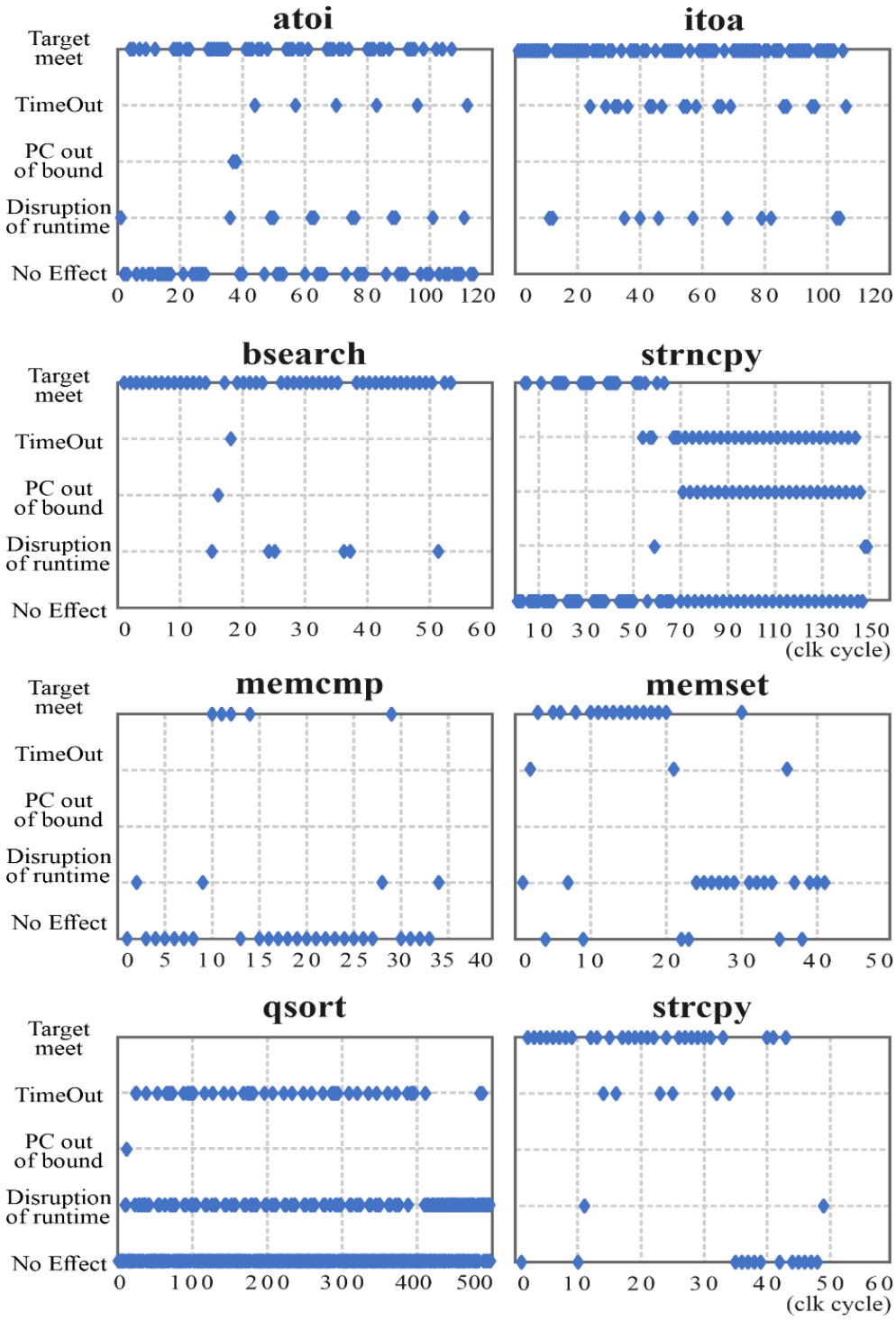

Figure 5.4: Simulation results of different functions

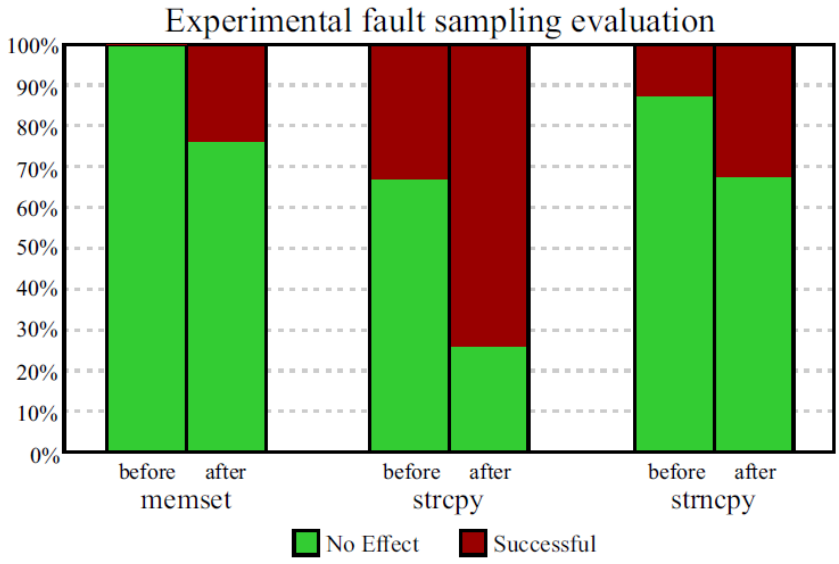

Figure 5.5: Fine Tuned Experimental Evaluation Results for memset, strcpy, and strncpy

5.2 An Offline Hardware Security Assessment Approach using Symbolic Assertion and Code Shredding

Chapter 4 has so far studied the assessment of different high-level patterns against the clock glitching attacks; however, this vulnerability evaluation is considered a local evaluation and is not efficient in practice. Accordingly, a global vulnerability factor is required for each software pattern, which depends on its code location and application. A modular vulnerability code analysis approach based on symbolic assertions can enable one to perform such analysis. In the following, a background of symbolic fault injection and related tools are presented. Then, a precise fault injection method, which utilizes these tools, is explained.

5.2.1 Background of the Symbolic Fault Injection

LLVM is an open-source compiler framework for various projects [96]. This compiler can be utilized for many objectives in the hardware security domain (e.g., automated embedded code reviewing and identifying the vulnerable operations against FIAs) [97]. The LLVM libraries are built on the LLVM Intermediate Representation (LLVM-IR), which is a hardware-independent assembler-like language. This LLVM-IR representation is usually obtained from a C target program using the front-end compiler of LLVM named Clang. In the case of a resilient evaluation against FIA, the symbolic execution can be employed to determine all of the possible execution paths and emulate the program's execution. This can improve the vulnerability identifications usually performed manually by the security analysts regarding the time and workload. KLEE is an LLVM tool that implements the required symbolic virtual machine to support the symbolic execution [98].

There are several resilience evaluation approaches against FIA which are based on symbolic

techniques. For instance, [99] proposes a novel symbolic LLVM-based evaluation framework for resilience evaluation. It employs a KLEE execution engine for the symbolic propagation analysis; however, its main focus is on software-implemented fault injection techniques. Another example is [100] that presents an evaluation implementation that operates at the LLVM-level, and the KLEE symbolic execution engine supports the intermediate code representation. They have used the output results to give appropriate countermeasures.

In the following, LLVM and KLEE are utilized to inject high-level faults and investigate the potential vulnerabilities by symbolic execution. This work emulates the same effect of hardware-oriented FIAs as Section 5.2.4, which can change 32-bit instruction to an unknown value at LLVM-IR pseudo-code. Then, the KLEE tool is used to test a wide range of symbolic input data and report the successful attacks by the detectors.

5.2.2 Precise Fault Injection Using Symbolic Execution

Figure 5.6 illustrates the system architecture in which the LLVM and KLEE tools are used.

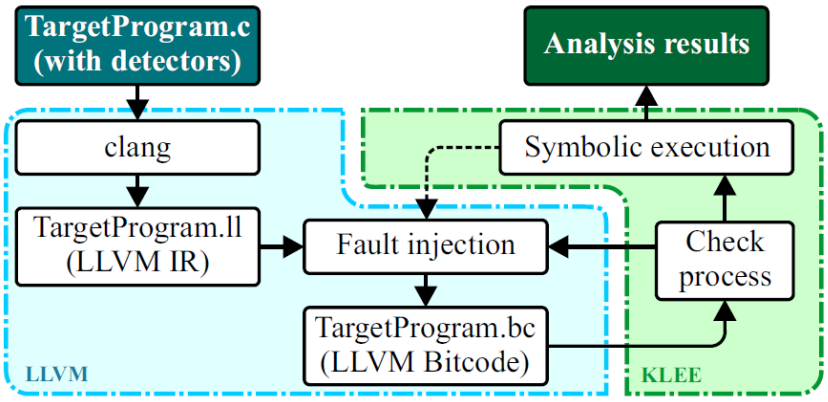

Figure 5.6: The architecture for our approach

To evaluate an application, first, it is partitioned into various code blocks in respect to the functionality and the main variables of each code block. Then, the detection patterns will be inserted into the code in order to report the successful attack. Some pattern examples are listed in the following:

- **Double-Check**: a mechanism that can detect an attack by its conditional re-checking,
- **Loop-Check**: a mechanism in which the loop variable is compared with the expected value,
- **Bypass-Check**: a mechanism that redoes and checks an execution of an important part of the code,
- **Corruption-Check**: a mechanism that is used to detect the hijacking attack by inserting the security variables.

Moreover, using the Klee-make-symbolic function, some sensitive and effective inputs of the program are symbolized. In addition, the Klee-assume (con) function can be used to limit the number of symbol input states, where, i.e., the con is the condition of the symbolic variable data for the C-Language. In this manner, the target program is ready to be given to the Clang. Then, the modified target program is compiled with Clang, and the LLVM-IR file is generated. This file is a platform-independent virtual instruction set. LLVM-IR enables the option that the fault being injected in a layer between the assembly and the C-language.

The impact of fault injection would be non-execution or a change in the operand(s) of an instruction. The change of operand(s) will cause the result not to get stored in the correct memory destination, or a wrong value will be saved. In this way, in each step of the fault injection in the LLVM-IR file, one of the semi-assembly instructions will be manipulated. Afterward, the Clang compiler is used again to convert the fault injected LLVM-IR file into the LLVM bitcode and be transferred to the KLEE unit. The fault injection can deteriorate the whole program execution and therefore, no outputs can be obtained from the KLEE. Regarding this, to reduce the simulation time, first, the program is executed alone, and one must see the results. The correctness of the results is not evaluated in this step, and only reaching the final step of the execution is important. Finally, the simulation is performed with the KLEE tool. In this order, for each fault injection, the program is evaluated with a big group of symbolic input data, and the target program is being evaluated using the integrated detection patterns. If a successful fault is found in the output of KLEE, it would appear in the report of the analysis results, and this process is repeated indefinite cycles.

As a result, one can obtain a parameter that is defined as Vulnerability-Factor. It can be an important parameter to show the security weakness of each partition of the code. Combining this parameter with the local weakness of each function can give a global view of the evaluation results of a complex application. For the sake of clarity, in the following subsection, this approach is explained by using a case study.

5.2.3 A CASE-STUDY

The Sec-Pump project consists of several sections where it is divided into five partitions, including:

- Data receiver: This block is responsible for receiving all the input packets and extracting data from them.

- Packet arbiter: This block is responsible for interpreting the received packet. It has access to the Create, Modify, Delete, Emergency stop, and History functions. This block should be limited in terms of access.

- Run cure: This block is executed periodically and is responsible for injecting the medicine. It also removes the curing process if the permissible doses are injected.

- Update Cure Values: After each injection operation, the volume and duration values must be updated.
- Main update volume: In this block, based on the patient's previous history and blood sugar levels, the experimental parameters are determined to be injected at the next time. This part is important because it runs periodically and is related to the patient's health.

The detection patterns are inserted in each one of Sec-Pump's partitions to prepare the code for the next steps. Listing 5.1 shows a piece of the data receiver code. In this partition, the duration code is extracted from the received data packet. Note that, three detection patterns Loop-check, Corruption-check, and Double-check, are used inside it. In this example, two' if decision' modes are examined by the Double-check pattern. Moreover, there is a loop in this example that has a break; therefore, an additional loop counter is used to perform the Loop-check. Finally, a variable is defined as the tmpFlow variable which is used to apply the corruption check pattern. It should have a constant value of 380, and other values can pinpoint a successful attack. This prepared code is compiled with clang and becomes quasi-assembly code which is ready to inject faults.

Listing 5.2 shows a piece of the compilation of the main program in the LLVM reference language. It is related to line codes 8 to 24 in Listing 5.2. In this block of code, which starts with address 25 and ends with address 46, Zext is used to define a variable, and line 28 defines a variable called i8 by the typing int32, in which the value of line 27 will be loaded. Note that the order of the instructions, which start with %, is not in accordance with Listing 5.1.

The fault injection process is automatically applied to the compiled Clang output and differs based on the instructions. It causes a change in operands or a change in an instruction to skip that instruction or make it ineffective. For example, line 29 is to compare the value of line 28 (variable i8) with the constant value of 35. If they are equal, the output value would be' 1'; otherwise, it would be' 0'. Here, the fault injector manipulates the value of 35 to an unknown value. Therefore, the loop will be potentially executed more cycles than the defined loop parameter, and consequently, the Loop-check pattern will detect it.

5.2.4 Experiments and Results

In this section, the simulation results of the Sec-Pump under evaluation with KLEE are presented. First, the program is tested from an execution point of view, and for this purpose, the check process step evaluates the fault injected code with normal conditions. If the injected faults do not corrupt the program and no bug is reported, the KLEE is used for the main simulation purpose in the next step. In each run-time, the KLEE assigns new outputs to the variables of symbolic software. If a bug is detected, then with the usage of the Klee-assert function, the execution of the program stops, and a report of a successful attack is presented. At the end of the simulation process, the output data is sent to the output, and the evaluation process is continued until a database of fault injection result is obtained.

Listing 5.1: A piece of the data receiver code

```
uint8_t counter = 0; //Loop check variable
int32_t tmpFlow = 0; //Flow check variable
...
tmpFlow += 40;
counter = 0;
for (i = 0; i < 35; i++, packetPtr++){
  if (*(packetPtr) == '\0'){
    packetPtr++;
    //[double check]
    if (*(packetPtr) != '\0'){
      printf("DC flag in block 1-1\n");
      klee_assert(0);
    }
    break;
  }
  else {
    //[double check]
    if (*(packetPtr) == '\0') {
      printf("DC flag in block 1-2\n");
      klee_assert(0);
    }
  }
  *(durationLocation + i) = *(packetPtr);
  counter++;
}
tmpFlow *= 2;
//Some important C functions
strcpy(durationValue, durationLocation);
tmpFlow -= 125;
if (counter != i | *(packetPtr-1) != '\0' | counter >= 35) { //[loop check]
  printf("LC flag in block 1\n");
  klee_assert(0);
}
...
if (tmpFlow != 380) { //[Flow check]
  printf("FC flag in block 1\n");
  klee_assert(0);
}
```

Listing 5.2: The semi LLVM language output.

```
 25:
  %26 = load i8*, i8** %12, align 8
  %27 = load i8, i8* %26, align 1
  %28 = zext i8 %27 to i32
  %29 = icmp eq i32 %28, 35
  br i1 %29, label %30, label %40
30:                                   ; preds = %25
  %31 = load i8*, i8** %12, align 8
  %32 = load i8, i8* %31, align 1
  %33 = zext i8 %32 to i32
  %34 = icmp ne i32 %33, 35
  br i1 %34, label %35, label %37
35:                                   ; preds = %30
  %36 = @printf("DC flag in block 1-1\n")
  br label %37
37:                                   ; preds = %35, %30
  %38 = load i8*, i8** %12, align 8
  %39 = (calculating a memory address)
  store i8* %39, i8** %12, align 8
  br label %62
40:                                   ; preds = %25
  %41 = load i8*, i8** %12, align 8
  %42 = load i8, i8* %41, align 1
  %43 = zext i8 %42 to i32
  %44 = icmp eq i32 %43, 35
  br i1 %44, label %45, label %47
45:                                   ; preds = %40
  %46 = @printf("DC flag in block 1-2\n")
  br label %47
```

Table 5.2 presents the simulation results for injecting 46 faults into the LVMM-IR code of the Sec-Pump. Overall, the KLEE has 109551 execution paths in our simulations, in which, on average, 2381 control-flow paths exist for each simulation. Next, the vulnerability factor is obtained, and it was depicted that with the increase of the number of the fault injection, the preciseness of the Vulnerability factor goes up.

Table 5.2: The calculation of vulnerability factor for Sec-Pump's software blocks

Block	Integrated Detection Patterns								Undetected Attacks	Vulnerability factor
	Double Check		Loop Check		Corruption Check		Bypass Check			
	DEL	DSA	DEL	DSA	DEL	DSA	DEL	DSA		
Data Receiver	1289	13991	0	18157	55	3709	21	1368	6945	0.577
Packet arbiter	2510	3579	-	-	102	2040	10	1464	10252	0.11
Run cure	54	907	-	-	79	1820	28	257	1984	0.046
Update Cure Values	125	5289	-	-	124	1369	78	273	3968	0.107
Main Update volume	5812	6897	1102	1387	548	2028	-	-	9925	0.16

DEL: Detected but Effectless Attacks DSA: Detected and Successful Attacks

The experimental results show that after injecting 40 uniform faults in the LLVM-IR code of the Sec-Pump, the vulnerability-factor tends to toward a value that even the increasing of the number of fault injection attack does not impact it. The results of the Vulnerability-factor reported in Table. 5.3 are the ratio of the number of successful attacks for each block in respect to the whole program. The implemented detection patterns were able to detect 69.7% of the injected faults. Note that it is possible that some of the injected faults are not detected, and it would be needed to utilize another type of pattern to improve the evaluation accuracy. Moreover, it is also possible that some faults are not detected as some injected faults may not have any impact.

Figure 5.7 shows the percentage of the successful attacks for each block considering the utilized detection patterns. In the data receiver block, the loop check pattern has the highest value of detection because this block has many execution iterations. Since this block is the main input packet of the whole program, having faults here results in incorrect execution of the whole program.

It can be observed that this block has the highest number (57.6%) of Vulnerability- Factor in comparison with other blocks. The arbiter block has the most undetected attacks due to its access limitations. On the other hand, it includes a high percentage of detection (11%) mostly by the Double-check pattern due to having many conditional branches. The Run Cure block is a critical section in directing the flow of the Sec-Pump program. Therefore, one could see the most detected faults by the Corruption-check pattern. Update Cure Values and Main Update Volume blocks use the previously stored values for their executions. Hence, the Double-check pattern is the most efficient pattern to detect faulty executions.

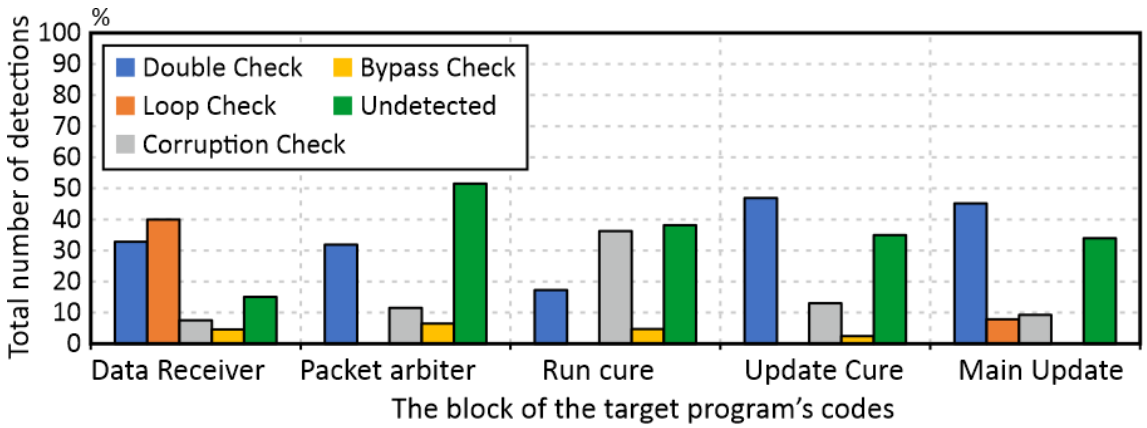

Figure 5.7: The percentage of the successful attacks for each block considering our utilized detection patterns

5.3 CONCLUSION

This chapter has further analyzed the optimized mechanisms of fault injection attacks on embedded software. The attack mechanisms are categorized based on the C-functions and patterns. A mixture of simulation and experimentation has been selected to detect the security flaws against the clock glitching FIA. The simulation operates on defined fault models as fault injection attacks into the RISC-V micro-architecture. It could help to find the detailed mechanism of fault effect propagation in the chosen instruction level. This enabled to improve the FIA timing characterization in the experimental attacks. Therefore, the experimentation attack could give more detail and more precise results.

Utilizing the applied fault models, this thesis has then analyzed the vulnerability of software patterns by using the symbolic execution. In this approach, the software is divided into easy to analysis blocks, and some assertions have been added to each block. This results in obtaining an evaluation parameter named the vulnerability factor. This parameter can be an efficient criterion to evaluate all the corner case vulnerabilities of software blocks against FIA. The obtained results on the Sec-Pump blocks could show the potential risks of overlooking such vulnerabilities.

6 CONCLUSIONS

There has been significant growth in the design and usage of embedded IoT devices in day-to-day lives. The IoT devices, such as the sensors and actuators, are generally easily accessible, and therefore they are attractive targets for adversaries to tamper with. This can lead to revealing or corrupting the data from unauthorized sources. The critical functionalities of many of the IoTs and their quality of data clearly state the importance of hardware security attacks. For example, a tampered wearable medical IoT that collects health-related information is a crucial and critical application. Consequently, both IoT manufacturers and embedded software designers must consider hardware attacks as a serious concern and balance the product between its time to market and security. In many cases, the product's security can only come with adequate software-level evaluations. Therefore, the burden of providing proper tools and evaluation mechanisms is left to the hardware security specialist, which has been the main scope of this thesis work.

In Chapter 1, the context of the IoTs and different hardware security attacks has been investigated. It was exhibited that non-invasive fault injection attacks, such as voltage or clock glitching, are more threatening for critical but low-cost and constrained targets. There are several reasons behind that, including 1) Non-invasive attacks do not require any physical tampering, 2) These attacks can be reproduced and updated by using low-cost and easy-to-access equipment, even in a small laboratory and by mid-level knowledge attackers, and 3) They have proven that a high success rate can be achieved in a short time. Consequently, the embedded software evaluation against non-invasive attacks became the main subject of further research in the thesis.

Chapter 2 reviewed the state-of-the-art clock and voltage glitch generators and described their important characteristics such as complexity, cost, required knowledge, and hardware dependability. It has been observed that the existing evaluation platforms lack the proper configuration characteristics, such as being easy to use for embedded software developers. Accordingly, the specifications of an efficient and low-cost fault injection platform have been acquired. It has also been stated that, besides the evaluation platform, a guideline is required for non-hardware-security experts to detect the system's vulnerabilities.

Chapter 3 has focused on the clock glitching attack and proposed a new hardware evaluation platform to analyze the embedded software vulnerabilities. This platform consists of three main components: 1) Fault Configuration Interface, 2) Fault Generator, and 3) Fault Effects Analyzer. First, a configurator interface defines a search space for different FIA parameters. Afterward, two

different clock glitch generator architectures based on CSC and CDFC approaches have been implemented and compared. Moreover, an example of applying clock glitching FIA on the AES algorithm was presented to validate the efficiency of implemented fault generators. Then, different high-level analysis methods were proposed to help the software developer eliminate the existing vulnerabilities against these hardware-based security attacks. The work revealed that having only an evaluation platform and a high-level analyzer is not enough for the non-security specialists, and a systematic evaluation approach is needed.

Chapter 4 defined and applied a methodized approach named ICEM on evaluating an embedded application. This approach can outline an IoT application's assets and detect its vulnerabilities in front of the fault injection attacks. It can help one to define the important system values and functions, discover the vulnerabilities, determine the security risks, and illustrate the probability and consequences of the potential successful attacks. Finally, the evaluation framework is applied to a medical IoT pump (Sec-Pump) as a descriptive example of critical embedded IoTs. The Sec-Pump has been targeted by FIA and been analyzed step by step to detect where the vulnerability of each important function is originating from.

The lack of a precise FIA setting parameter has led to Chapter 5, where two different approaches were proposed to improve the FIA results. They can optimize the hardware security assessment scenarios for non-security specialists and embedded software developers. The first approach is based on getting help from FIA simulation results, where an open-source and cycle-accurate simulator named RIPES has been used to fine-tune the experimental fault injection campaign parameters. This results in revealing more vulnerabilities within an embedded IoT application. In particular, the vulnerabilities of the target embedded software running on a RISC-V-based system have been observed. The evaluation results could help to find the instruction level fault propagations that cause software level corruptions. The second approach was based on the symbolic executions, where a global vulnerability factor of different software blocks of the target application can be obtained. This methodology has been applied on Sec-Pump as an example of a secured embedded application.

Overall, this thesis work has proposed and implemented a complete assessment structure to be used by embedded designers to evaluate the embedded systems against FIA. The proposed framework in this thesis includes an open-source implemented hardware platform, a methodology to identify the critical assets, an analysis approach of the potential functions, and a simulation utilization to apply precise FIA. The results indicated that this thesis's framework could efficiently and accurately evaluate the embedded IoT applications against fault injection attacks.

7 PERSPECTIVES

Having an evaluation framework proposed in this thesis, there could be some ideas to extend this work in the future. One would be to perform a more comprehensive review of the non-invasive and low-cost semi-invasive attacking tools and to propose the most appropriate assessment approaches in accordance with the desired security level. This gets more important as the adversaries may gain access to more powerful attacking tools such as electromagnetic fault injectors.

Next, this thesis presented some primary high-level and modular assessment methods for the embedded software developers which can be improved and become more advanced. For instance, one can consider not only the individual assets and examine the effects of high-level effects but can also fully simulate the results of an actual attack. Accordingly, more accurate experiments can be performed by integrating other control/ data flow evaluations and by making the quality of the security testing as resourceful as possible.

Another possible improvement to the works of this thesis is to automate the configurator interface in order to apply all of the possible glitch parameters. This can help the developers to save the processing resources and time for hardware security testing. Accordingly, Artificial Intelligence can embrace the responsibility of configuring the automated evaluation tool and assure high testing coverage. AI can also help to create a detailed report on the existing vulnerabilities and to provide an overview of the most sensitive parts of the application. So, perspective work can employ AI to improve the speed, transparency, and time efficiency of the hardware security assessments.

Last by not least, in accordance with the vulnerabilities which can be detected by utilizing the proposed framework in this thesis work, one can design software-based methods at different abstract levels such as in assembly or higher levels of the software. This would lead to mitigating the effects of injected faults by the clock glitching FIA. These countermeasures can be designed by using complementary information about software-level or RTL level fault models. Finally, the evaluation framework in this thesis can verify the effectiveness of the designed countermeasures against experimental attacks.

8 PUBLICATIONS

- **Journal Papers:**

  Z. Kazemi, D. Hely, M. Fazeli, and V. Beroulle, "A Review on Evaluation and Configuration of Fault Injection Attack Instruments to Design Attack Resistant MCU-Based IoT Applications", Electronics, vol. 9, no. 7, p. 1153, Jul. 2020, doi: 10.3390/electronics9071153.

- **International Conferences, Workshops and Presentations:**

  Z. Kazemi, M. Fazeli, D. Hely, and V. Beroulle, "Hardware Security Vulnerability Assessment to Identify the Potential Risks in A Critical Embedded Application", presented at the 2020 IEEE 26th International Symposium on On-Line Testing and Robust System Design (IOLTS), Napoli, Italy, Italy, Jul. 2020. doi: 10.1109/IOLTS50870.2020.9159739.

  Z. Kazemi, A. Norollah, A. Kchaou, M. Fazeli, D. Hely, and V. Beroulle," An in-depth vulnerability analysis of RISC-V micro-architecture against fault injection attack", International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFTS), 2021.

  Z. Kazemi, A. Norollah, M. Fazeli, D. Hely, and V. Beroulle," An Offline Hardware Security Assessment Approachusing Symbol Assertion and Code Shredding ", 23rd International Symposium on Quality Electronic Design (ISQED'22), Apr 2022, Virtual event, United States.

  Z. Kazemi, A. Papadimitriou, D. Hely, M. Fazcli, and V. Beroulle," Hardware Security Evaluation Platform for MCU-Based Connected Devices: Application to Healthcare IoT", in 2018 IEEE 3rd International Verification and Security Workshop (IVSW), Costa Brava, Jul. 2018, pp. 87-92.

  Z. Kazemi et al.," On a Low-Cost Fault Injection Framework for Security Assessment of Cyber-Physical Systems: Clock Glitch Attacks", in 2019 IEEE 4th International Verification and Security Workshop (IVSW), Rhodes Island, Greece, Jul. 2019, pp. 7–12.

  A. Nejat,**Z. Kazemi**, V. Beroulle, D. Hely and M. Fazeli, "Restricting Switching Activity Using Logic Locking to Improve Power Analysis-Based Trojan Detection", 2019 IEEE 4th International Verification and Security Workshop (IVSW), 2019, pp. 49–54.

  Z. Kazemi, C. Bresch, D. Hely, and V. Beroulle," A Systematic Approach for Hardware Security Assessment of Secured IoT Applications", presented at the TRUDEVICE 2020: Workshop on

Trustworthy Manufacturing and Utilization of Secure Devices, DATE, March. 2020.

Z. Kazemi, "Hardware Security Evaluation of Secured Embedded Applications", Ph.D. Forum in Design, Automation, and Test in Europe (DATE) Conference 2021.

Zahra Kazemi, David Hély, Vincent Beroulle, "Hardware Security Assessment Methodology Applied on an Embedded Application: A Case Study on a Medical IoT", Hardwear.io Security Conference, Netherlands 2020.

Zahra Kazemi," Hardware Security Evaluation of Embedded Applications Against Fault Injection Attack", JAIF2021 : Journée thématique sur les attaques par injection de fautes, Sep.2021, Paris, France.

- **Article in submission progress:**

  Z. Kazemi, A. Norollah, M. Fazeli, D. Hely, and V. Beroulle" Build Up a Data Base on Clock Glitch Configurations for Embedded ApplicationAssessment against Fault Injection Attacks"

- **Other publications:**

  A.Norollah, **Z. Kazemi**, H. Beitollahi and D. Hely, "Hardware Support for Efficient and Low-power data Sorting in Massive Data Application: The 3D Sorting Method", in IEEE Consumer Electronics Magazine, doi: 10.1109/MCE.2021.3076979.

  A. Norollah, **Z. Kazemi** and D. Hely, "3D-Sorter: 3D Design of a Resource-Aware Hardware Sorter for Edge Computing Platforms Under Area and Energy Consumption Constraints", 2020 IEEE Computer Society Annual Symposium on VLSI (ISVLSI), 2020, pp. 42-47.

  A. Norollah, **Z. Kazemi**, N. Sayadi, H. Beitollahi, M. Fazeli and D. Hely, "Efficient Scheduling of Dependent Tasks in Many-Core Real-Time System Using a Hardware Scheduler," 2021 IEEE High Performance Extreme Computing Conference (HPEC), 2021, pp. 1-7.

BIBLIOGRAPHY

[1] B. D. Davis, J. C. Mason, and M. Anwar, "Vulnerability studies and security postures of iot devices: A smart home case study," *IEEE Internet of Things Journal*, vol. 7, no. 10, pp. 10 102–10 110, 2020.

[2] S. Lavanya, G. Lavanya, and J. Divyabharathi, "Remote prescription and i-home healthcare based on iot," in *2017 International Conference on Innovations in Green Energy and Healthcare Technologies (IGEHT)*, 2017, pp. 1–3.

[3] H. Ramalingam and V. Venkatesan, "Conceptual analysis of internet of things use cases in banking domain," in *TENCON 2019 - 2019 IEEE Region 10 Conference (TENCON)*, 2019, pp. 2034–2039.

[4] M. Kooli and G. Di Natale, "A survey on simulation-based fault injection tools for complex systems," in *2014 9th IEEE International Conference on Design Technology of Integrated Systems in Nanoscale Era (DTIS)*, 2014, pp. 1–6.

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

[6] Z. Kazemi, D. Hely, M. Fazeli, and V. Beroulle, "A review on evaluation and configuration of fault injection attack instruments to design attack resistant mcu-based iot applications," *Electronics*, vol. 9, no. 7, p. 1153, Jul 2020. [Online]. Available: http://dx.doi.org/10.3390/electronics9071153

[7] Z. Kazemi, A. Papadimitriou, D. Hely, M. Fazcli, and V. Beroulle, "Hardware security evaluation platform for mcu-based connected devices: Application to healthcare iot," in *2018 IEEE 3rd International Verification and Security Workshop (IVSW)*, 2018, pp. 87–92.

[8] C. Bresch, "Secpump," https://github.com/r3glisss/SecPump, 2017.

[9] ——, "Approaches, Strategies, and Implementations of Memory Safety Defenses in Critical and Constrained Embedded Systems," Theses, Université Grenoble Alpes [2020-....], Oct. 2020. [Online]. Available: https://tel.archives-ouvertes.fr/tel-03118575

[10] G. Acar, D. Y. Huang, F. Li, A. Narayanan, and N. Feamster, "Web-based attacks to discover and control local IoT devices," in *Proceedings of the 2018 Workshop on IoT Security and*

Privacy. ACM, pp. 29–35. [Online]. Available: https://dl.acm.org/doi/10.1145/3229565.3229568

[11] G. Mullen and L. Meany, "Assessment of buffer overflow based attacks on an IoT operating system," in *2019 Global IoT Summit (GIoTS)*, pp. 1–6.

[12] A. Barenghi, G. Bertoni, E. Parrinello, and G. Pelosi, "Low voltage fault attacks on the RSA cryptosystem," in *2009 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*. IEEE, pp. 23–31. [Online]. Available: http://ieeexplore.ieee.org/document/5412860/

[13] H. Choukri and M. Tunstall, "Round reduction using faults," pp. 13–24.

[14] H. Martin, T. Korak, E. S. Millan, and M. Hutter, "Fault attacks on STRNGs: Impact of glitches, temperature, and underpowering on randomness," vol. 10, no. 2, pp. 266–277. [Online]. Available: http://ieeexplore.ieee.org/document/6965651/

[15] J. Deogirikar and A. Vidhate, "Security attacks in IoT: A survey," in *2017 International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC)*. IEEE, pp. 32–37. [Online]. Available: http://ieeexplore.ieee.org/document/8058363/

[16] Y. Yang, L. Wu, G. Yin, L. Li, and H. Zhao, "A survey on security and privacy issues in internet-of-things," *IEEE Internet of Things Journal*, vol. 4, no. 5, pp. 1250–1258, 2017.

[17] Y. Lu and L. D. Xu, "Internet of things (iot) cybersecurity research: A review of current research topics," *IEEE Internet of Things Journal*, vol. 6, no. 2, pp. 2103–2115, 2019.

[18] T. Borgohain, U. Kumar, and S. Sanyal, "Survey of security and privacy issues of internet of things," 2015.

[19] Y. Zhou and D. Feng, "Side-channel attacks: Ten years after its publication and the impacts on cryptographic module security testing," 2005, zyb@is.iscas.ac.cn 13083 received 27 Oct 2005. [Online]. Available: http://eprint.iacr.org/2005/388

[20] P. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," in *Advances in Cryptology âĂŤ CRYPTO' 99*, M. Wiener, Ed. Springer Berlin Heidelberg, pp. 388–397.

[21] A. Said, "Measuring the strength of partial encryption schemes," in *IEEE International Conference on Image Processing 2005*, vol. 2, 2005, pp. II–1126.

[22] "TEMPEST: A signal problem." [Online]. Available: https://www.nsa.gov/portals/75/documents/news-features/declassified-documents/cryptologic-spectrum/tempest.pdf

[23] P. C. Kocher, "Timing attacks on implementations of diffie-hellman, RSA, DSS, and other systems," in *Advances in Cryptology, CRYPTO '96*, N. Koblitz, Ed. Springer Berlin Heidelberg, pp. 104–113.

[24] L. Dureuil, M.-L. Potet, P. Choudens, C. Dumas, and J. Clédière, "From code review to fault injection attacks: Filling the gap using fault model inference," in *Revised Selected Papers of the 14th International Conference on Smart Card Research and Advanced Applications - Volume 9514*, ser. CARDIS 2015. Berlin, Heidelberg: Springer-Verlag, 2015, pp. 107 – 124. [Online]. Available: https://doi.org/10.1007/978-3-319-31271-2_7

[25] F. Koeune and F.-X. Standaert, *A Tutorial on Physical Security and Side-Channel Attacks*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 78–108. [Online]. Available: https://doi.org/10.1007/11554578_3

[26] M. Kasuya, T. Machida, and K. Sakiyama, "New metric for side-channel information leakage: Case study on em radiation from aes hardware," in *2016 URSI Asia-Pacific Radio Science Conference (URSI AP-RASC)*, 2016, pp. 1288–1291.

[27] P. Gu, D. Stow, R. Barnes, E. Kursun, and Y. Xie, "Thermal-aware 3d design for side-channel information leakage," in *2016 IEEE 34th International Conference on Computer Design (ICCD)*, 2016, pp. 520–527.

[28] F.-X. Standaert, *Introduction to Side-Channel Attacks*. Boston, MA: Springer US, 2010, pp. 27–42. [Online]. Available: https://doi.org/10.1007/978-0-387-71829-3_2

[29] E. Bursztein, "Hacker's guide to deep-learning side-channel attacks: the theory," 2021, accessed: 2021-09-30. [Online]. Available: https://elie.net/blog/security/hacker-guide-to-deep-learning-side-channel-attacks-the-theory/

[30] M. Randolph and W. Diehl, "Power side-channel attack analysis: A review of 20 years of study for the layman," *Cryptography*, vol. 4, no. 2, p. 15, May 2020. [Online]. Available: http://dx.doi.org/10.3390/cryptography4020015

[31] N. Beringuier-Boher, K. Gomina, D. Hely, J.-B. Rigaud, V. Beroulle, A. Tria, J. Damiens, P. Gendrier, and P. Candelier, "Voltage glitch attacks on mixed-signal systems," in *2014 17th Euromicro Conference on Digital System Design*, 2014, pp. 379–386.

[32] D. Karaklajić, J.-M. Schmidt, and I. Verbauwhede, "Hardware designer's guide to fault attacks," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 21, no. 12, pp. 2295–2306, Dec 2013.

[33] C. Bozzato, R. Focardi, and F. Palmarini, "Shaping the glitch: Optimizing voltage fault injection attacks," *TCHES*, vol. 2019, pp. 199–224, Feb. 2019. [Online]. Available: https://tches.iacr.org/index.php/TCHES/article/view/7390

[34] O. M. Guillen, M. Gruber, and F. De Santis, "Low-cost setup for localized semi-invasive optical fault injection attacks," in *Constructive Side-Channel Analysis and Secure Design*, S. Guilley, Ed. Cham: Springer International Publishing, 2017, pp. 207–222.

[35] D. Samyde, S. Skorobogatov, R. Anderson, and J.-J. Quisquater, "On a new way to read data from memory," in *First International IEEE Security in Storage Workshop, 2002. Proceedings.*, 2002, pp. 65–69.

[36] R. Piscitelli, S. Bhasin, and F. Regazzoni, "Fault attacks, injection techniques and tools for simulation," in *2015 10th International Conference on Design Technology of Integrated Systems in Nanoscale Era (DTIS)*, 2015, pp. 1–6.

[37] Y. Li, M. Chen, and J. Wang, "Introduction to side-channel attacks and fault attacks," in *2016 Asia-Pacific International Symposium on Electromagnetic Compatibility (APEMC)*, vol. 1. IEEE, 2016, pp. 573–575.

[38] H. Liu, Z. Liu, Y. Qiao, and Z. Lu, "Clock glitch fault injection attacks on an fpga aes implementation," *Journal of Electrotechnology, Electrical Engineering and Management*, vol. 1, pp. 23–27, 2017.

[39] T. Korak, M. Hutter, B. Ege, and L. Batina, "Clock glitch attacks in the presence of heating," in *2014 Workshop on Fault Diagnosis and Tolerance in Cryptography*, 2014, pp. 104–114.

[40] T. Korak and M. Hoefler, "On the effects of clock and power supply tampering on two microcontroller platforms," in *Workshop on Fault Diagnosis and Tolerance in Cryptography*, 2014, pp. 8–17.

[41] B. Yuce, N. F. Ghalaty, and P. Schaumont, "Improving fault attacks on embedded software using risc pipeline characterization," in *Proceedings of the 2015 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*, ser. FDTC '15. USA: IEEE Computer Society, 2015, p. 97âĂŞ108. [Online]. Available: https://doi.org/10.1109/FDTC.2015.16

[42] A. Barenghi, L. Breveglieri, I. Koren, and D. Naccache, "Fault injection attacks on cryptographic devices: Theory, practice, and countermeasures," *Proceedings of the IEEE*, vol. 100, no. 11, pp. 3056–3076, Nov 2012.

[43] M. Joye and M. Tunstall, Eds., *Fault Analysis in Cryptography*, ser. Information Security and Cryptography. Springer Berlin Heidelberg. [Online]. Available: http://link.springer.com/10.1007/978-3-642-29656-7

[44] Z. Kazemi, A. Papadimitriou, I. Souvatzoglou, E. Aerabi, M. M. Ahmed, D. Hely, and V. Beroulle, "On a low cost fault injection framework for security assessment of cyber-physical systems: Clock glitch attacks," in *2019 IEEE 4th International Verification and Security Workshop (IVSW)*, July 2019, pp. 7–12.

[45] Z. Kazemi, M. Fazeli, D. Hely, and V. Beroulle, "Hardware security vulnerability assessment to identify the potential risks in a critical embedded application," in *2020 IEEE 26th Inter-*

national Symposium on On-Line Testing and Robust System Design (IOLTS), July 2020, pp. 1–6.

[46] A. Barenghi, L. Breveglieri, I. Koren, G. Pelosi, and F. Regazzoni, "Countermeasures against fault attacks on software implemented aes: Effectiveness and cost," in *Proceedings of the 5th Workshop on Embedded Systems Security*, ser. WESS '10. New York, NY, USA: Association for Computing Machinery, 2010. [Online]. Available: https://doi.org/10.1145/1873548.1873555

[47] N. Moro, K. Heydemann, E. Encrenaz, and B. Robisson, "Formal verification of a software countermeasure against instruction skip attacks," *CoRR*, vol. abs/1402.6461, 2014. [Online]. Available: http://arxiv.org/abs/1402.6461

[48] J.-F. Lalande, K. Heydemann, and P. Berthomé, "Software countermeasures for control flow integrity of smart card C codes," in *ESORICS - 19th European Symposium on Research in Computer Security*, ser. Lecture Notes in Computer Science, M. Kutylowski and J. Vaidya, Eds., vol. 8713. Wroclaw, Poland: Springer International Publishing, Sep. 2014, pp. 200–218. [Online]. Available: https://hal.inria.fr/hal-01059201

[49] Z. Kazemi, C. Bresch, D. Hely, and V. Beroulle, "A systematic approach for hardware security assessment of secured IoT applications," in *TRUDEVICE 2020: Workshop on Trustworthy Manufacturing and Utilization of Secure Devices, DATE'20*.

[50] Z. Kazemi, A. Norollah, A. Kchaou, M. Fazeli, D. Hely, and V. Beroulle, "An in-depth vulnerability analysis of risc-v micro-architecture against fault injection attack," in *2021 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, Oct 2021, pp. 1–6.

[51] C. O'Flynn and Z. D. Chen, "Chipwhisperer: An open-source platform for hardware embedded security research," in *Constructive Side-Channel Analysis and Secure Design*, E. Prouff, Ed. Cham: Springer International Publishing, 2014, pp. 243–260.

[52] 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*, Sep. 2011, pp. 105–114.

[53] T. Korak, M. Hutter, B. Ege, and L. Batina, "Clock glitch attacks in the presence of heating," in *2014 Workshop on Fault Diagnosis and Tolerance in Cryptography*, Sep. 2014, pp. 104–114.

[54] J. Obermaier, R. Specht, and G. Sigl, "Fuzzy-glitch: A practical ring oscillator based clock glitch attack," in *2017 International Conference on Applied Electronics (AE)*, Sep. 2017, pp. 1–6.

[55] M. Agoyan, J.-M. Dutertre, D. Naccache, B. Robisson, and A. Tria, "When clocks fail: On critical paths and clock faults," in *Smart Card Research and Advanced Application*, D. Gollmann,

J.-L. Lanet, and J. Iguchi-Cartigny, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 182–193.

[56] T. Korak and M. Hoefler, "On the effects of clock and power supply tampering on two microcontroller platforms," in *2014 Workshop on Fault Diagnosis and Tolerance in Cryptography, FDTC 2014, Busan, South Korea, September 23, 2014*, A. Tria and D. Choi, Eds. IEEE Computer Society, 2014, pp. 8–17. [Online]. Available: https://doi.org/10.1109/FDTC.2014.11

[57] S. Endo, T. Sugawara, N. Homma, T. Aoki, and A. Satoh, "An on-chip glitchy-clock generator for testing fault injection attacks," *Journal of Cryptographic Engineering*, vol. 1, no. 4, p. 265, Oct 2011. [Online]. Available: https://doi.org/10.1007/s13389-011-0022-y

[58] T. Fukunaga and J. Takahashi, "Practical fault attack on a cryptographic lsi with iso/iec 18033-3 block ciphers," in *2009 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*, Sep. 2009, pp. 84–92.

[59] M. Matsubayashi, A. Satoh, and J. Ishii, "Clock glitch generator on sakura-g for fault injection attack against a cryptographic circuit," in *2016 IEEE 5th Global Conference on Consumer Electronics*, Oct 2016, pp. 1–4.

[60] H. Martín, T. Korak, E. S. Millán, and M. Hutter, "Fault attacks on STRNGs: Impact of glitches, temperature, and underpowering on randomness," *IEEE Transactions on Information Forensics and Security*, vol. 10, no. 2, pp. 266–277, Feb 2015.

[61] N. Beringuier-Boher, K. Gomina, D. Hély, J.-B. Rigaud, V. Beroulle, A. Tria, J. Damiens, P. Gendrier, and P. Candelier, "Voltage glitch attacks on mixed-signal systems," in *2014 17th Euromicro Conference on Digital System Design*, Aug 2014, pp. 379–386.

[62] N. Timmers and C. Mune, "Escalating privileges in Linux using voltage fault injection," in *2017 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*, Sep. 2017, pp. 1–8.

[63] A. Barenghi, G. Bertoni, E. Parrinello, and G. Pelosi, "Low voltage fault attacks on the RSA cryptosystem," in *Proceedings of the 2009 Workshop on Fault Diagnosis and Tolerance in Cryptography*, ser. FDTC '09. USA: IEEE Computer Society, 2009, p. 23âĂŞ31. [Online]. Available: https://doi.org/10.1109/FDTC.2009.30

[64] F. E. Potestad-Ordóñez, C. J. Jimé nez Fernández, and M. Valencia-Barrero, "Vulnerability analysis of Trivium FPGA implementations," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 12, pp. 3380–3389, Dec 2017.

[65] B. Yuce, N. F. Ghalaty, and P. Schaumont, "Improving fault attacks on embedded software using RISC pipeline characterization," in *2015 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*, Sep. 2015, pp. 97–108.

[66] B. Yuce, P. Schaumont, and M. Witteman, "Fault attacks on secure embedded software: Threats, design, and evaluation," *Journal of Hardware and Systems Security*, vol. 2, no. 2, pp. 111–130, Jun 2018. [Online]. Available: https://doi.org/10.1007/s41635-018-0038-1

[67] A. Barenghi, G. M. Bertoni, L. Breveglieri, and G. Pelosi, "A fault induction technique based on voltage underfeeding with application to attacks against AES and RSA," vol. 86, no. 7, pp. 1864–1878. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0164121213000320

[68] L. Zussa, J.-M. Dutertre, J. Clédiere, 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)*, July 2013, pp. 110–115.

[69] L. Zussa, J.-M. Dutertre, J. Clédière, B. Robisson, and A. Tria, "Investigation of timing constraints violation as a fault injection means," in *27th Conference on Design of Circuits and Integrated Systems (DCIS)*, Avignon, France, Nov. 2012, p. pas encore paru. [Online]. Available: https://hal-emse.ccsd.cnrs.fr/emse-00742652

[70] L. Zussa, J.-M. Dutertre, J. Clediere, and B. Robisson, "Analysis of the fault injection mechanism related to negative and positive power supply glitches using an on-chip voltmeter," in *2014 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, May 2014, pp. 130–135.

[71] N. Timmers, A. Spruyt, and M. Witteman, "Controlling PC on ARM using fault injection," in *2016 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*, Aug 2016, pp. 25–35.

[72] H. Choukri and M. Tunstall, "Round reduction using faults," pp. 13–24.

[73] R. Anderson and M. Kuhn, "Low cost attacks on tamper resistant devices," in *Security Protocols*, B. Christianson, B. Crispo, M. Lomas, and M. Roe, Eds. Springer Berlin Heidelberg, pp. 125–136.

[74] J. Korczyc and A. Krasniewski, "Evaluation of susceptibility of fpga-based circuits to fault injection attacks based on clock glitching," in *2012 IEEE 15th International Symposium on Design and Diagnostics of Electronic Circuits Systems (DDECS)*, April 2012, pp. 171–174.

[75] B. Selmke, F. Hauschild, and J. Obermaier, "Peak clock: Fault injection into pll-based systems via clock manipulation," in *Proceedings of the 3rd ACM Workshop on Attacks and Solutions in*

Hardware Security Workshop, ser. ASHES'19. New York, NY, USA: Association for Computing Machinery, 2019, p. 85âĂŞ94. [Online]. Available: https://doi.org/10.1145/3338508.3359577

[76] T. Katashita, Y. Hori, H. Sakane, and A. Satoh, "Side-channel attack standard evaluation board sasebo-w for smartcard testing," 2011.

[77] N. Selmane, S. Guilley, and J.-L. Danger, "Practical setup time violation attacks on aes," in *2008 Seventh European Dependable Computing Conference*, May 2008, pp. 91–96.

[78] Y. Romailler and S. Pelissier, "Practical fault attack against the ed25519 and eddsa signature schemes," in *2017 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*, Sep. 2017, pp. 17–24.

[79] M. Hutter, J.-M. Schmidt, and T. Plos, "Contact-based fault injections and power analysis on rfid tags," in *2009 European Conference on Circuit Theory and Design*, Aug 2009, pp. 409–412.

[80] *Spartan-6 FPGA Data Sheet*, Xilinx Inc, 1 2015, vol. 162.

[81] A. Note and J. Tatsukawa, *MMCM and PLL Dynamic Reconfiguration MMCM and PLL Configuration Bit Groups*, Xilinx Inc.

[82] Wikipedia, "Advanced Encryption Standard — Wikipedia, the free encyclopedia," http://fr.wikipedia.org/w/index.php?title=Advanced%20Encryption%20Standard&oldid=181841434, 2021, [Online; accessed 23-November-2021].

[83] A. A. Abdelrahman, M. M. Fouad, and H. Dahshan, "Analysis on the aes implementation with various granularities on different gpu architectures," *Advances in Electrical and Electronic Engineering*, vol. 15, no. 3, pp. 526–535, 2017.

[84] J. C. Talwana and H. Hua, "Smart world of internet of things (iot) and its security concerns," *2016 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, pp. 240–245, 2016.

[85] A. Barenghi, L. Breveglieri, I. Koren, and D. Naccache, "Fault injection attacks on cryptographic devices: Theory, practice, and countermeasures," *Proceedings of the IEEE*, vol. 100, no. 11, pp. 3056–3076, Nov 2012.

[86] A. B. Pawar and S. Ghumbre, "A survey on iot applications, security challenges and counter measures," in *2016 International Conference on Computing, Analytics and Security Trends (CAST)*, Dec 2016, pp. 294–299.

[87] S. Picek, L. Batina, P. Buzing, and D. Jakobovic, "Fault injection with a new flavor: Memetic algorithms make a difference," in *Constructive Side-Channel Analysis and Secure Design*, S. Mangard and A. Y. Poschmann, Eds. Springer International Publishing, pp. 159–173.

[88] C. Bresch, S. Chollet, and D. Hély, "Towards an inherently secure run-time environment for medical devices," in *2018 IEEE International Congress on Internet of Things (ICIOT)*, July 2018, pp. 140–147.

[89] Running a RISC-v processor on the arty a7 - digilent reference. [Online]. Available: https://digilent.com/reference/programmable-logic/arty-a7/arty_a7_100_risc_v/start

[90] Open on-chip debugger. [Online]. Available: https://openocd.org/

[91] "Digilent/digilent-xdc," original-date: 2017-04-18T23:52:32Z. [Online]. Available: https://github.com/Digilent/digilent-xdc/blob/10e32cb88446b8d60cf7a34ed51cecd4e0aa0d9b/Arty-A7-35-Master.xdc

[92] D. A. Patterson and A. Waterman, "The risc-v reader: An open architecture atlas," 2017.

[93] A. Waterman and K. Asanovi'c, "The RISC-v instruction set manual volume i: User-level ISA."

[94] T. Given-Wilson, N. Jafri, J.-L. Lanet, and A. Legay, "An automated formal process for detecting fault injection vulnerabilities in binaries and case study on present," in *2017 IEEE Trustcom/BigDataSE/ICESS*, Aug 2017, pp. 293–300.

[95] M. B. Petersen, "RIPES: A visual computer architecture simulator and assembly code editor built for the RISC-V instruction set architecture." [Online]. Available: https://github.com/mortbopet/Ripes

[96] "The LLVM Compiler Infrastructure Project." [Online]. Available: https://llvm.org

[97] R. Corin and F. A. Manzano, "Taint analysis of security code in the klee symbolic execution engine," in *ICICS*, 2012.

[98] "KLEE Symbolic Execution Engine." [Online]. Available: https://klee.github.io/

[99] H. M. Le, V. Herdt, D. Große, and R. Drechsler, "Resilience evaluation via symbolic fault injection on intermediate code," in *2018 Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2018, pp. 845–850.

[100] E. Boespflug, C. Ene, L. Mounier, and M.-L. Potet, "Countermeasures Optimization in Multiple Fault-Injection Context," in *"Fault Diagnosis and Tolerance in Cryptography" FDTC 2020*, Milan (Virtual Workshop), Italy, Sep. 2020. [Online]. Available: https://hal.archives-ouvertes.fr/hal-02951150