Thwarting Electromagnetic Fault Injection Attack Utilizing Timing Attack Countermeasure

Marjan Ghodrati

Thesis submitted to the Faculty of the

Virginia Polytechnic Institute and State University

in partial fulfillment of the requirements for the degree of

Master of Science

in

Computer Engineering

December 15, 2017

Blacksburg, Virginia

Keywords: Electromagnetic Fault Injection, Countermeasure, Attack

Thwarting Electromagnetic Fault Injection Attack Utilizing Timing Attack Countermeasure

Marjan Ghodrati

Abstract

The extent of embedded systems role in modern life has continuously increased over the years. Moreover, embedded systems are assuming highly critical functions with security requirements more than ever before. Electromagnetic fault injection (EMFI) is an efficient class of physical attacks that can compromise the immunity of secure cryptographic algorithms. Despite successful EMFI attacks, the effects of electromagnetic injection on a processor are not well understood. This includes lack of solid knowledge about how EMFI affects the circuit and deviates it from proper functionality. Also, effects of EM glitches on the global networks of a chip such as power, clock and reset network are not known. We believe to properly model EMFI and develop effective countermeasures, a deeper understanding of the EM effect on a chip is needed. In this thesis, we present a bottom-up analysis of EMFI effects on a RISC microprocessor. We study these effects at three levels: at the wire-level, at the chip-network level, and at the gate-level considering parameters such as EM-injection location and timing. We conclude that EMFI induces local timing errors implying current timing attack detection and prevention techniques can be adapted to overcome EMFI. To further validate our hypothesis, we integrate a configurable timing sensor into our microprocessor to evaluate its effectiveness against EMFI.

Thwarting Electromagnetic Fault Injection Attack Utilizing Timing Attack Countermeasure

Marjan Ghodrati

General Audience Abstract

In the current technology era, embedded systems play a critical role in every human's life. They are collecting very precise and private information of the users. So, they can become a potential target for the attackers to steal this valuable information. As a result, the security of these devices becomes a serious issue in this era.

Electromagnetic fault injection (EMFI) is an efficient class of physical attacks that can inject faults to the state of the processor and deviate it from its proper functionality. Despite its growing popularity among the attackers, limitations and capabilities of this attack are not very well understood. Several detection techniques have been proposed so far, but most of them are either very expensive to implement or not very effective. We believe to properly model EMFI and develop effective countermeasures, a deeper understanding of the EM effect on a chip is needed. In this research work, we try to perform a bottom-up analysis of EM fault injection on a RISC microprocessor and do a comprehensive study at all wire-level, chip-network level, and gate-level and finally propose a solution for it.

To My Happy Family

My devoted parents: Abbas Ghodrati and Narjes Ansari

My wonderful husaband: Ali Marzoughi

My best friend and lovely son: Keon Marzoughi

Acknowledgments

I am glad to have an opportunity to demonstrate my gratitude to all those who supported me since I joined Virginia Tech.

I would like to express my sincere gratitude to my advisor, Dr. Leyla Nazhandali, for her support and encouragement during my studies. I have learned so much from her rich knowledge, method of approaching a problem and devoted attitude toward research.

It is an honor for me to have Prof. Patrick Schaumont as a committee member. I am very thankful to his rigorous knowledge, great ideas and gentle personality. I am sincerely grateful to my other committee member, Dr. Lynn Abbot for inspiring me to look at the problems that I was facing from different angles. I would never be able to discover as many aspects of my work that I have discussed, without discussions, opinions and suggestions from the committee members.

I would like to acknowledge my lab-mates in Secure Embedded System lab, especially those with whom I worked in FAME project: Bilgiday Yuce, Chinmay Deshpande.

I also appreciate all other colleagues and friends outside the projects on which I have worked:

Conor Patrick, Yuan Yao, Archanaa Krishnan and Surabhi Gujar. It was a great fortune to have the opportunity to learn from each other in the past two-and-a-half years, and widen the knowledge of different areas in the vast field of hardware security.

I also would like to give special praise to my great husband for all his help and support. Without his support and encouragement, this accomplishment would be impossible.

I am sure I have missed so many names in my appreciation.

To you all, thank you!

Contents

List of Figures

Chapter 1

Introduction

The extent of embedded systems' role in modern life has continuously increased over the years. Moreover, embedded systems are assuming highly critical functions with security requirements more than ever before. Physical attacks can compromise the effectiveness of the most secure cryptographic algorithms by manipulating or just observing the hardware that implements them. There are two types of physical attacks, side-channel attacks, and fault attacks. Unlike side-channel attacks which are passive, fault attacks are active. In these attacks, the attacker manipulates the working conditions of the circuit to the point that it deviates from its correct functionality [11] [21]. This can result in either bypassing the security checks or creating new side-channels that help uncover the cryptographic keys. The focus of this research is on the second category (fault attacks) which has gained a lot of attention recent years. In section 1.1, different types of fault attacks are discussed. Subsection 1.1.1 gives an overview to Electromagnetic current induction and talks about how it can turn into an attack. Section 1.2 presents the motivation behind this research. We provide the contribution of this work and organization of the thesis in section 1.3 and 1.4 respectively.

1.1 Fault Attack

Fault injection is the process of deliberately changing the operational condition of a device to deviate it from its proper functionality. So, fault injection can lead a vulnerable computing device to a state that leaks the secure information and bypass security [20]. There are different types of fault attacks. Below, some of the most common ones are discussed.

- **Clock glitching:** Clock glitching is a fault injection technique that relies on modification of the processors clock signal [8]. By using this method, the attacker increases the clock frequency of a digital circuit for a short period. So, if the clock frequency of the circuit fall below the critical path of it, different kind of faults can be injected depends on when the clock line of the processor is manipulated [9]. An instruction can be changed or skipped, a corrupted data might be loaded from memory, or data can be stored wrongly in a memory address [1] [10]. Clock glitching is one of the simplest methods of fault injection. That is why it is broadly using to break the security of microprocessors. However, there are very promising and effective countermeasures available for this type of fault attacks.

- **Voltage glitching:** Voltage glitching or power glitching is a common method of fault injection that the attacker increases or decreases the voltage of a target device above or below a threshold for a fixed period of time [11]. Also, inserting transients into the power supply lines known as spikes may lead to faulty executions with controllable timing accuracy [28] [31].

- **Laser-based fault injection:** In this method of fault injection, the attacker uses laser beams to induce faults at precise locations [27] [8]. To implement this attack, the chip should be decapsulated to allow the light from the laser to reach the surface of the chip. This light can switch on or off each transistor of the processor, and as a result, change the behavior of it.

- **Micro-probing:** By placing very tiny needles called probes, the attacker can measure or overwrite a signal permanently or at a specific time [25]. So in this case, the chip should be depackaged until the attacker gets access to the surface to observe or manipulate the integrated circuit.

- **Temperature:** Heating or cooling a crypto-device may induce an error into a crypto-device and alter its behavior [28] [26]. It is proven by [17] that heating a circuit changes its behavior. In this paper [17], the author observed some flip bits in memory while it was heated to 100C.

- **Electromagnetic fault injection:** Electromagnetic fault injection which is the topic of this research, is a method of fault injection which is based on electromagnetic induction. In this method, the attacker places a metal coil close to the surface of the chip, and creates a magnetic field around it. As a result, currents can be induced into the processor. Since each processor consists of millions of transistors, the induced currents can change the state of these transistors and as a result change the behavior of the processor [14] [19] [18]. This method does not need decapsulation, which is a complicated process with the risk of damaging the target device.

In principle, there are three kinds of fault attacks, **non-invasive**, **semi-invasive**, and **invasive** [25]. The faults that do not need any modification to the chip are categorized in the first group, non-invasive, such as clock glitching, voltage glitching and heating. Fault attacks that require the chip to be decapsulated, but do not modify the circuit are categorized in the second group, semi-invasive such as laser fault injection. Finally, the faults that need modification to the chip such as cutting a wire on the circuit are classified as invasive attacks [25]. Micro-probing is an example of the third category. Electromagnetic fault injection, which is the focus of this thesis, is a non-invasive type of attacks. Since, it does not require decapsulation or any modification of the device.

Additionally, faults can have different timely behavior [18]:

- **Permanent faults** are the faults that damage the chip or a component permanently [18].
- **Temporary faults** are recoverable. They only affect the circuit behavior for a limited time like a single transistor connectivity [18].
- **Transient faults** are similar to temporary faults but the fault may put the system

in an erroneous state. As a result, the system may need a reboot to go back to the normal operation [18].

Based on all the experiments that have been done in this research, the only effects of EMFI could be either temporary or transient.

1.1.1 Electromagnetic Fault Injection

Electromagnetic fault injection is referred to perturbations that are based on electromagnetic induction and injected to the target system/processor to interfere in its operation. The aim of injecting this type of fault is to stop the device from operating properly. To be more specific, by introducing a parasitic to the processor while it is in the middle of calculations, an error is introduced to the output. In order to inject an electromagnetic fault, first, a voltage signal is generated and fed to a probe. The voltage V creates current I in the probe, leading to a magnetic field B according to Maxwell-Faraday equations. The intensity of the magnetic field B in a ring-shaped conductor with radius R and current I in vacuum can be calculated according to equations (1.1).

$$B := (\mu_0 \mu_r I)/2R \tag{1.1}$$

[10] has formulated the magnetic filed B of a one loop coil as (1.2):

$$B := (\mu_r I b^2)/2(b^2 + z^2)^(3/2) \tag{1.2}$$

In the above equation, μ_r is a constant representing permeability of the material that the magnetic field is passing through. For vacuum, this constant is equal to unity. However, for other materials, it has different (higher) magnitudes. b and z parameters are the radius of the coil and the distance from the plane of the coil respectively. Then the magnetic flux can be calculated by equation (1.3).

$$\phi_B := BAcos(\theta) \tag{1.3}$$

So, if a current pulse is sent through the coil, the current I of the coil suddenly changes [28]. Changing the current I, will change the electromagnetic field B which in-turn will affect the magnetic flux ϕ_B of the coil. Once the intensity of the magnetic field was calculated from (1.2), The Maxwell-Faraday equation (1.4) can be used to link the voltage E induced across a conductor that is subject to the field.

$$\int E.dl = -d/dt \int \int B.dS \tag{1.4}$$

According to the equation above, variations in the magnetic field intensity can introduce a voltage in a conductor. Also, moving a conductor in a constant magnetic field can introduce a voltage across it. The method used for injection of EM fault into processors and systems is to apply an alternating voltage V (resulting in an alternating current I) to a conductor. Accordingly, the resulting alternating magnetic field will induce a voltage to the system under attack, interfering with its operation. So according to equations (1.2) and (1.3) the injected magnetic flux is affected by:

- The angle between the target device and the coil (setting the angle to 90, makes the $cos(\theta)$ one and generates a stronger magnetic flux)

- The distance between the coil and the target device (smaller distance will lead to a more significant magnetic flux)
- The current amplitude that runs through the coil (more current will result in a bigger magnetic flux)
- The area of the coil (bigger probe will create a bigger magnetic flux)
- The magnetic permeability of the coils core (a material with higher magnetic permeability will create a stronger magnetic flux)

So we should consider the above-mentioned parameters while we are injecting electromagnetic faults to implement a successful attack.

1.2 Motivation

As discussed in the previous section, there are several techniques for fault injection such as clock glitching, voltage glitching, optical fault injection, temperate, microprobing and electromagnetic fault injection that can compromise the immunity of secure cryptographic algorithms. These methods can affect the processor in different ways, like skipping an instruction, bit set, bit reset and so on.

The most common forms of fault attacks are clock and voltage glitching. However, designers have proposed several countermeasures against these specific types of fault attacks which have rendered them unusable. For another popular form of fault attack, optical injection, the attacker needs to decapsulate the chip, which is a complicated process with the risk of damaging the target device. Moreover, it is possible to employ countermeasures such as light sensors to detect such attacks. Therefore, there has been an increasing interest in EMFI in the past several years as there is no need to decapsulate the chip. Furthermore, with EMFI, the attackers have a broad range of options such as radiation intensity as well as offset and duration of the radiation at their disposal. These options allow the adversaries to fine tune their attack to quickly achieve their desired faulty behavior.

Clearly, EMFI is a method of fault injection with several advantages over the other methods which is able to inject faults to the state of the processor and deviate it from its proper operation. It has been shown by many publications that EMFI is able to break the cryptographic algorithms such as AES and RSA. In [13], the authors used EMFI to break AES code on an 8-bit microcontroller and extracted the secure key.

Despite its growing popularity, limitations and capabilities of this attack are not very well understood. Several detection techniques have been proposed, but most of them are either very expensive to implement or not very effective. We believe to properly model EMFI and develop effective countermeasures, a deeper understanding of the EM effect on a chip is needed. In this research work, we try to perform a bottom-up analysis of EM fault injection on a RISC microprocessor and do a comprehensive study at all wire-level, chip-network level, and gate-level and finally propose a solution for it.

1.3 Contributions

In this research, a total of three goals is being pursued. First, it is desired to explore the effects of EMFI on a microprocessor at different levels to have a better knowledge of what is occurring while the chip is under EM fault attack. Second, this research is trying to optimize the parameters that affect the EMFI to have the highest success rate of fault injection. Finally, based on results and the conclusion, an effort is made to find an effective and low-cost countermeasure for EM fault attack. The main research questions of this thesis are as follows:

1. What are the effects of EMFI at wire-level on a 32-bit microprocessor?
2. What are the effects of EMFI at chip-level on a 32-bit microprocessor?
3. What are the effects of EMFI at gate-level on a 32-bit microprocessor?
4. How to optimize the parameters that are affecting EMFI to have the highest success rate of fault injection?
5. Is the common timing fault detector (glitch detector) effective as a countermeasure against EMFI?

Currently, there is little information available about the effects of EMFI at different levels on a microprocessor. Without knowing the effects of EMFI on a microprocessor, it would not be possible to propose an efficient and effective countermeasure against this type of fault attack. So after answering the first three questions, we can provide an answer to the final question.

1.4 Organization

This chapter provides an introduction to the fault attacks and more specifically Electromagnetic fault injection. It also presents the motivation behind this work as well as research goals. The rest of this thesis is organized as follows:

Chapter 2: provides a brief look at other related works.

Chapter 3: introduces the target device which is used in this research as well as the EMFI setup.

Chapter 4: provides a bottom-up analysis of EMFI on a RISC microprocessor

Chapter 5: proposes a possible solution to protect embedded systems against EMFI

Chapter 6: Finally, in this chapter, we summarize the research work and warp it up with a conclusion and future works.

Chapter 2

Related Work

Electromagnetic fault injection was introduced by [24] in 2002. According to [24], an external electromagnetic field can induce eddy currents on the surface of the chip, and as a result, inject faults into the transistors and memory cells of the processor. They showed this induced fault caused some flip bits in RAM and EPROM memory cells. Later in 2007, reference [19] used high-frequency spark gaps instead of magnetic field to disturb a cryptographic computation. The induced spark-gaps change the flowing current suddenly and lead to a powerful electromagnetic burst and radiation. They used this method to break the CRT-based RSA algorithm which is using very commonly in cryptographic systems. Finally, they could extract the secure key successfully. Dehbaoui et al. in [14] used EM pulses to implement a successful attack on a software implementation of AES on an 8-bit microcontroller and a hardware AES embedded in an FPGA. The authors were able to inject faults into every byte of AES. They were also able to inject single or multiple faults to the hardware AES on the

FPGA. Based on their observation, they draw a hypothesis that EM pulses induce timing violations. Then they added a timing sensor on the FPGA that had AES on it. During their experiments, they noted the sensor was only able to detect 10% of induced faults. A year later, Moro et al. in [22] explored the impact of an electromagnetic glitch fault injection on a state-of-the-art microcontroller and built an associated register-transfer level fault model. According to this paper, EM pulses can affect both data flow and program flow. So, an attacker can change one instruction to another one, or change the value of a piece of data loaded from memory. They also showed some instructions or registers are more vulnerable than the others, but due to lack of access to the architecture of the microprocessor, their fault model is not comprehensive. The authors in [23] created an experimental setup for performing fault injection and were able to provide the preliminary nature of the induced fault model. They experimentally demonstrated that the faults were very local and produced a combination of bit-set/reset faults.

Reference [30] explored the efficiency of a glitch detector against EMFI. The detector was a delay based timing sensor which was first designed to detect clock or power glitches. This method identifies setup time violations by using fixed guarding delay which is set slightly higher than critical path of the design. Hence, voltage disturbances will be detected, and an alarm will be issued. They proved one glitch detector is not enough to detect a good percentage of electromagnetic fault injections.Since, EM is more local than global. They also investigated the efficiency of five glitch detector against EMFI. It is reported that only 32% of faults could be detected by all five sensors. So as mentioned in this paper, the number of detectors to be used to get a high detection ratio, as well as their optimal placement on the IC surface, remained open problems. Wei et al. in [21] proposed a PLL based sensor composed of a watchdog ring oscillator (RO) and a phase-locked-loop (PLL) to detect the EMFI-induced phase shift in a watchdog ring oscillator. This detector compares two clock input timings, one is an external reference clock, and the other one is a feedback internal core clock. In case of any phase or frequency difference between two clock sources, PLL may enter an unlocked state and raise the alarm. It is mentioned in this paper, that by distributing the feedback clock path all over the security sensitive cryptographic core, the EMFI attack can be easily captured as an unlock event in PLL loop. The experiments show a very good detection ratio with significant power security margin. However, the proposed countermeasure is based on PLL module which is not available in low power-constrained devices due to very high power consumption. Authors in [12] proposed a novel EMFI detector which is based on Hogge Phase-Detector. They implemented a Hogge Phase Detector to detect the ring oscillator disturbance which is induced by EMFI. They reported 93.15% detection ratio with a failure of 0.0069. However, the shortcoming of this paper is considering and detecting the injection attempts which even does not inject any faults to the state of the processor. In their experiments, they vary the glitch intensity in a range of 40% to 100% randomly. So some of these injections (usually below 75%) are not powerful enough to corrupt the operation of the processor, and by raising the alarm for all these disturbances, the performance will be sacrificed. Chinmay et al. in [16] proposed an EMFI detector based on two flip-flops (FFs), main and shadow ones. These two FFs always have complementary values. If due to EMFI injection, either main FF or shadow FF gets affected, the injected fault would be detected by XNORing the output of these two registers. The drawback of this work is the very high overhead. Since for a full detection, all the flip-flops should be doubled.

Chapter 3

EMFI Experimental Setup and the Target Device

This Chapter gives an overview of the setup which is used to implement and induce EM faults, and the target device that is used as the device under test for doing experiments in this research.

3.1 The EMFI Setup

The EMFI setup is shown in figure 3.1. It consists of a computer which has the inspector on it, a glitch controller (VC glitcher), the EMFI probe, an oscilloscope, XYZ motorized stage, and finally the target processor. Below, each of these elements will be described briefly.

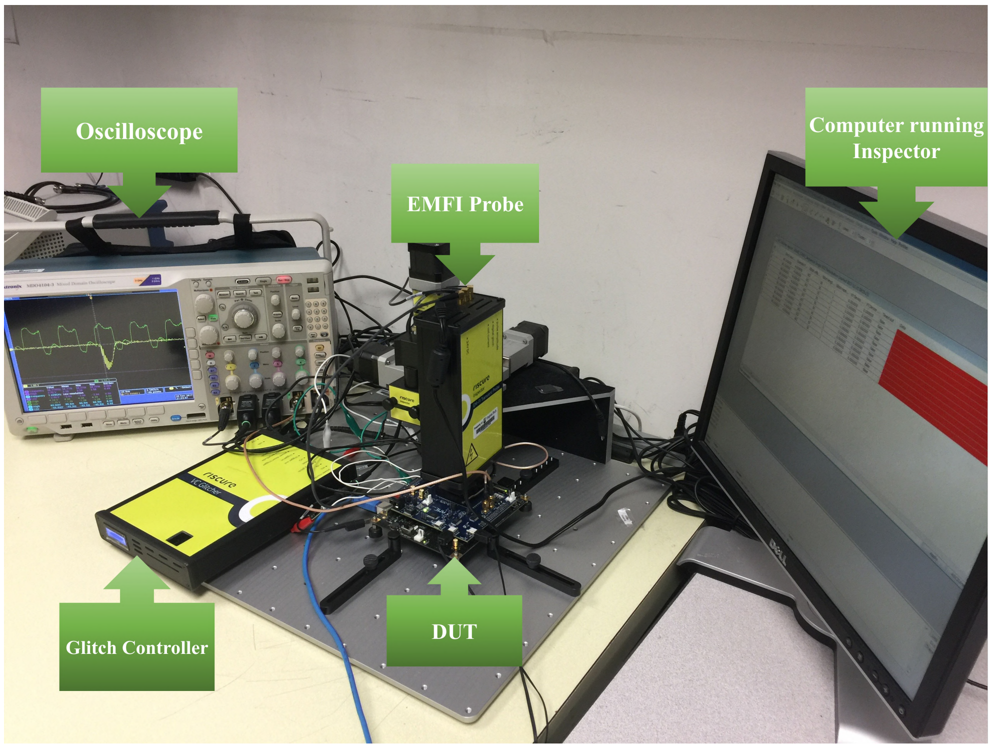


Figure 3.1: Picture of our experimental setup.

3.1.1 Motorized XYZ table

The motorized XYZ stage which is shown in figure 3.1 is an instrument that the EMFI probe can be placed on it, and it will provide the fully automated scan over the chip surface. The XYZ stage is movable in all directions automatically and can be programmed by the inspector. Small step size and little repositioning error of the XYZ table allow profiting from the spatial resolution provided by the EM probe [5]. So the EMFI probe can move automatically on different locations on the surface of the chip as well as move vertically to get closer or further from the surface of the processor by using the XYZ motorized stage.

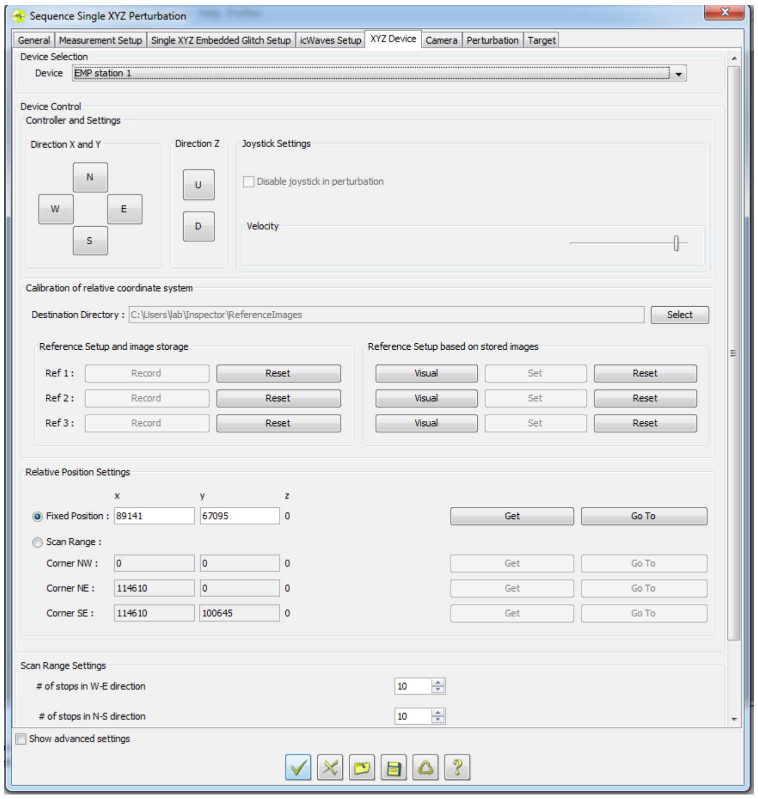
3.1.2 Inspector

Inspector which is developed by Riscure [4] is a software runs on the PC to control the XYZ table, configure the attack parameters and send them to the glitch controller, and communicate with the target processor. In the following, these features will be explained in more details.

- **Controlling the XYZ table:** There is a window which is shown in figure 3.2 available in inspector that can be used to set the location of fault injection. There are two options available, scanning the chip or injecting fault at a single point. To scan the chip, three points should be set as reference points, and the number of injection on both X or Y axis should be defined

- **Configuring the attack parameters:** Different attack parameters can be set by Inspector such as Glitch offset, Glitch source power, Glitch length and glitch repetition. Figure 3.3 shows the configuration panel of the inspector which is used to set EMFI parameters.

  - *Gitch offset*: refers to the timing of EMFI relative to the trigger signal and can take any values to target a specific instruction. In our setup, there is always some delays (about 110ns) between the time trigger raises and the moment the EMFI probe creates the glitch. To attack a specific instruction, we should always consider this delay, and regarding this value, set the glitch offset. We will later see that the glitch offset has a significant effect on the attack success.


Sequence Single XYZ Perturbation
General | Measurement Setup | Single XYZ Embedded Glitch Setup | icWaves Setup | XYZ Device | Camera | Perturbation | Target
Device Control
Controller and Settings
Direction X and Y
N
W
E
S
Direction Z
U
D
Joystick Settings
Velocity
Calibration of relative coordinate system
Select
Reference Setup and image storage
Reference Setup based on stored images
Record
Reset
Visual
Set
Relative Position Settings
Get
Go To
Scan Range Settings

Device Selection
Device: EMP station 1

☐ Disable joystick in perturbation

Destination Directory : C:\Users\lab\Inspector\ReferenceImages

	x	y	z
● Fixed Position :	89141	67095	0
○ Scan Range :			
Corner NW :	0	0	0
Corner NE :	114610	0	0
Corner SE :	114610	100645	0

# of stops in W-E direction: 10

# of stops in N-S direction: 10

☐ Show advanced settings

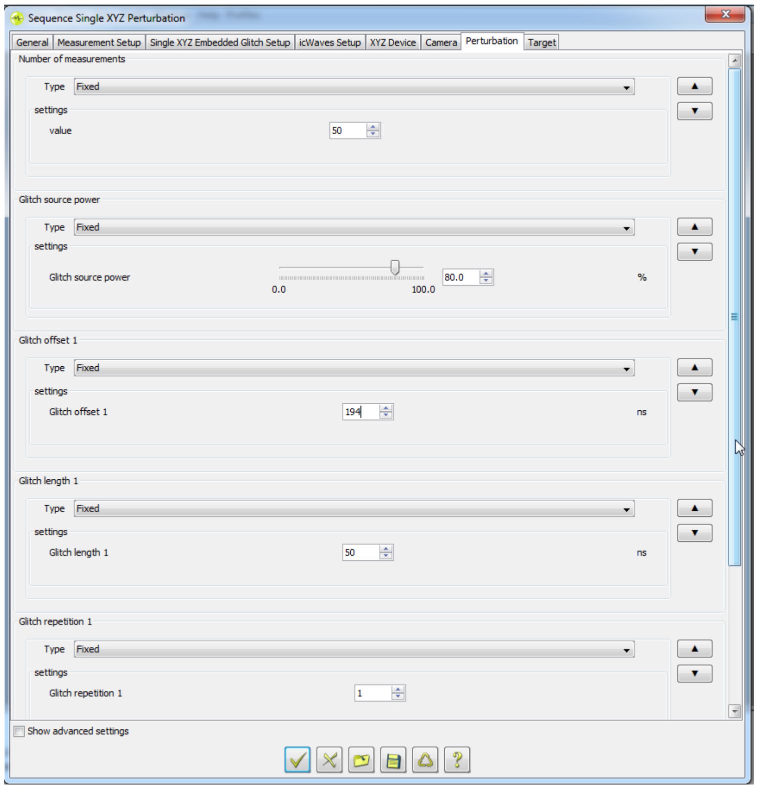
Figure 3.2: Controlling the XYZ table

– *Glitch source power*: refers to the intensity of the glitch, and it is a percentage of the highest injection voltage, which is 450V. For example, an EMFI with 45V pulse is considered at 10% intensity.

– *Glitch length*: is the amount of time that the EMFI probe should provide current through coil continuously. In these experiments, we fixed it to 50ns.

– *Glitch repetition*: is the number of glitches the EMFI probe will generate. In our study, we set it to one glitch.

- **Communicating with the target device:** The inspector communicates with the target devices through a UART port. We have a UART module implemented on the


Sequence Single XYZ Perturbation

General | Measurement Setup | Single XYZ Embedded Glitch Setup | icWaves Setup | XYZ Device | Camera | Perturbation | Target

Number of measurements
Type: Fixed
settings
value: 50

Glitch source power
Type: Fixed
settings
Glitch source power: 80.0 % (0.0 – 100.0)

Glitch offset 1
Type: Fixed
settings
Glitch offset 1: 194 ns

Glitch length 1
Type: Fixed
settings
Glitch length 1: 50 ns

Glitch repetition 1
Type: Fixed
settings
Glitch repetition 1: 1

Show advanced settings

Figure 3.3: Perturbation window

microprocessor. Also, there is a java module on inspector that is modified to implement this communication. It sends a command to the target device to start the operation and finally receives all the results from the processor. This module compares the received results with the correct result and determines if it is faulty or not.

3.1.3 VC Glitcher

VC glitcher or glitch controller is a tool developed by Riscure to perform clock/voltage fault injection as well as control optical pulses and EMFI [6]. It is connected to the inspector, EMFI probe, and the target device. It receives the attack parameters from the inspector, and once, it receives the trigger from the target processor, it will send the attack commands to the EMFI probe to implement an electromagnetic fault injection.

3.1.4 EMFI Probe

The EM-FI transient probe induces fast, high power, EM pulses on a user-defined location of the chip [3]. It is connected to the VC glitcher and receives the attack parameters such as the glitch intensity from the inspector through the VC glitcher. There are different probes available to generate different EM field over the chip surface. The tip of two probes is single loop metal coil with a diameter of 1.5mm, while the tip's diameter of the other two is 4mm. The current pulse of the 4 mm tip is weaker and longer due to the higher inductance of the larger tip coil. However, since the loop area of 4mm tip is six times bigger than 1.5 mm tip, the EM pulse will be stronger. For this reason, 4mm tip is suggested when targets thickness exceeds 1.5 mm. Otherwise, 1.5 mm tip is suggested [3].

The polarity of the probes is also different. We have two red probes with positive polarity and two black probes with negative polarity. The polarity of the tips determines the polarity of glitch voltage induces in the target device.

3.1.5 Oscilloscope

The oscilloscope Techtronix is used to measure the trigger signals, coil current and the perturbation.

3.1.6 Target Processor

Target processor is placed below the EMFI probe with a less than a 2mm distance between them. The trigger signal which should be connected to the trigger port of the VC glitcher, is generated by the target processor. It is set during execution of the target program at a specific instruction which is the target instruction of the attack. In next section, the target device which is used in this research is introduced.

3.2 Target Device

In this research, FAME microprocessor which stands for Fault-attack Awareness using Microprocessor Enhancements used as the target device. The features of FAME which is a custom-made microprocessor by our team are provided in next subsection.

3.2.1 FAME Microprocessor

The device under attack, which is shown in figure 3.4 is a microprocessor with an in-order RISC processor implementing the SPARC-V8 instruction set. The processor is manufactured in 180 *nm* TSMC process on a $5 \times 5\ mm^2$ packaged die. The processor core is based on the open-source 32-*bits* LEON3-design [2]. In addition to the RISC core, the prototype chip includes 2KB data cache and 1KB instruction cache. It also provides a 4-*pin* General Purpose Input Output (GPIO) port with interrupt capability and a Universal Asynchronous

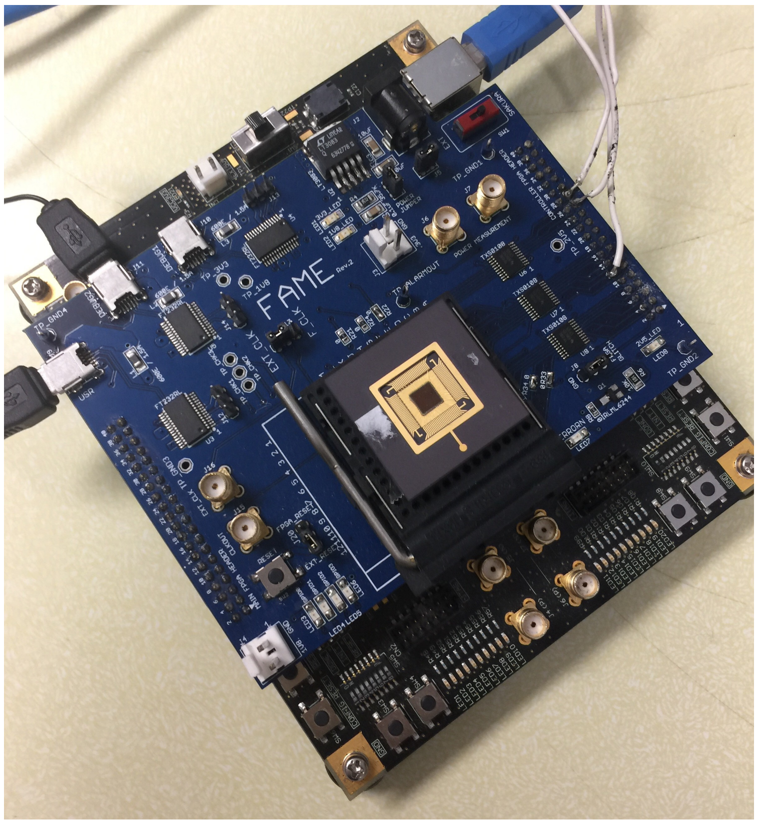


Figure 3.4: FAME microprocessor

Receiver/Transmitter (UART) interface. The chip mounts on a 108-pin Pin Grid Array (PGA) package, which is socketed on a custom testing board which is developed as an extension to the SAKURA-G Board [7] and attaches to it using the standard expansion headers. FAME microprocessor has seven pipeline stages and a maximum clock frequency of 80 MHz. A configurable delay-based timing sensor is integrated into the chip to detect faults at the hardware level and mitigate them by an appropriate response in software. There are three reasons why we use a custom-made processor as the target device as opposed to an off-the-shelf microprocessor:

Knowledge of layout: We have the detailed information about the layout including the location of the clock network, memories, various pipeline flip-flops (FFs), etc. This helps us explain why attacking the processor at various points on the chip results in different outcomes.

Extra observability: We have designed our processor such that we can observe its state in its most detailed form. All the pipeline FFs on the processor are memory-addressable and therefore, readable using the program. Furthermore, we have the option to copy the state of all the FFs in an identical set of shadow FFs utilizing an input signal that we control from outside the chip. This allows us to take a snapshot of the processor's state at any desired time, particularly right after the EM injection.

Timing Sensor: As discussed earlier a configurable delay-based timing sensor is integrated into the chip. To look for its efficiency against EMFI, we need a microprocessor that has at least one timing sensor on it.

So, this target device provides all necessary requirements needed for analyzing the effect of EMFI on a microprocessor. On the contrary to commercial processors, it is not like a black box. That is why we can explore and understand the EMFI effects on it much better.

Chapter 4

A Bottom-up Analysis of EMFI on a RISC Microprocessor

As discussed earlier, Electromagnetic fault injection (EMFI) is an efficient class of physical attacks that can compromise the immunity of secure cryptographic algorithms. Despite successful EMFI attacks, the effects of electromagnetic injection (EM) on a processor are not well understood. In this chapter, we present a bottom-up analysis of EMFI effects on a RISC microprocessor. We study these effects at three levels: at the wire-level, at the chip-network level, and at the gate-level considering parameters such as EM-injection location and timing.

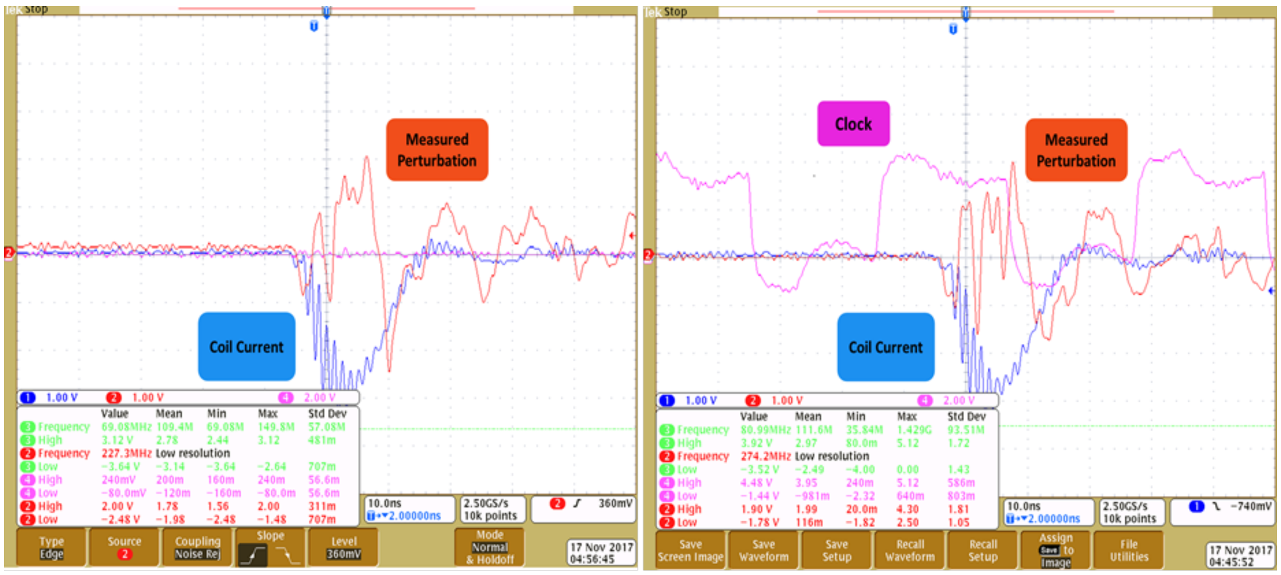
4.1 Wire-level effects of EM injection

An EM injection affects the victim circuit in a fashion that is very similar to a transformer's operation. The coil on the EM probe acts as the primary winding of the transformer. The wires inside the chip form some loops that serve as the secondary winding of the transformer. During EM injection, a variable current flows through the coil, due to coupling some current is generated on the wires. This, in turn, results in changes in the voltage of the wires.

Figure 4.1a shows the effect of the EM injection on one of the pins of the chip that is not powered up. The blue waveform shows the coil current, and the red waveform is the perturbation on the mentioned pin. When the current in the coil ramps up, the wire's voltage is positive while when the current goes down, its voltage is negative. Based on Faraday's law, the magnitude of this voltage depends on the rate that the probe's current fluctuates. As the probe's current dampens, so does the voltage on the wire.

Figure 4.1b shows the same effect on one of the pins when the chip is powered up. The observed pin is a General purpose I/O (GPIO) pin that the processor is driving it to logic '0'. When the current increases, the pin's voltage shoots higher to the point that it can be considered logic '1'. As the current in the coil decreases, the voltage drops back to the ground.

Figures 4.2a-d show the same effect on a clock net. The injection is happening in the low phase of the clock. The difference between the figures is the intensity of injection. While at 30% intensity (Figures 4.2a), the difference is merely a nudge, at 70% (Figures 4.2d) a



(a)

```
1 1.00 V    2 1.00 V              4 2.00 V
              Value      Mean    Min     Max     Std Dev
3 Frequency   69.08MHz   109.4M  69.08M  149.8M  57.08M
3 High        3.12 V     2.78    2.44    3.12    481m
2 Frequency   227.3MHz   Low resolution
3 Low         -3.64 V    -3.14   -3.64   -2.64   707m
4 High        240mV      200m    160m    240m    56.6m
4 Low         -80.0mV    -120m   -160m   -80.0m  56.6m
2 High        2.00 V     1.78    1.56    2.00    311m
2 Low         -2.48 V    -1.98   -2.48   -1.48   707m

10.0ns          2.50GS/s      2  360mV
T▸▾2.00000ns    10k points
17 Nov 2017
04:56:45
```

(b)

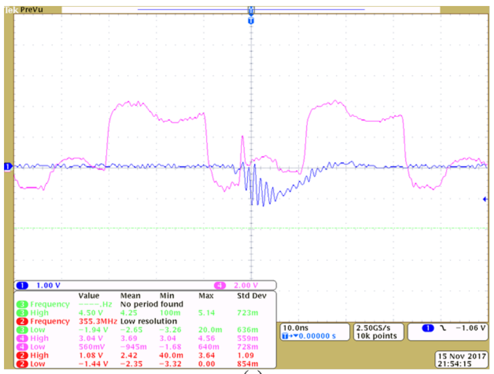
```
1 1.00 V    2 1.00 V              4 2.00 V
              Value      Mean    Min     Max     Std Dev
3 Frequency   80.99MHz   111.6M  35.84M  1.429G  93.51M
3 High        3.92 V     2.97    80.0m   5.12    1.72
2 Frequency   274.2MHz   Low resolution
3 Low         -3.52 V    -2.49   -4.00   0.00    1.43
4 High        4.48 V     3.95    240m    5.12    586m
4 Low         -1.44 V    -981m   -2.32   640m    803m
2 High        1.90 V     1.99    20.0m   4.30    1.81
2 Low         -1.78 V    116m    -1.82   2.50    1.05

10.0ns          2.50GS/s      1  -740mV
T▸▾2.00000ns    10k points
17 Nov 2017
04:45:52
```

Figure 4.1: Effect of EM glitch on a pin. (a) powered-down circuit (b) powered-up circuit

full-grown glitch is observable.

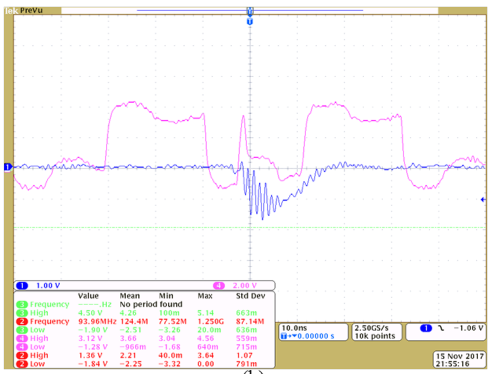
In a digital circuit, the previously observed phenomenon translates into a more specific effect: When the net carries logic 1 (VDD), increasing the current in the coil, does not have any impact as the voltage is capped at VDD, but decreasing of the current can lower the net's voltage. If the electromagnetic field is powerful enough, it can even push the net to logic 0 (GND). On the other hand, when the net carries logic 0 (GND), decreasing the coil current does not affect the voltage, but increasing it can increase the net's voltage, sometimes turning the net to logic 1 (VDD). Figures 4.3a-b captures this phenomenon. In (a), the attack is done while the clock is high and in (b), the attack is performed when the clock is low. In (a), there is no noticeable change in the voltage while the coil's current increases, but the voltage drops to GND as the current decreases. In (b), the clock net is pushed to VDD as



```
               Value      Mean          Min     Max     Std Dev
3 Frequency    ----.Hz    No period found
3 High         4.50 V     4.25          100m    5.14    723m
2 Frequency    355.3MHz   Low resolution
3 Low          -1.94 V    -2.65         -3.26   20.0m   636m
4 High         3.04 V     3.69          3.04    4.56    559m
4 Low          560mV      -945m         -1.68   640m    728m
2 High         1.08 V     2.42          40.0m   3.64    1.09
2 Low          -1.44 V    -2.35         -3.32   0.00    854m
```

```
10.0ns           2.50GS/s      1 ⤵ -1.06 V
T→▼0.00000 s     10k points
                               15 Nov 2017
                               21:54:15
```

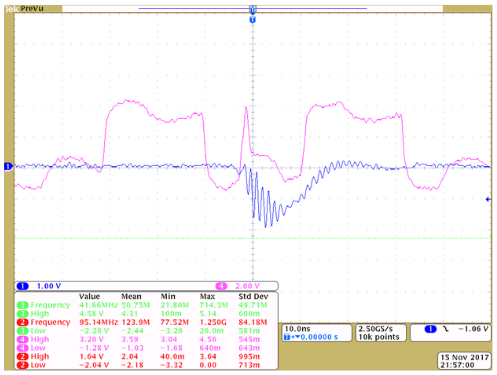
(a)



```
               Value      Mean          Min     Max     Std Dev
3 Frequency    ----.Hz    No period found
3 High         4.50 V     4.26          100m    5.14    663m
2 Frequency    93.96MHz   124.4M        77.52M  1.250G  87.14M
3 Low          -1.90 V    -2.51         -3.26   20.0m   636m
4 High         3.12 V     3.66          3.04    4.56    559m
4 Low          -1.28 V    -966m         -1.68   640m    715m
2 High         1.36 V     2.21          40.0m   3.64    1.07
2 Low          -1.84 V    -2.25         -3.32   0.00    791m
```

```
10.0ns           2.50GS/s      1 ⤵ -1.06 V
T→▼0.00000 s     10k points
                               15 Nov 2017
                               21:55:16
```

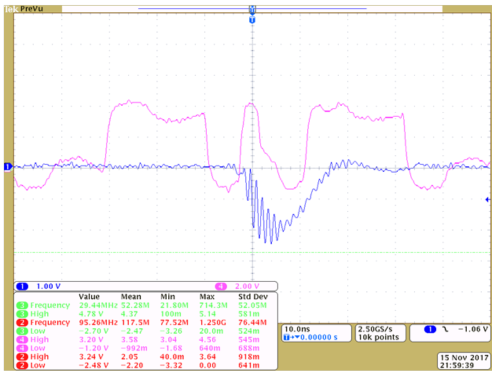
(b)



```
               Value      Mean          Min     Max     Std Dev
3 Frequency    41.66MHz   50.75M        21.80M  714.3M  49.71M
3 High         4.58 V     4.31          100m    5.14    600m
2 Frequency    95.14MHz   123.9M        77.52M  1.250G  84.18M
3 Low          -2.26 V    -2.44         -3.26   20.0m   581m
4 High         3.20 V     3.59          3.04    4.56    545m
4 Low          -1.28 V    -1.03         -1.68   640m    643m
2 High         1.64 V     2.04          40.0m   3.64    995m
2 Low          -2.04 V    -2.18         -3.32   0.00    713m
```

```
10.0ns           2.50GS/s      1 ⤵ -1.06 V
T→▼0.00000 s     10k points
                               15 Nov 2017
                               21:57:00
```

(c)



```
               Value      Mean          Min     Max     Std Dev
3 Frequency    29.44MHz   52.28M        21.80M  714.3M  52.05M
3 High         4.78 V     4.37          100m    5.14    581m
2 Frequency    95.26MHz   117.5M        77.52M  1.250G  76.44M
3 Low          -2.70 V    -2.47         -3.26   20.0m   524m
4 High         3.20 V     3.58          3.04    4.56    545m
4 Low          -1.20 V    -992m         -1.68   640m    688m
2 High         3.24 V     2.05          40.0m   3.64    918m
2 Low          -2.48 V    -2.20         -3.32   0.00    641m
```

```
10.0ns           2.50GS/s      1 ⤵ -1.06 V
T→▼0.00000 s     10k points
                               15 Nov 2017
                               21:59:39
```

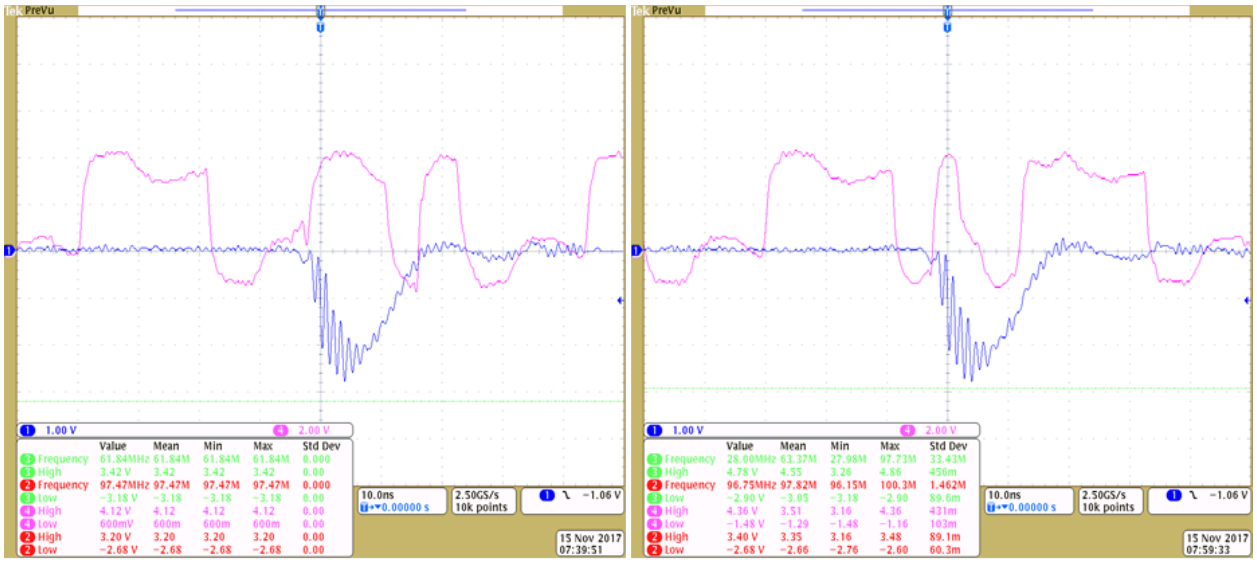
(d)

Figure 4.2: Effect of glitch intensity on the EM perturbation. (a)30%, (b)40%, (c)60%, (d)70%

the current increases and drops back to GND when the current decreases.

4.2 Chip-level Effects of EM injection

In the previous section, we showed the impact of injection on individual wires/pins on the chip. During an EM injection, the electromagnetic field is the strongest right under the probe. As we move further from the probe, the EM is reduced and so does its effect on the wire regarding altering the voltage. Nevertheless, EM can use the global networks on the



(a)

	Value	Mean	Min	Max	Std Dev
3 Frequency	61.84MHz	61.84M	61.84M	61.84M	0.000
3 High	3.42 V	3.42	3.42	3.42	0.00
2 Frequency	97.47MHz	97.47M	97.47M	97.47M	0.000
3 Low	−3.18 V	−3.18	−3.18	−3.18	0.00
4 High	4.12 V	4.12	4.12	4.12	0.00
4 Low	600mV	600m	600m	600m	0.00
2 High	3.20 V	3.20	3.20	3.20	0.00
2 Low	−2.68 V	−2.68	−2.68	−2.68	0.00

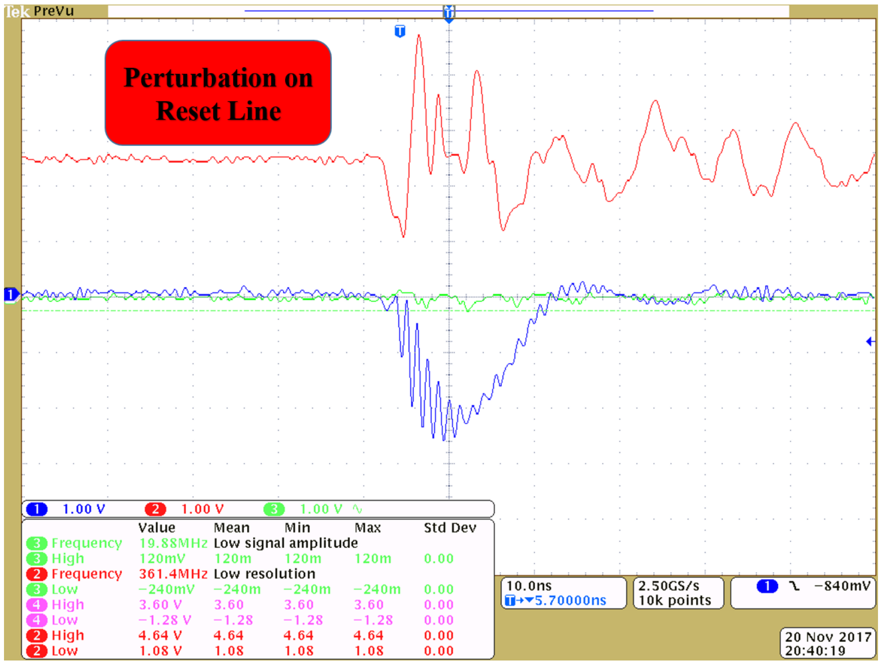
(b)

	Value	Mean	Min	Max	Std Dev
3 Frequency	28.00MHz	63.37M	27.98M	97.73M	33.43M
3 High	4.78 V	4.55	3.26	4.86	456m
2 Frequency	96.75MHz	97.82M	96.15M	100.3M	1.462M
3 Low	−2.90 V	−3.05	−3.18	−2.90	89.6m
4 High	4.36 V	3.51	3.16	4.36	431m
4 Low	−1.48 V	−1.29	−1.48	−1.16	103m
2 High	3.40 V	3.35	3.16	3.48	89.1m
2 Low	−2.68 V	−2.66	−2.76	−2.60	60.3m

Figure 4.3: Effect of EM injection on the clock signal. Two plots show injecting during (a) high phase, and (b) low phase of the clock.

chip to extend its influence. In this section, we study such effects.

There are three global networks on a typical chip: power network, clock network, and reset network. The power network is designed such that it withstands noise and other coupling effects. It is excessively redundant to ensure there is always enough power to keep the circuit in working condition without slowing down. Designers perform elaborate voltage drop analysis to find weak points in the network and correct it by adding more power pins, power straps or power rings. With such sophisticated design, it is not surprising that when we examine power or ground pins that are relatively far from the point of attack, we cannot observe any effect. In other words, the power network does not allow the influence of a single affected power pin to propagate to the rest of the chip.



	Value	Mean	Min	Max	Std Dev
3 Frequency	19.88MHz	Low signal amplitude			
3 High	120mV	120m	120m	120m	0.00
2 Frequency	361.4MHz	Low resolution			
3 Low	−240mV	−240m	−240m	−240m	0.00
4 High	3.60 V	3.60	3.60	3.60	0.00
4 Low	−1.28 V	−1.28	−1.28	−1.28	0.00
2 High	4.64 V	4.64	4.64	4.64	0.00
2 Low	1.08 V	1.08	1.08	1.08	0.00

Figure 4.4: Effect of the EM injection on top of the reset pin

Another global network is the clock network. As for the clock, it is designed to ensure a high speed in distributing the clock. There are numerous buffers on the clock tree to boost the signal and allow fast transfer of the clock voltage to all corners of the chip. We expect that this works in favor of an attacker. If an attacker affects the root of the clock tree or one of the nodes close to the root, we expect that the malicious changes in the voltage will propagate quickly throughout the chip. Unlike power network where we have access to many power and ground pins around the chip, we are not able to directly observe the clock in various locations to verify our hypothesis. Instead, we will use circumstantial clues to support this theory, which we will present in the next section.

Finally, the reset network is used to reset all the FFs in the design to zero. A typical reset network is not as fast as a clock network to reduce area and power consumption. On our chip, the reset is active low and in its inactive state is driven by a high current voltage regulator which does not allow the reset pin to deviate from high voltage. Figure 4.4 shows the effect of the EM injection on top of the reset pin. We can see that the reset voltage does not change to the point that causes a reset. It is possible that inside the chip, local reset wires get affected by EM injection and induce a "reset" on some FFs. We will consider this possibility in the next section.

4.3 Gate-level effects of EM injection

In this section, we make an effort to explain how the EM effects at the wire/chip level manifest themselves at the gate-level to the point that puts the microprocessor in a faulty state. Based on the observations in the previous two sections, we can envision three scenarios that turn the processor's state into an erroneous state:

1: Some of the inputs of the FFs swing to the incorrect logic, and while they are in this state, the positive edge of the clock arrives, and the FF captures this wrong value.

2: The EM injection activates the reset wire for some of the FFs, and that forces those FFs to go to logic '0', which can be a wrong value for some of those FFs. Since our design uses a synchronous reset, this can also only happen at the rising edge of the clock.

3: The EM injection creates a clock glitch. Similar to a timing attack, the FF opens its gates and latches the values before the correct output has been calculated resulting in a faulty state.

We think that one or more of the above situations might be at works when a fault happens. However, we do not have access to internal wirings of the chip and cannot prove or disprove any of the above cases by observing them. Nevertheless, we can use some indirect clues to conclude which ones are more likely to be the source of the errors. Based on our experiments, we believe most of the EM injection faults are caused due to glitches in the clock network. We provide two arguments to support our claim:

4.3.1 Temporal argument: Effect of the timing of the injection

Figure 4.5 shows the clock under attack with varying offset values. The numbers on the top left corner of each plot shows the ratio of the number of corrupted FFs to the total number of FFs. These results are the average for several points on the chip. We can see that the effect varies from no glitches (cases (b) and (d) with 0% faulty FFs), to one glitch (cases (a), (c) and (d) with 44%, 13%, and 23% faulty FFs), and finally to two glitches (case (f) with 72% faulty FFs). All of these cases can be explained by the phenomenon described in Section 4.1. If any of scenarios 1 or 2 described earlier in this section were true, we should only observe faults for case (f), where the attack happens shortly before the rising edge of the clock. Nevertheless, we can see that the number of faults is non-zero for cases (a), (c), and (e) despite the fact that the injection is not close to the rising edge. Instead, the number of faults becomes 0 when the clock is not affected, namely cases (b) and (d). These results

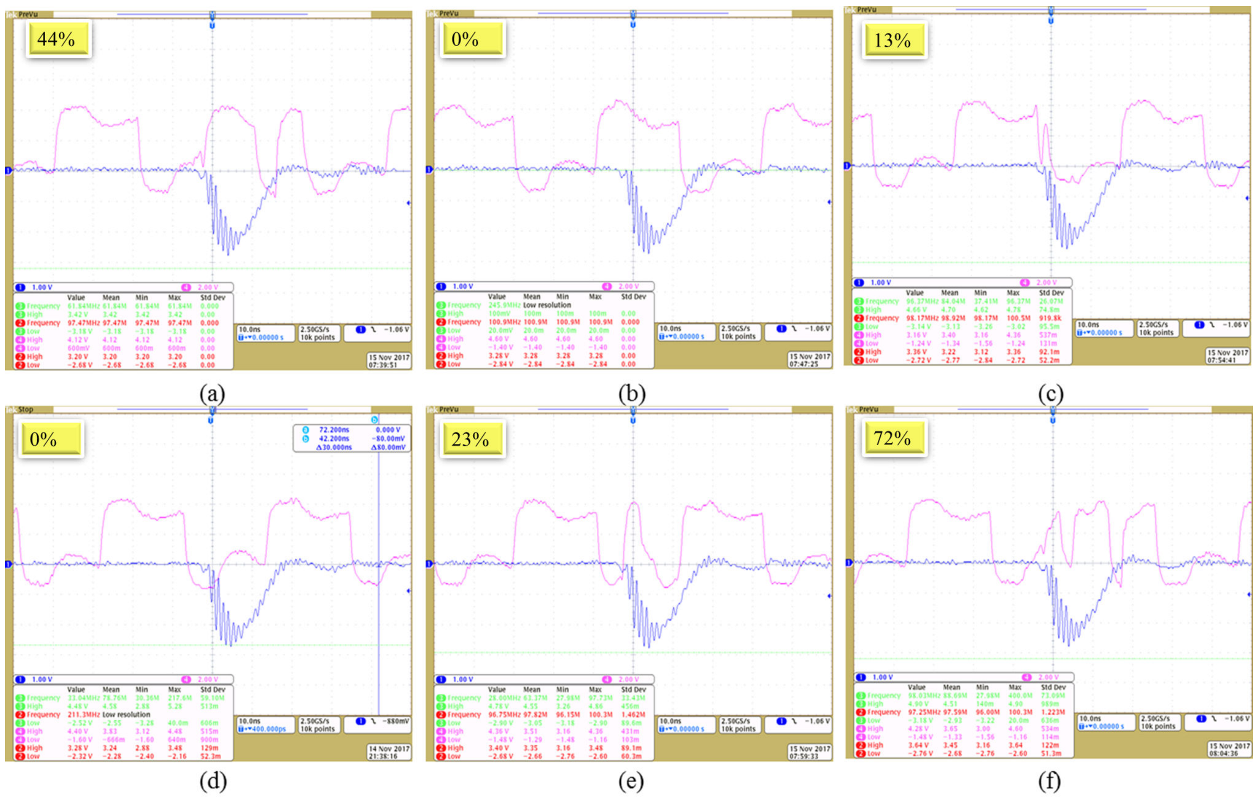


Figure 4.5: Glitch offset effect on clock. (a) offset = 0ns, (b) offset = 10ns, (c) offset = 20ns, (d) offset = 25ns, (e) offset = 30ns, (f) offset = 40ns

.

strongly suggest that the EM faults happen due to disturbances in the clock caused by EM.

4.3.2 Spatial argument: Effect of the attack location relative to the clock tree

Figure 4.6 presents an image of our chip layout where the top levels of the clock tree and all the FFs are highlighted. It also shows two points where we inject EM to explore the effects

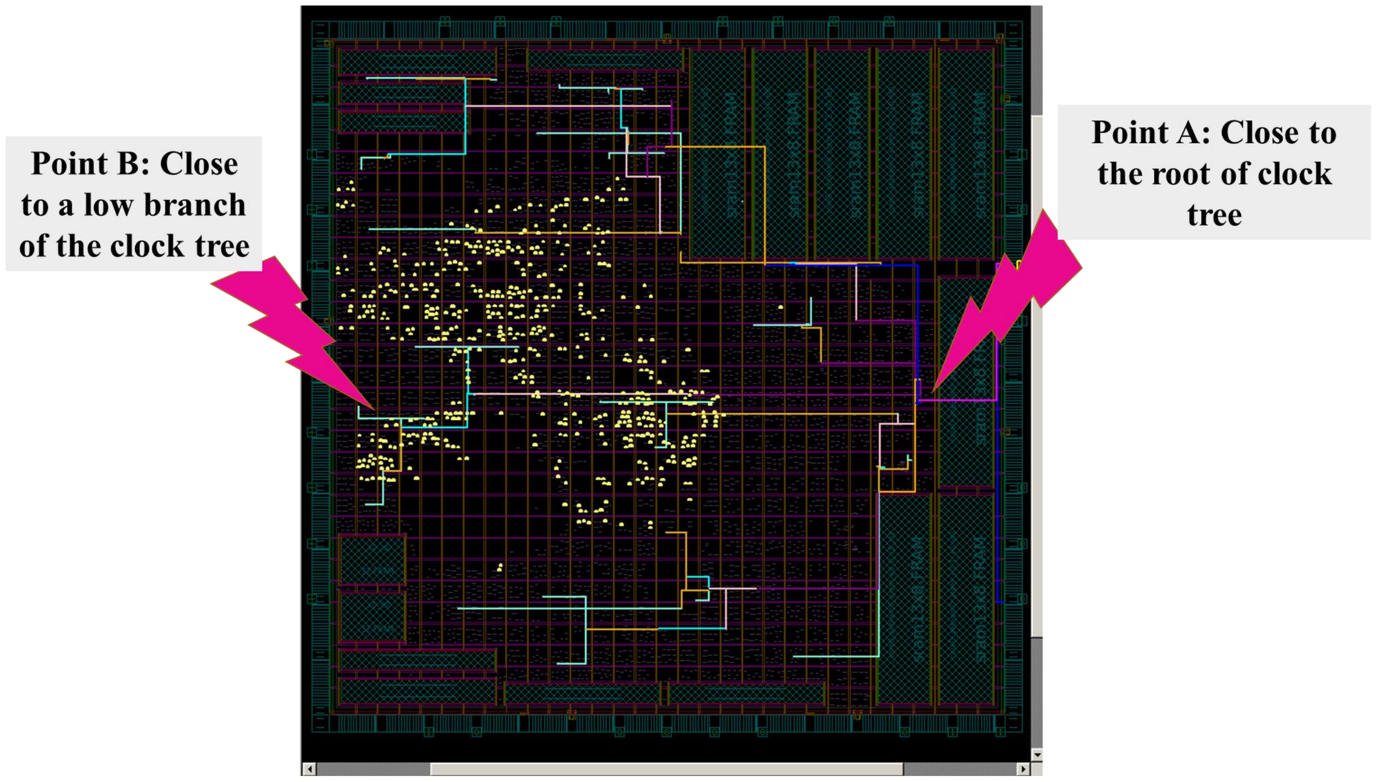
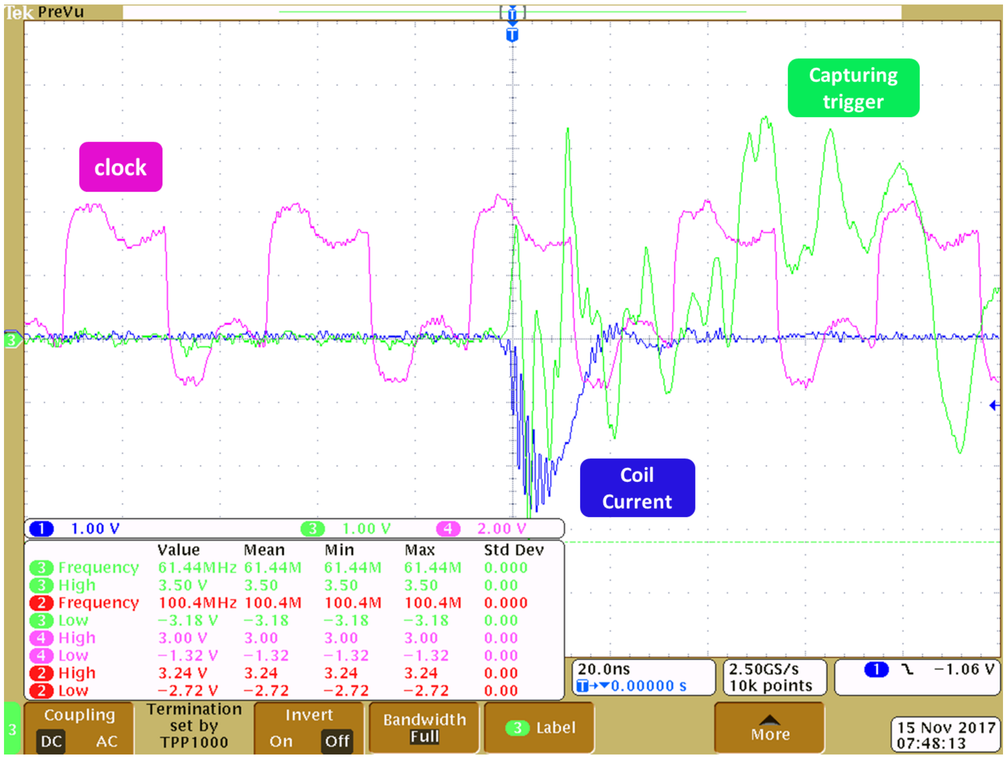

Point B: Close to a low branch of the clock tree
Point A: Close to the root of clock tree

Figure 4.6: Clock tree, observable FFs and two injection points on the layout of the processor

of EMFI on different locations on the clock tree. While the injection point A is very close to the root of the clock, point B sits at much lower branches of the clock tree (level 15). As discussed earlier, we can take a snapshot of the processor state using a trigger, which is set one clock cycle after EM injection. The trigger is shown in figure 4.7.



	Value	Mean	Min	Max	Std Dev
3 Frequency	61.44MHz	61.44M	61.44M	61.44M	0.000
3 High	3.50 V	3.50	3.50	3.50	0.00
2 Frequency	100.4MHz	100.4M	100.4M	100.4M	0.000
3 Low	−3.18 V	−3.18	−3.18	−3.18	0.00
4 High	3.00 V	3.00	3.00	3.00	0.00
4 Low	−1.32 V	−1.32	−1.32	−1.32	0.00
2 High	3.24 V	3.24	3.24	3.24	0.00
2 Low	−2.72 V	−2.72	−2.72	−2.72	0.00

Figure 4.7: The trigger signal which is used to capture the pipeline registers' values

In our program, we read the content of these FFs and record them in a file. We create two files, one after the attack and one for correct operation. Using some scripts, we discover all the faulty FFs. Figure 5.5a-b shows the corrupted FFs on the layout for each of the attack points described earlier. When we inject the EM close to the root of the tree, there are 146 affected FFs, and they are spread all over the chip whereas, for the other case, there are only 24 corrupted FFs, and they are compacted around the injection point. Based on these results, we conclude that the clock tree is propagating the glitch effect from the higher levels to the lower branches of the clock tree, confirming that the EM attack, in essence, is a timing attack. We verified this observation at several other points on the chip with different programs, and we always observe similar outcomes.

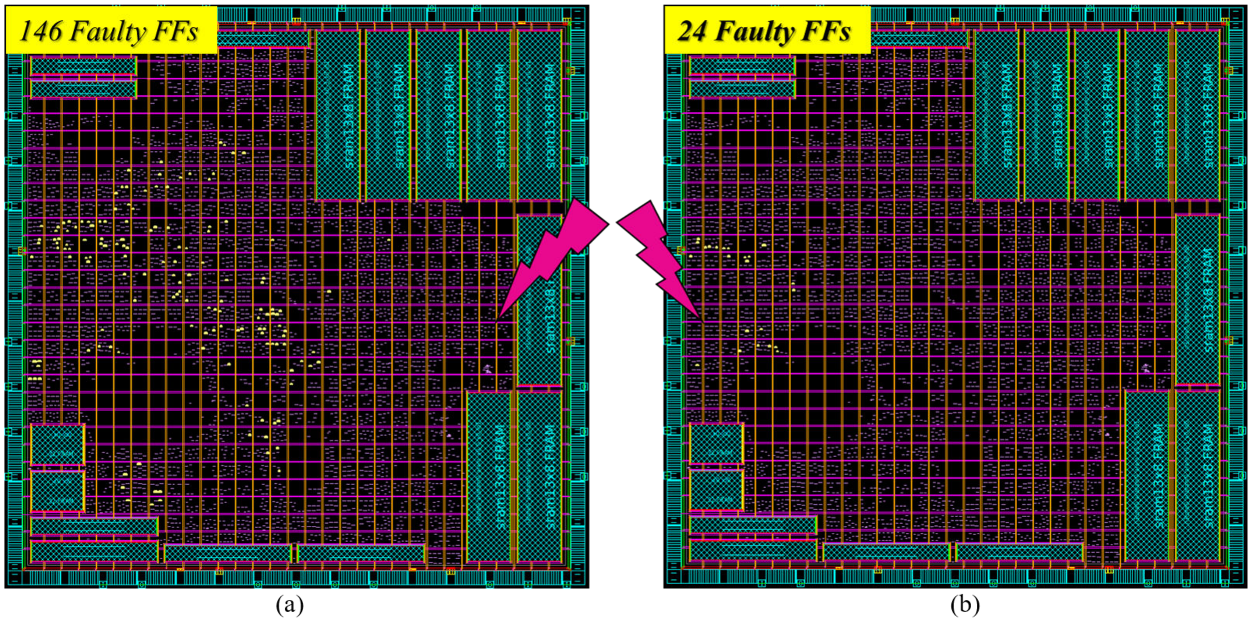


Figure 4.8: Faulty FFs on the layout. (a) injection point on the root of clock tree, (b) injection point on the lower levels of clock tree

Chapter 5

EM Countermeasure

Based on our experimental observations and analysis, we could postulate that EMFI induces local timing faults in a chip. To further validate our hypothesis, we integrated a configurable timing sensor into our microprocessor. The architecture of the implemented timing sensor, as well as details about the detection mechanism used on it, are presented in section 5.1. Section 5.2 provides the necessary steps toward integration of the mentioned timing sensor into a microprocessor. Finally, section 5.3 demonstrates experimental results to show the effectiveness of the timing sensor for detecting EMFI.

5.1 Principal of the integrated timing sensor

The timing sensor which is demonstrated in figure 5.1 monitors the propagation delay in the critical path of the circuit and raises the alarm if there are violations of the setup time constraints [15].

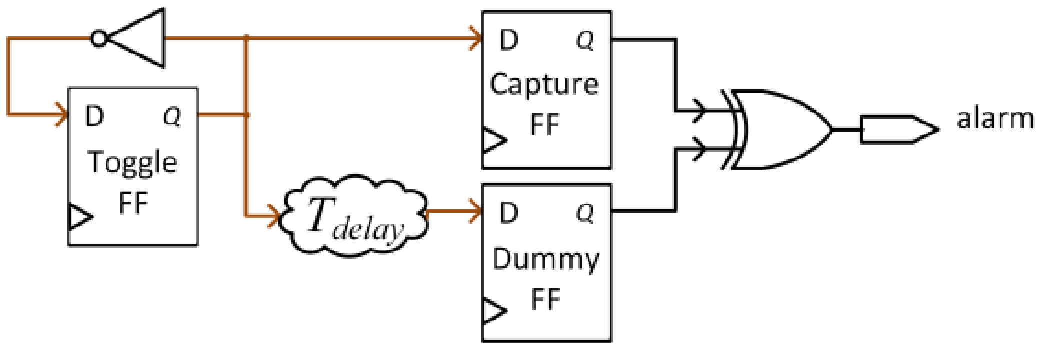


Figure 5.1: Timing sensor structure [29]

It consists of three flip-flops, toggle, capture and dummy flip-flops. There is a specific amount of delay before the dummy flip-flop which is less than a clock cycle. During a normal operation, the dummy flip-flop receives the data after the mentioned delay. Finally, the output of these two flip-flops will be XORed. Since the dummy flip-flop receives the data after a delay but before the next clock cycle, the alarm signal which is the xor of the two capture and dummy flip-flops will be zero. However, if a clock glitch is injected into the state of the processor, the clock period becomes smaller. As a result, the delay value becomes bigger than the clock period, and the dummy flip-flop receives the data in the next clock cycle. So, the capture and dummy flip-flops will have different values, and the alarm will be raised.

5.2 Integration of the timing sensor into the microprocessor

In this section, we provide all the steps needed to integrate a timing sensor into a microprocessor. The design flow which is shown in figure 5.2 consists of RTL design and synthesis which is called front-end and place and route which is called back-end.

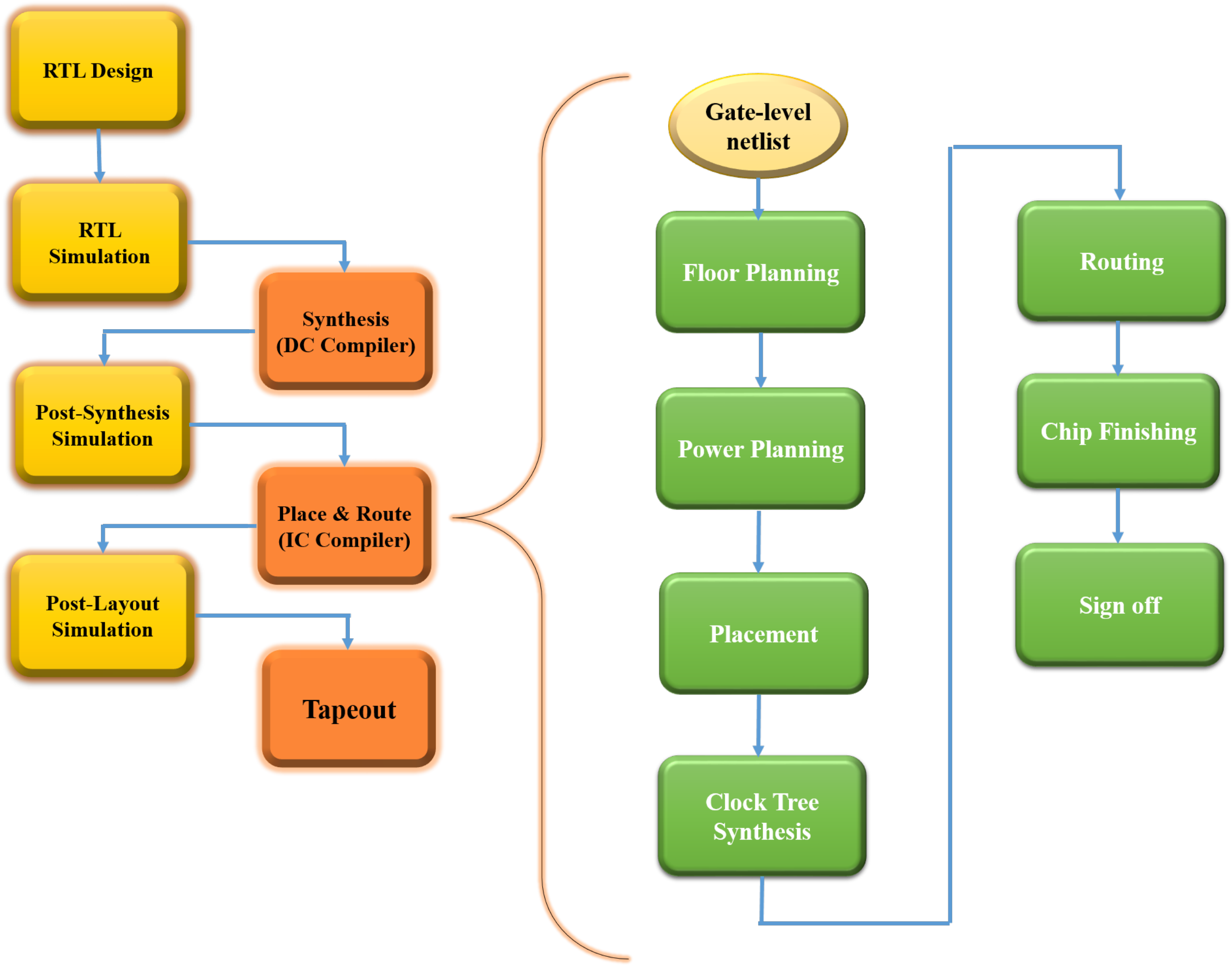


Figure 5.2: Design flow

In the RTL design, the timing sensor code in VHDL is combined with the processor code, and after the RTL simulation and synthesizing, a gate-level netlist of the design is extracted and imported to a software tool called IC Compiler. We used IC compiler from Synopsys company to implement the back-end of the manufactured chip. Back-end flow which is shown with the green diagrams at the right-hand side of figure 5.2 consists of floor planning, power planning, placement, clock tree synthesis, routing, chip finishing and finally sign-off verification. Each of the mentioned steps is explained in more details below.

5.2.1 Floorplanning

All the steps of floorplanning are as follows:

1. First, the tool environment setup and the technology libraries should be prepared. All paths to the standard cell library, I/O pad library, bonding pad library and memory library should be set.

2. The gate-level netlist which is extracted after synthesis, as well as the SDC (design constrained) file should be imported to the IC Compiler.

3. The floorplan of the chip is created in this step. The core utilization, the chip boundary, and the chip size should be specified here.

4. The gaps between I/O pads should be filled with I/O filler cells. To do so, we are using command *insert_pad_filler*.

5. The I/O pads should be powered up. The command *create_pad_rings* makes the power rails to go through all I/O pads.

6. Memory macros are placed at this step. We create a keep-out margin of all the memory macros and fix their placement.

The script of floorplanning is attached in Appendix A to provide more information.

5.2.2 Powerplanning

Once the desired floorplan was created, we move on to design power networks for the whole chip. The steps to create the power networks are as follows.

1. The power plan script starts by defining power network synthesis (PNS) constraints, including layer constraints, power rings and straps constraints, and global constraints. For our FAME chip, we make use of METAL 5 layer for horizontal distribution of the straps and METAL6 layer for vertical distribution of the straps.

2. The *no_routing_over_macros* constraint is enabled making sure that we do not route power lines over the memory macros.

3. We define constraints on the power and ground rings that are automatically created around plan groups and macros.

4. Then we synthesize power network based on the specified constraints.

5. The power network (power/ground wires and vias) based on power network synthesis (PNS) results are committed.

The script for power planning is attached in Appendix B.

5.2.3 Placement

In placement process, the tool will place all the cells at the appropriate places and try to optimize the area, power, etc. The steps necessary for the placement are mentioned below.

1. The placement process starts by connecting the pins in instances to power and ground targets. Next, it hooks up power pins on all macros and standard cells to the power and ground rails.

2. The *set_pnet_options* command prevents standard cells to be placed under PNS layer to avoid power strap violations.

3. We place hard macros and standard cells virtually using timing-driven placement mode while preventing the grouping of cells within the same hierarchal block.

4. This is followed by global routing on the design using Zroute and timing optimization on the design.

5. We perform the actual placement while enabling area recovery for the cells not on the critical timing paths. The use of congestion removal algorithms for improved routability and scan chain optimization during placement is ensured.

6. An incremental logic optimization will be performed to optimize power, area as well as fix the design rule violations.

The script for placement is attached in Appendix C.

5.2.4 Clock tree synthesis

Clock Tree Synthesis (CTS) is a process which makes sure that the clock gets distributed evenly to all sequential elements in a design. We present the steps of CTS in a microprocessor below.

1. The clock network synthesis starts by setting the target skew/latency constraints for clock trees. For our chip, the target skew and clock uncertainty are fixed to 0.1ns.

2. We force the CTS to use clock-double-spacing rules on all clock route segments, except for the first level sinks. We also restricted the clock routes to metal layers 3 to 5.

3. The *set_clock_tree_references* command makes use of specific buffers, inverters to perform CTS. By using this, we can use only buffers and inverters that have balanced rising and falling time.

4. We also set the clock delay calculator to Arnoldi. This makes the tool to use the more accurate Arnoldi-based delay models on clock nets.

5. We perform clock tree synthesis and optimize it to improve the area and scan chain routing.

6. We make sure that all the setup and hold violations are fixed.

The script for clock tree synthesis is attached in Appendix D.

5.2.5 Routing

The routing of the chip consists of four main steps presented below.

1. The routing process starts by setting the Zroute as the tool to route our chip. The medium effort level is used to optimize wire length and via counts.
2. We set the options common to all phases of Zroute routing. For our ASIC, the automatic redundant via insertion after each detail routing step is set to medium.
3. We perform one-step global routing, track assignment, and detail routing.
4. *route_zrt_eco* command is used to perform ECO routing. It connects the open nets as well as fixed the DRC violations.

The script for routing is attached in Appendix E.

5.2.6 Chip finishing

Chip finishing is the process of preparing the final layout for fabrication. This consists of several steps:

1. We check the design for any form of violation such as timing, DRC, LVS, antenna errors. In case we observe some errors, we need to fix them before proceeding further. The commands used to report and fix these errors are presented in Appendix F.

2. We use filler cells to cover empty spaces between standard cells to ensure continuous power and ground rail connections in each row. We perform this step after routing is done, but before metal filling.

3. We insert redundant vias into the chip for a similar reason as previous step.

4. We fill the chip with metal layers to meet the metal density rules.

5. We connect the power, ground pins to respective VDD, VSS network for the pins which might have got the power connections removed during optimization or fixing antenna errors.

6. We write the GDSII of the design into a file. This GDSII is the final output from the IC compiler ready for physical verification using Calibre from Mentor Graphics.

All the commands used for this section mentioned in Appendix F.

The final layout with the timing sensor integrated into it is shown in figure 5.3. The bold white thing is the mentioned timing sensor. Due to having only one timing sensor in this design, the IC Compiler placed the sensor itself somewhere on the chip surface. However, if there is more than one timing sensor that needs to be placed on the chip surface, we should create macros out of them and place them at different locations. The placement of sensors should be fixed until the tool cannot change it during optimization. After extracting the GDSII from the final layout, and checking it for DRC/LVS/Antenna violation in the second verification tool (Calibre from Mentor Graphics), the chip sent for fabrication.

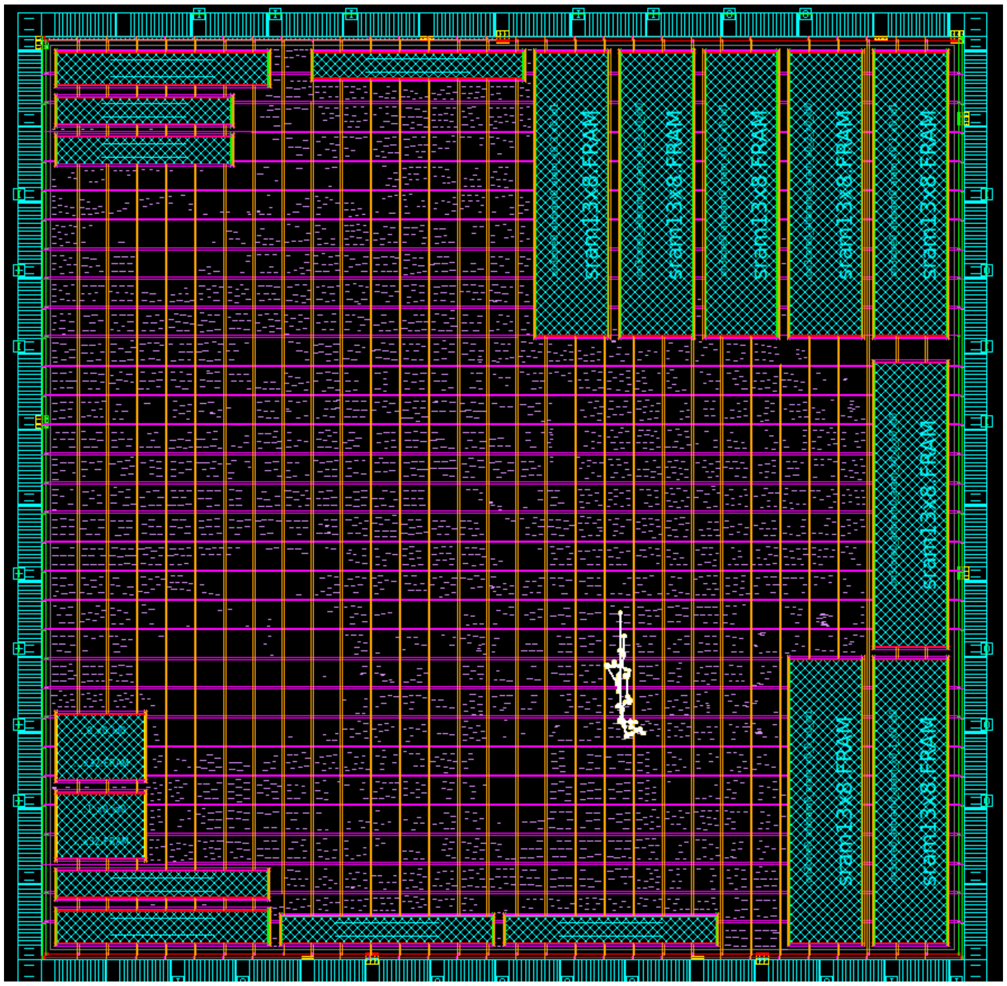


Figure 5.3: Layout of the chip with the timing sensor highlighted on it

As mentioned in chapter 3, the chip is built in 180 nm TSMC, with a frequency of up to 80 MHz. We mounted the chip on a custom-made PCB board and placed the PCB on a Sakura-G FPGA board. Then we tested the chip functionality, and it worked as it was supposed to.

Finally, the chip went under electromagnetic fault injection to check its integrated sensor's effectiveness against this type of attack. In the next section, the results of testing the sensor are presented.

5.3 Experimental results

We started our experiments by performing EMFI on 100 different locations on the processor (scanning the chip surface 10×10), where each location was attacked 100 times. We found that the timing sensor was able to detect 100% of the injected faults on some locations whereas it could not detect any faults on some other other locations. Then, we started to find an answer for this behavior. We used the same method described in chapter 4 for capturing the affected pipeline registers. We highlighted the faulty pipeline registers on the layout for both detected and undetected cases. We noticed when the pipeline registers are on the same clock tree as the sensor, the detection is successful. This means the EM effect is present on both the pipeline registers and the sensor flip-flops. However, when the pipeline registers that get affected are far from the sensor, more specifically on another clock tree branches, the EM effect is not propagated to the sensor, and as a result, the sensor is not able to detect it. Figure 5.4 a-b show these two cases, (a) is the case which was detected by the sensor, and (b) is the case which remained undetected. To investigate more, we looked to find a relationship between detection ratio of the sensor and the place of EM injection. We noticed that the sensor is able to detect the faults with a very good percentage (mostly

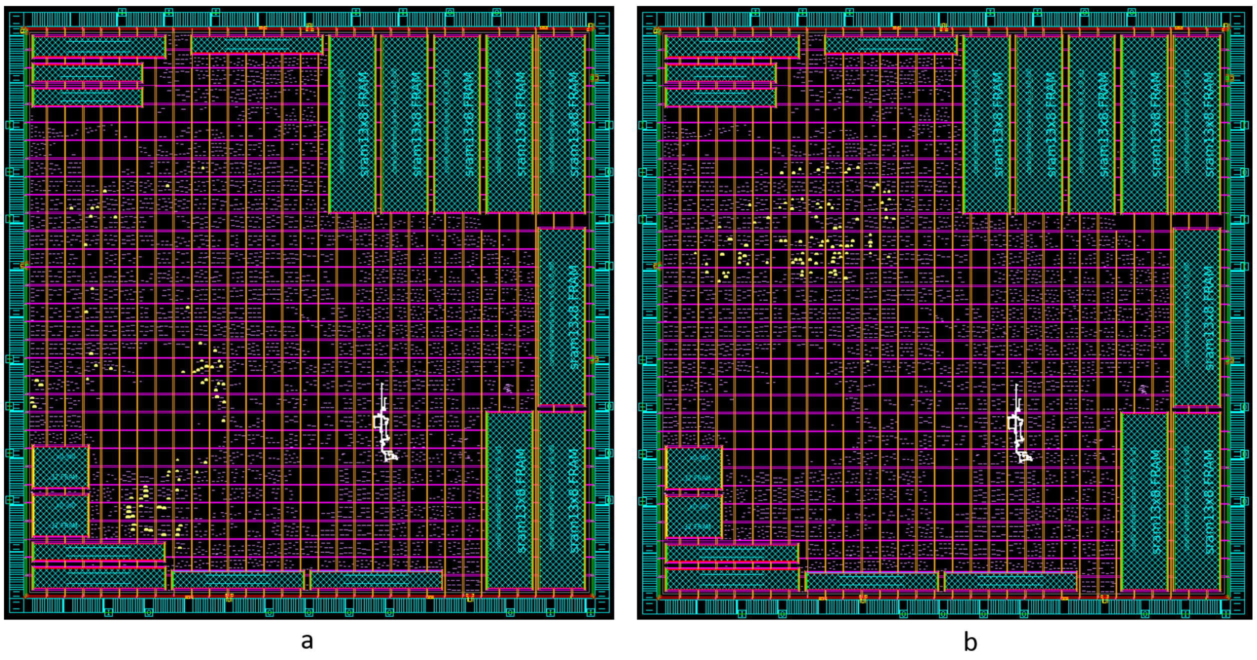


Figure 5.4: detected vs not_detected cases. (a) is detected by the sensor, (b) is not detected by the sensor

100%, for some cases close to 100%) when EM is injected around the root of the clock tree as shown in Figure 5.5a. On the contrary, it is not able to detect any faults when the injection point is on the lower levels of the clock tree similar to the case shown in Figure 5.5b.

figure 5.6 shows the results of the experiment in more detailed format. The numbers in each square shows the percentage of fault injection. Different colors is used in this figure to show the detection ratio of each point. Red color means the detection ratio is 0% while blue color shows the detection ratio of 100%. Gray squares mean no fault has injected at these spots on the chip, while the black squares means UART or IU errors has occurred. On these spots, the processor moved to an error mode and did not provide any response due to EMFI. With

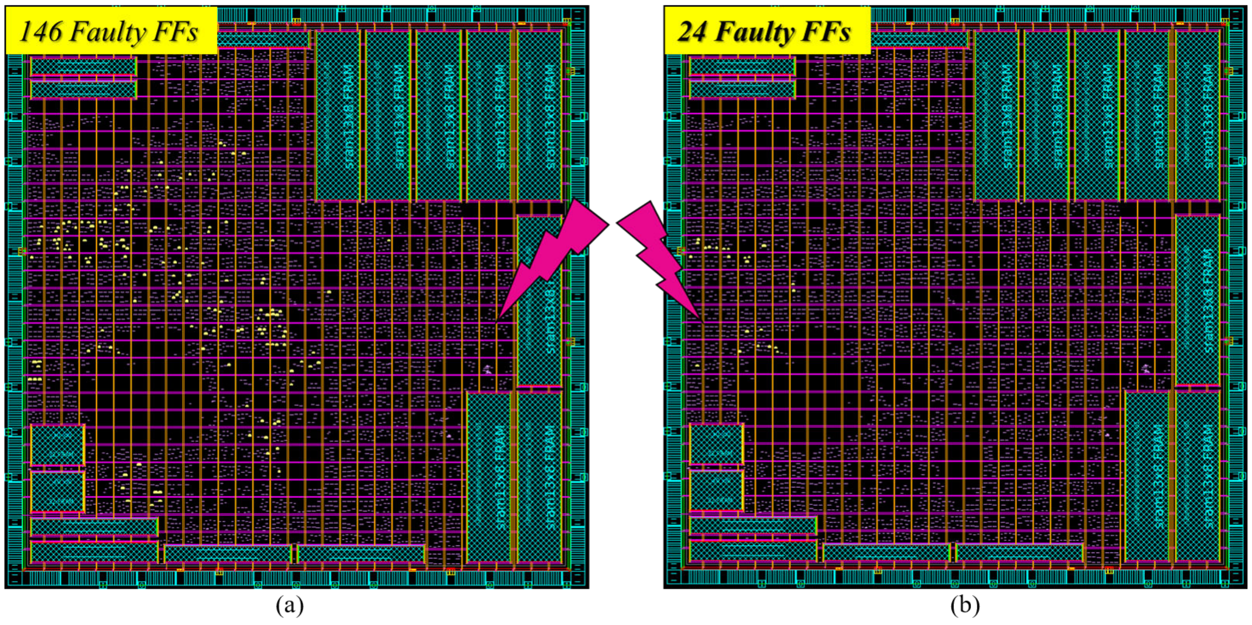


Figure 5.5: (a) injection point on the root of the clock tree which is always detected by the sensor, (b) injection point on the lower levels of the clock tree which is always remained as undetected.

this observations, we could infer that when the fault is injected at the top level of the clock tree, the glitch effect gets propagated to the lower levels including the branch that timing sensor resides. As a result, the timing sensor can detect it. However, when the fault is injected at the lower levels of the clock tree, the effect does not propagate and therefore, remains local. This means the timing sensor observes a healthy clock and assumes no fault has happened.

Suitable timing sensor for EMFI: Based on the above observations and analysis, we can interpret that to increase the percentage of EM fault detection, the number of timing sensors in the chip should increase. Their placing is also very important. The designer needs

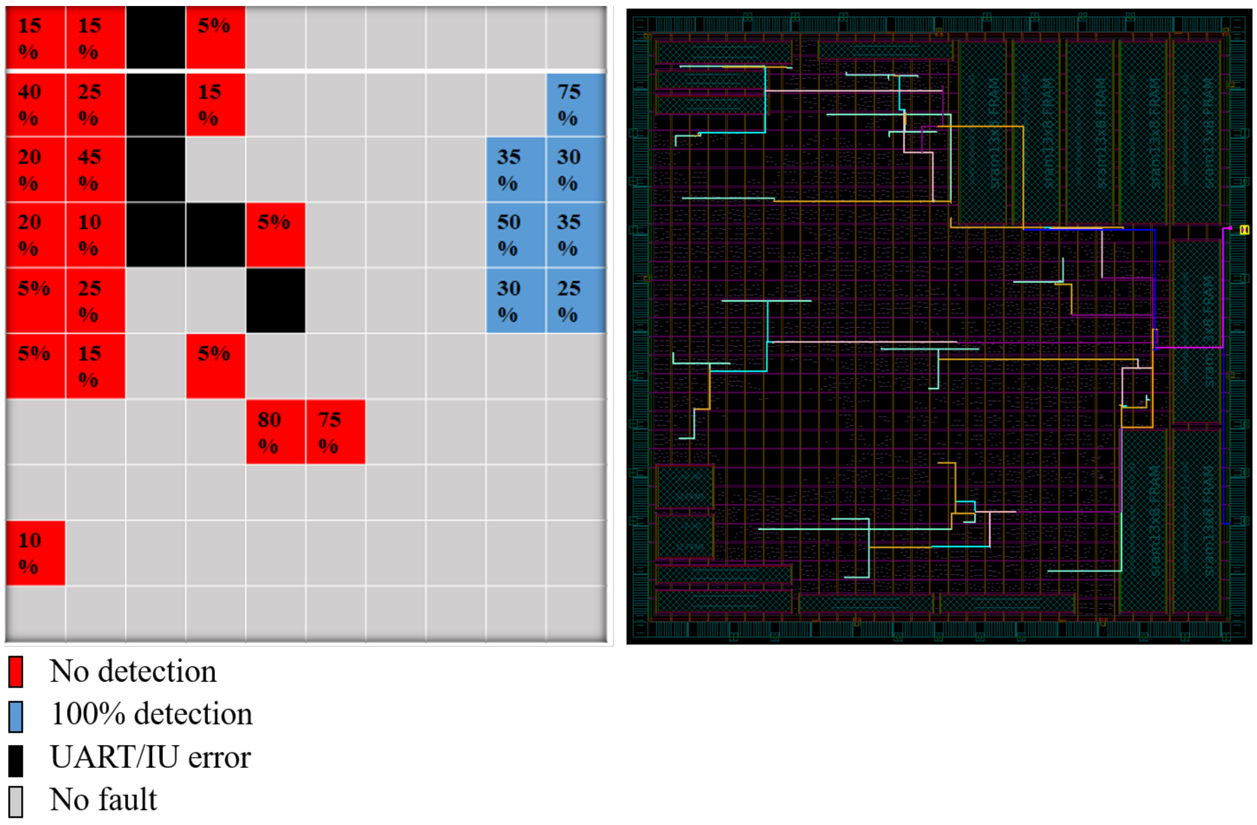


Figure 5.6: Fault injection ratio and detection ratio

to make sure that no matter how far from the root of the clock the injection happens, there is a timing sensor in the vicinity that can catch the fault. The number of needed timing sensors and their locations is part of our future work.

Detecting faulty state versus detecting EM injection: Even at a proper offset, location, and intensity, an EM injection may not be successful to induce faults due to noise effects. For example, when injecting EM at point A (from Figure 4.6), with 80% intensity and 40 ns offset, out of 1000 injections, only 720 of them push the processor into an incorrect state. Our timing detector can catch all these 720 cases and does not raise the alarm for the remaining 280 injections. On the contrary, if we had used a coil to detect the presence of EM filed, the alarm would have been raised for all those 1000 times regardless of whether the injection is causing a fault or not. Depending on the application, one or the other might be desirable. For a typical design where both security and speed are essential, the timing detector is preferable as it prevents wasted cycles during those cases that no fault has occurred. Nevertheless, for some devices where security has the utmost importance in design, the designer might prefer to catch an attack even if it is not causing a fault yet.

Chapter 6

Conclusion

In this research, we made an effort to unravel the way an EM attack affects a microprocessor on different levels of abstraction. We presented a bottom-up analysis of EMFI effects on a RISC microprocessor at three levels: at the wire-level, at the chip-network level, and at the gate-level. At wire-level, we explored the EM effects on the pins/wires of the chip. At chip-network level, we explored the EM effects on all the global networks of the chip, namely power, clock and reset network. From our experiments, we saw that these effects were most apparent in the clock network when compared to the other two global networks. Then, we made an effort to explain how the EM effects at the wire/chip level manifest themselves at the gate-level to the point that puts the microprocessor in a faulty state. Based on our experiments, we believe most of the EM injection faults are caused due to glitches in the clock network. Furthermore, we showed that relative delay and position of the EM injection concerning the clock plays a significant role in the effectiveness of EMFI.

In conclusion, our work suggested that EM attacks behaved similar to timing attacks and could propagate using the clock tree network. Thus, with some modifications, we believe that the current countermeasures for timing attacks like timing detectors can be used to thwart EM attacks. To validate our hypothesis, we integrated a configurable timing sensor into our microprocessor. The timing sensor monitors the propagation delay in the critical path of the circuit and raises the alarm if there are violations of the setup time constraints. We found that the timing sensor was able to detect 100% of the injected faults on some locations whereas it could not detect any faults on some other spots. Based on our observations and results, we could make the inference that when the fault is injected at the top levels of the clock tree, the glitch effect gets propagated to the lower levels including the branch that timing sensor resides. As a result, the timing sensor can detect it. However, when the fault is injected at the lower levels of the clock tree, the effect does not propagate and therefore remains local. We plan to expand this work by integrating more timing sensors into a microprocessor and evaluate their effectiveness against EMFI. We believe the detection ratio will be increased with this change. The number of needed timing sensors and their locations is part of our future work.

Appendix A

floorplanning

```
set_host_options -max_core 72

set tech 180nm

set grtechlibdir ”/opt/libs/tsmc180/extracted”

set cell_path ”$grtechlibdir/tsmc/cl018g/sc9_base_rvt/2008q3v01/”

set cell_lib_path ”$cell_path/db/”

set io_path ”$grtechlibdir/gpio/TPZ018NV/TS02IG502/fb_tpz018nv_280a_r6p0-02eac0/”

set io_lib_path ”$io_path/timing_power_noise/NLDM/tpz018nv_280a/”

set memory_path ”/home/bilgiday/sram_new/”

set memory_lib_path ”$memory_path/db/”
set grtechlibpath ”. $cell_lib_path io_lib_path memory_lib_path”

set grtechtargetlib ”sage-x_tsmc_cl018g_rvt_ss_1p62v_125c.db”

set iolib ”tpz018nvwc.db”

set memlib ”/home/bilgiday/sram_new/db/sram12x8_slow_syn.db /home/bilgiday/sram_new/db/sram6x2
/home/bilgiday/sram_new/db/sram8x25_slow_syn.db /home/bilgiday/sram_new/db/sram9x32_slow_syn.d
/home/bilgiday/sram_new/db/sram6x32_slow_syn.db /home/bilgiday/sram_new/db/sram8x32_slow_syn.d
/home/bilgiday/sram_new/db/sram2p8x32_slow_syn.db /home/bilgiday/sram_new/db/sram13x8_slow_syn

set grtechlinklib ”$grtechtargetlib $iolib $memlib”

set search_path $grtechlibpath

set target_library $grtechtargetlib

set link_library $grtechlinklib

set_tlu_plus_files -max_tluplus /home/bilgiday/sram_new/typical.tluplus -min_tluplus /home-
/bilgiday/sram_new/typical.tluplus -tech2itf_map /home/bilgiday/sram_new/tluplus.map

set icc_input_cel leon3mp_dct

set mw_design_library leon3mp_mw_lib

open_mw_cel $icc_input_cel -library ./$mw_design_library

read_sdc synopsys/leon3mp_dare.sdc

read_def synopsys/leon3mp_scan.def -verbose
```

```
check_scan_chain

report_scan_chain

derive_pg_connection -power_net VDD -ground_net VSS -power_pin VDD -ground_pin VSS

derive_pg_connection -power_net VDD -ground_net VSS -tie

check_mv_design -power_nets

source -echo pad_cell_constraints.tcl

source -echo bonding_pads.tcl

create_floorplan -core_utilization 0.8 -control_type width_and_height -core_width 4516 -core_height
4516 -left_io2core 50 -right_io2core 50 -top_io2core 50 -bottom_io2core 50 -flip_first_row -
start_first_row

insert_pad_filler -cell {PFILLER20 PFILLER10 PFILLER5 PFILLER1 PFILLER05 PFILLER0005
}

create_pad_rings -nets {VDD VDDPST VSS VSSPST}

set_fp_placement_strategy -sliver_size 50

set_fp_placement_strategy -macros_on_edge on -sliver_size 50 -virtual_IPO on

set_keepout_margin -type hard -all_macros -outer 20 20 20 20

create_fp_placement -timing_driven -no_hierarchy_gravity

set_dont_touch_placement [all_macro_cells]
```

save_mw_cel -as floorplan_cell

Appendix B

Powerplanning

```
set_fp_rail_constraints -add_layer -layer METAL5 -direction horizontal -max_strap 128 -min_strap
16 -max_width 12 -min_width 8 -spacing minimum

set_fp_rail_constraints -add_layer -layer METAL6 -direction vertical -max_strap 128 -min_strap
16 -max_width 12 -min_width 8 -spacing minimum

set_fp_rail_constraints -set_ring -nets {VDD VSS} -horizontal_ring_layer { METAL3 } -vertical_ring_layer
{ METAL4 } -ring_max_width 20 -ring_min_width 15 -extend_strap core_ring

set_fp_rail_constraints -set_global -no_routing_over_hard_macros

set_fp_block_ring_constraints -add -horizontal_layer METAL5 -horizontal_width 8.000 -horizontal_offset
0.600 -vertical_layer METAL6 -vertical_width 6.000 -vertical_offset 0.600 -block_type instance
-block {core0_leon3core0_leon3s0_leon3x0_cmem0_itags0_0_x0_id0 core0_leon3core0_leon3s0_leon3x0_rf0_rhu
core0_leon3core0_ahbram0_aram_x0_2_x0_id0 core0_leon3core0_ahbram0_aram_x0_2_x0_id1 core0_leon3core0
```

```
core0_leon3core0_ahbram0_aram_x0_0_x0_id0 core0_leon3core0_ahbram0_aram_x0_0_x0_id1 core0_leon3core0

core0_leon3core0_ahbram0_aram_x0_3_x0_id0 core0_leon3core0_ahbram0_aram_x0_3_x0_id1 core0_leon3core0

core0_leon3core0_leon3s0_leon3x0_cmem0_idata0_0_x0_id0 core0_leon3core0_leon3s0_leon3x0_rf0_rhu_x1_x0_i

core0_leon3core0_ahbram0_aram_x0_1_x0_id0 core0_leon3core0_ahbram0_aram_x0_1_x0_id1 core0_leon3core0

core0_leon3core0_leon3s0_leon3x0_tbmem0_ram0_0_x0_x0_id0 core0_leon3core0_leon3s0_leon3x0_cmem0_dta

-net {VDD VSS}

synthesize_fp_rail -power_budget {1000} -voltage_supply {1.8} -target_voltage_drop {100} -

output_directory ./pna_output -nets {VDD VSS} -create_virtual_rails {METAL1} -synthesize_power_plan

-synthesize_power_pads -use_strap_ends_as_pads -pad_masters VDD:PVDD1DGZ.FRAM VSS:PVSS1DGZ.


commit_fp_rail

save_mw_cel -as Powered_cell
```

Appendix C

Placement

```
preroute_instances

preroute_standard_cells -connect horizontal -remove_floating_pieces -fill_empty_rows -port_filter_mode
off -cell_master_filter_mode off -cell_instance_filter_mode off -voltage_area_filter_mode off -
route_type {P/G Std. Cell Pin Conn} preroute_standard_cells -connect horizontal -remove_floating_pieces
-fill_empty_rows -port_filter_mode off -cell_master_filter_mode off -cell_instance_filter_mode off
-voltage_area_filter_mode off -route_type {P/G Std. Cell Pin Conn}

set_pnet_options -complete {METAL5 METAL6} create_fp_placement -timing_driven -no_hierarchy_gravity

route_zrt_global

optimize_fp_timing -fix_design_rule

define_routing_rule clock_double_spacing -default_reference_rule -multiplier_spacing 2
```

```
report_routing_rule clock_double_spacing

set_clock_tree_options -routing_rule clock_double_spacing -layer_list {METAL3 METAL6} -
use_default_routing_for_sinks 1

place_opt -area_recovery -effort high -congestion -optimize_dft

legalize_placement

psynopt -area_recovery -power -only_design_rule

save_mw_cel -as placed_cell
```

Appendix D

Clock tree synthesis

```
set_clock_tree_options -target_skew 0.1

set_clock_uncertainty 0.1 [all_clocks]

define_routing_rule clock_double_spacing -default_reference_rule -multiplier_spacing 2

report_routing_rule clock_double_spacing

set_clock_tree_options -routing_rule clock_double_spacing -layer_list {METAL3 METAL6} -use_default_routing_for_sinks 1

set_clock_tree_references -references {CLKBUFX1 CLKBUFX2 CLKBUFX3 CLKBUFX4 CLKBUFX8 CLKBUFX12 CLKBUFX16 CLKBUFX20 CLKBUFXL CLKINVX1 CLKINVX2 CLKINVX3 CLKINVX4 CLKINVX8 CLKINVX12 CLKINVX16 CLKINVX20 CLKINVXL}
```

```
check_physical_design -stage pre_clock_opt -display

set_delay_calculation_options -arnoldi_effort medium check_clock_tree

clock_opt -only_cts -no_clock_route

set_fix_hold [all_clocks]

extract_rc

clock_opt -only_psyn -area_recovery -no_clock_route -optimize_dft

route_zrt_group -all_clock_nets -reuse_existing_global_route true

save_mw_cel -as CTS_cell
```

Appendix E

Routing

```
set_route_mode_options -zroute true

set_route_zrt_detail_options -optimize_wire_via_effort_level medium

set_route_zrt_common_options -post_detail_route_redundant_via_insertion medium

route_zrt_auto -max_detail_route_iterations 500 -save_after_detail_route true -save_cell_prefix
routed

route_zrt_eco

derive_pg_connection -power_net VDD -ground_net VSS -power_pin VDD -ground_pin VSS

derive_pg_connection -power_net VDD -ground_net VSS -tie

save_mw_cel -as route_Cell
```

Appendix F

Chip finishing

report_qor

focal_opt -drc_nets all

focal_opt -drc_pins all

route_opt

report_qor

report_constraint -all_violators

set lib [current_mw_lib] remove_antenna_rules $lib

define_antenna_rule $lib -mode 4 -diode_mode 3 -metal_ratio 400 -cut_ratio 20

define_antenna_layer_rule $lib -mode 4 -layer "METAL1" -ratio 400 -diode_ratio 0.203 0 400.00 2200 define_antenna_layer_rule $lib -mode 4 -layer ”METAL2” -ratio 400 -diode_ratio 0.203 0 400.00 2200

define_antenna_layer_rule $lib -mode 4 -layer ”METAL3” -ratio 350 -diode_ratio 0.203 0 400.00 2200

define_antenna_layer_rule $lib -mode 4 -layer ”METAL4” -ratio 400 -diode_ratio 0.203 0 400.00 2200

define_antenna_layer_rule $lib -mode 4 -layer ”METAL5” -ratio 400 -diode_ratio 0.203 0 400.00 2200

define_antenna_layer_rule $lib -mode 4 -layer ”METAL6” -ratio 400 -diode_ratio 0.203 0 8000.00 30000

define_antenna_layer_rule $lib -mode 4 -layer ”VIA12” -ratio 20 -diode_ratio 0.203 0 83.33 75

define_antenna_layer_rule $lib -mode 4 -layer ”VIA23” -ratio 20 -diode_ratio 0.203 0 83.33 75

define_antenna_layer_rule $lib -mode 4 -layer ”VIA34” -ratio 20 -diode_ratio 0.203 0 83.33 75

define_antenna_layer_rule $lib -mode 4 -layer ”VIA45” -ratio 20 -diode_ratio 0.203 0 83.33 75

define_antenna_layer_rule $lib -mode 4 -layer ”VIA56” -ratio 20 -diode_ratio 0.203 0 83.33 75

set_route_zrt_detail_options -insert_diodes_during_routing true -check_antenna_on_pg true -port_antenna_mode jump -macro_pin_antenna_mode jump

route_zrt_detail

```
route_zrt_detail -incremental true

save_mw_cel -as Antenna_free_cell

derive_pg_connection -power_net VDD -ground_net VSS -power_pin VDD -ground_pin VSS

derive_pg_connection -power_net VDD -ground_net VSS -tie

verify_lvs -check_short_locator

verify_pg_nets

verify_drc

insert_stdcell_filler -cell_with_metal ”FILL64 FILL32 FILL16 FILL8 FILL4 FILL2 FILL1”
-connect_to_power {VDD} -connect_to_ground {VSS} -between_std_cells_only

insert_zrt_redundant_vias -effort medium

insert_metal_filler -bounding_box { {175.165 168.730} {4672.595 4666.160} } -out self -
dont_overwrite -width {poly 0.3} -fill_poly
```

Bibliography

[1] http://www.t4f.org/articles/fault-injection-attacks-clock-glitching-tutorial/.

[2] LEON3 processor. http://www.gaisler.com/index.php/products/processors/leon3. [Online; accessed 18-May-2015].

[3] Riscure: EMFI Probe Datasheet. https://www.riscure.com/uploads/2017/07/datasheet_em-fi_transient_probe.pdf.

[4] Riscure: Inspector Datasheet. https://www.riscure.com/security-tools/inspector-sca/.

[5] Riscure: VC Glitcher Datasheet. https://www.riscure.com/uploads/2017/07/datasheet_emprobestation.pdf.

[6] Riscure: VC Glitcher Datasheet. https://www.riscure.com/documents/datasheet_vcglitcher.pdf.

[7] SAKURA hardware security project. http://satoh.cs.uec.ac.jp/SAKURA/index.html. [Online; accessed 29-April-2016].

[8] M. Aarts. Electromagnetic Fault Injection using Transient Pulse Injections. Master's thesis, Eindhoven University of Technology, Netherlands, 2013.

[9] M. Agoyan, J.-M. Dutertre, D. Naccache, B. Robisson, and A. Tria. *When Clocks Fail: On Critical Paths and Clock Faults*, pages 182–193. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[10] 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*, pages 105–114, Sept 2011.

[11] A. Barenghi, L. Breveglieri, I. Koren, and D. Naccache. Fault injection attacks on cryptographic devices: Theory, practice, and countermeasures. *Proceedings of the IEEE*, 100(11):3056–3076, Nov 2012.

[12] J. Breier, S. Bhasin, and W. He. An electromagnetic fault injection sensor using hogge phase-detector. In *Quality Electronic Design (ISQED), 2017 18th International Symposium on*, pages 307–312. IEEE, 2017.

[13] A. Dehbaoui, J.-M. Dutertre, B. Robisson, P. Orsatelli, M. Maurine, and A. Tria. Injection of transient faults using electromagnetic pulses -practical results on a cryptographic system. In *IACR Cryptology ePrint Archive (2012), vol. 2012. informal publication: p. 123*, 2012.

[14] A. Dehbaoui, J. M. Dutertre, B. Robisson, and A. Tria. Electromagnetic transient faults

injection on a hardware and a software implementations of aes. In *2012 Workshop on Fault Diagnosis and Tolerance in Cryptography*, pages 7–15, Sept 2012.

[15] C. Deshpande, B. Yuce, N. F. Ghalaty, D. Ganta, P. Schaumont, and L. Nazhandali. A configurable and lightweight timing monitor for fault attack detection. In *2016 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pages 461–466, July 2016.

[16] C. Deshpande, B. Yuce, P. Schaumont, and L. Nazhandali. Employing dual-complementary flip-flops to detect emfi attacks. In *VLSI (AsianHOST), 2017 Asian Hardware Oriented Security and Trust Symposium*. IEEE, 2017.

[17] S. Govindavajhala and A. W. Appel. Using memory errors to attack a virtual machine. In *2003 Symposium on Security and Privacy, 2003.*, pages 154–165, May 2003.

[18] T. Hummel. Exploring Effects of Electromagnetic Fault Injection on a 32-bit High Speed Embedded Device Microprocessor. Master's thesis, University of Twente, Netherlands, 2014.

[19] J. marc Schmidt and M. Hutter. Optical and em fault-attacks on crt-based rsa: Concrete results.

[20] P. Maurine, K. Tobich, T. Ordas, and P. Y. Liardet. Yet Another Fault Injection Technique : by Forward Body Biasing Injection. In *YACC'2012: Yet Another Conference on Cryptography*, Porquerolles Island, France, Sept. 2012.

[21] N. Miura, Z. Najm, W. He, S. Bhasin, X. T. Ngo, M. Nagata, and J.-L. Danger. Pll to

the rescue: a novel em fault countermeasure. In *Design Automation Conference (DAC), 2016 53nd ACM/EDAC/IEEE*, pages 1–6. IEEE, 2016.

[22] N. Moro, A. Dehbaoui, K. Heydemann, B. Robisson, and E. Encrenaz. Electromagnetic fault injection: Towards a fault model on a 32-bit microcontroller. In *2013 Workshop on Fault Diagnosis and Tolerance in Cryptography*, pages 77–88, Aug 2013.

[23] S. Ordas, L. Guillaume-Sage, K. Tobich, J.-M. Dutertre, and P. Maurine. *Evidence of a Larger EM-Induced Fault Model*, pages 245–259. Springer International Publishing, Cham, 2015.

[24] J.-J. Quisquater and D. Samyde. Eddy current for magnetic analysis with active sensor. 01 2002.

[25] S. Skorobogatov. Semi-invasive attacks: A new approach to hardware security analysis. Technical report, University of Cambridge, 2005.

[26] S. Skorobogatov. Local heating attacks on flash memory devices. In *2009 IEEE International Workshop on Hardware-Oriented Security and Trust*, pages 1–6, July 2009.

[27] S. P. Skorobogatov and R. J. Anderson. Optical fault induction attacks. In *Revised Papers from the 4th International Workshop on Cryptographic Hardware and Embedded Systems*, CHES '02, pages 2–12, London, UK, UK, 2003. Springer-Verlag.

[28] R. Spyk, R. Velegalati, and J. Woudenberg. Electro magnetic fault injection in practice. 2013.

[29] B. Yuce, N. F. Ghalaty, C. Deshpande, C. Patrick, L. Nazhandali, and P. Schaumont. Fame: Fault-attack aware microprocessor extensions for hardware fault detection and software fault response. In *Proceedings of the Hardware and Architectural Support for Security and Privacy 2016*, HASP 2016, pages 8:1–8:8, New York, NY, USA, 2016. ACM.

[30] L. Zussa, A. Dehbaoui, K. Tobich, J.-M. Dutertre, P. Maurine, L. Guillaume-Sage, J. Clediere, and A. Tria. Efficiency of a glitch detector against electromagnetic fault injection. In *Design, Automation and Test in Europe Conference and Exhibition (DATE), 2014*, pages 1–6. IEEE, 2014.

[31] L. Zussa, J.-M. Dutertre, J. Cldire, B. Robisson, and A. Tria. Investigation of timing constraints violation as a fault injection means. 11 2012.