Detecting Electromagnetic Injection Attack on FPGAs Using In Situ Timing Sensors

Surabhi Satyajit Gujar

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

Leyla Nazhandali, Chair

Patrick R. Schaumont

A. Lynn Abbott

July 30, 2018

Blacksburg, Virginia

Keywords: Electromagnetic Fault Injection, Timing Sensor, Hardware Security, FPGA

Detecting Electromagnetic Injection Attack on FPGAs Using In Situ Timing Sensors

Surabhi Satyajit Gujar

Abstract

Nowadays, security is one of the foremost concerns as the confidence in a system is mostly dependent on its ability to protect itself against any attack. The area of Electromagnetic Fault Injection (EMFI) wherein attackers can use electromagnetic (EM) pulses to induce faults has started garnering increasing attention. It became crucial to understand EM attacks and find the best countermeasures. In this race to find countermeasures, different researchers proposed their ideas regarding the generation of EM attacks and their detection. However, it is difficult to see a universal agreement on the nature of these attacks.

In this work, we take a closer look at the analysis of the primary EMFI fault models suggested earlier. Initial studies had shown that EM glitches caused timing violations, but recently it was proposed that EM attacks can create bit sets and bit resets. We performed a detailed experimental evaluation of the existing detection schemes on two different FPGA platforms. We present their comparative design analysis concerning their accuracy, precision, and cost. We propose an in situ timing sensor to overcome the disadvantages of the previously proposed detection approaches. This sensor can successfully detect most of the electromagnetic injected faults with high precision. We observed that the EM attack behaves like a localized timing attack in FPGAs which can be identified using the in situ timing sensors.

Detecting Electromagnetic Injection Attack on FPGAs Using In Situ Timing Sensors

Surabhi Satyajit Gujar

General Audience Abstract

When computers are built only for a specific application, they are called embedded systems. Over the past decade, there has been an incredible increase in the number of embedded systems around us. Right from washing machines to electronic locks, we can see embedded systems in almost every aspect of our lives. There is an increasing integration of embedded systems in applications such as cars and buildings with the advent of smart technologies. Due to our heavy reliance on such devices, it is vital to protect them against intentional attacks. Apart from the software attacks, it is possible for an attacker to disrupt or control the functioning of a system by physically attacking its hardware using various techniques. We look at one such technique that uses electromagnetic pulses to create faults in a system. We experimentally evaluate two of the previously suggested methods to detect electromagnetic injection attacks. We present a new sensor for this detection which we believe is more effective than the previously discussed detection schemes.

Dedication

To My Family

Acknowledgments

I would like to thank my committee members Dr. Leyla Nazhandali, Dr. Patrick Schaumont, and Dr. Lynn Abbott, for their strong support throughout my time at Virginia Tech. I am very grateful to Dr. Nazandali for her constant advice, mentoring and direction which helped me achieve my research goals. I would also like to give my special thanks to my colleagues in the Secure Embedded Systems lab namely Abhishek Bendre, Bilgiday Yuce, Chinmay Deshpande, and Yuan Yao for their prompt help in fixing issues with the setup and answering my questions which helped me to carry forward my research. I would also like to thank Archanaa Krishnan and Daniel Dinu for their supportive and cheerful companionship. I would like to thank the Virginia Tech libraries, Virginia Tech ECE department and the Virginia Tech Graduate School for their contribution in making the masters' thesis a smooth and fulfilling journey for me. I am much thankful to my friends at Virginia Tech for their constant push which motivated to do my best work. Last but not the least, I would like to thank my mom, my dad and my younger brother for their continued confidence in me and my efforts. I have gathered valuable knowledge and essential skills in perseverance and patience during this thesis for which I am eternally obliged to everyone involved.

Contents

List of Figures

Chapter 1

Introduction and Background

1.1 Motivation

The importance of protecting embedded systems from an attack cannot be stressed enough. With the availability of a vast amount of user data, the privacy and security of this data have become a cause of worry. This concern is valid at all levels and organizations ranging from top companies to individual users using embedded systems in so many areas of their daily life. Efforts to make fool-proof embedded devices is a constant contest of finding new ways to attack devices and subsequently its countermeasures. Some attacks in the field of hardware security, like timing attacks and optical attacks, are now quite mature with extensive research and have well-established countermeasures against them.

However, there remains a large area of attacks, namely electromagnetic attacks, whose com-

plete understanding is still an ongoing effort. This understanding is especially crucial for building the best countermeasures for them, in turn, making hardware more and more secure. This work is an effort in this direction to make FPGAs more robust against EMFI attacks. Fault attacks were first introduced in 1997 by Boneh et al. [2] when he discovered that there were errors in computations inside a chip due to radioactive emissions. We discuss fault attacks in detail in the next section.

1. **Hardware Fault Attacks**

   We can broadly divide fault attacks into physical attacks and non-physical attacks. While non-physical attacks do not have any physical access to the hardware, physical attacks can fault by tampering with the hardware or by just observing it. We can further classify the physical attacks into active attacks and passive attacks. Passive attacks, also known as side-channel attacks, take advantage of information leakage from a measurable source such as the power consumption, execution time, electromagnetic radiations, sound and light emissions. These attacks do not cause any disturbance in the target device and instead process the leak in information to extract the secret keys. On the other hand, active attacks modify the circuit behavior to create faults in the circuit's computations. These attacks physically inject faults into a target device by altering it's operating conditions. We also call these attacks as Fault Injection (FI) attacks. We can define fault injection as the process of stressing a target in such a way that it no longer functions according to its specifications. FI attacks are extremely dangerous as the attacker can modify the correct functioning of the embedded systems

creating faults which can be a threat for critical applications.

Based on the attack techniques, we can classify these attacks into the following types.

- **Clock glitching:** In this attack, the attacker alters the external clock signal such that the supplied signal has one or more very short pulses in between the normal clock signal. The clock glitch period is much shorter than the standard clock period. If this glitched clock period drops below the critical path clock period, incorrect values can latch in the memory leading to a wrong state of the circuit. This type of attack affects the global clock network and are therefore said to be global. A clock glitch fault injection is one of the simplest and cheapest ways of injecting faults in a circuit.
- **Voltage glitching:** These attacks involve increasing the voltage of the target embedded device above its threshold value for a fixed or instantaneous period. Power supply spikes can cause the processor to skip instructions or misread values. Similar to clock glitching attacks, voltage attacks are also global.
- **Voltage underfeeding:** Underpowering the power supply is another low-cost fault injection method which will lead to faulty circuit behavior. It mainly will artificially increase the critical path length beyond the standard clock period resulting in fault outputs. However, there is no precise timing control in this type of attack as opposed to the clock glitching attacks. A recent variant of this technique knows as body biasing involves using a needle in contact with the IC silicon to create forward or reverse static biases to modulate the threshold voltages

of CMOS transistors.

- **Temperature attacks:** Electronic devices have a range of acceptable temperatures where they function correctly. If the temperature goes above or below this range, the natural functioning is disturbed that can lead to faulty circuit operations. An attacker can perform these attacks by heating or by cooling a target device to specific temperatures resulting in either bit sets or bit resets.

- **Optical attacks:** Electric circuits are sensitive to light because of the photoelectric effect. If we expose a circuit to intense white light for a brief period, the photons in light can cause the flow of electric current to flow can thereby switch off the transistors in the circuit. If the circuit is unshielded to the laser source, the ionization can also cause faults. Laser attacks can be used to achieve precise control on location wherein a single bit of memory can be set or reset. These types of fault attacks usually require the chip to be decapsulated to allow light or laser to reach the surface. The decapsulation is expensive and if not done correctly can damage the chip.

- **Hardware Trojans:** In this type of attack, an attacker has direct access to the circuit design. He can modify the design to install a fault circuitry which can bypass the security features of the critical functions of the design. This type of threat is especially coming into the limelight in recent research reports. Hardware Trojans are sometimes added by people when they produce an IP after which they sell it to the industries.

- **Electromagnetic pulses:** Previous works suggest that EMFI faults occur due to the violation of targets timing constraints. They do not need any decapsulation since the electromagnetic field can penetrate the IC packaging. The other advantage with EMFI attacks is that they are localized, so a global detector is not enough to detect these attacks. These attacks are highly configurable too as the fault injection timing, intensity and location can be adjusted using various EM injection parameters. These attacks are the primary focus of this study, and we discuss them in much detail in the section below.

2. **EMFI attacks** Electromagnetic faults attacks are one area of these attacks where the coupling of electromagnetic waves with the victim chip can change its correct behavior by creating ground bounces or voltage drops. There has been exciting research done mainly in the recent times to try and explain the nature of these faults and ways to counteract it. Electromagnetic injection disturbs the target device through its magnetic flux which can lead to computational faults. The electromagnetic probe is used to create the short high-intensity voltage pulses. This pulse generates a current I in the probe coil. This current produces a magnetic field B around the coil and the magnetic flux (ϕ) is basically the part of the magnetic field that passes through the coil surface. If we consider the angle between the coil surface and magnetic lines of field B to be θ, the magnetic flux is given by the Equation (1.1).

$$\phi = BA\cos\theta \tag{1.1}$$

where B is the magnetic field, A is the surface area that the ϕ is passing through and θ is the angle between the normal of the surface A and the magnetic field lines of B. The basic law of electromagnetism called the Faraday's law of induction states that whenever a magnetic field interacts with an electric circuit, it generates an electromotive force (ϵ) given by Equation (1.2).

$$\epsilon = -N\frac{d\phi}{dt} \tag{1.2}$$

where N is the total number of loops of the coil and ϕ is the magnetic flux through a single loop. On combining Equations (1.1) and (1.2), we get the following Equation (1.3) for a single loop of coil.

$$\epsilon = -N\frac{d(BA\cos\theta)}{dt} \tag{1.3}$$

A sudden change in the coil current affects the magnetic field which causes the induced EMF to create spikes or drops in the voltage values inside the circuit. We can also observe that the induced emf is affected by the following factors:

- the angle θ between coil and target device: The higher value of magnetic flux will be obtained at an angle of 90 so that the cos $\theta = 1$.
- the distance between the coil and the target device: The smaller the distance between the coil and the target, the higher the magnetic flux through the target.

- the current through the coil and the voltage that generates it: a high voltage pulse produces more current resulting in higher magnetic flux.
- the size and material of the coil: The magnetic flux will be higher if the area of the coil is large and the material is of higher magnetic permeability [15].

1.2 Contributions

The following contributions were made during the course of this research

- Carried out a detailed evaluation of the standalone timing sensors and the in situ EM sensors detection schemes.
- Proposed a new detection scheme to detect EMFI attacks in FPGA based embedded systems - In situ Timing Sensor.
- Performed a comparative analysis of the advantages and disadvantages of the above detection schemes.
- Discussed the nature of EM attacks and its corresponding fault attack model.
- Co-authored the accepted the paper 'Inducing local timing fault through EM injection' at DAC, 2018

1.3 Thesis Organization

This chapter gives background about fault attacks, introduces the topic and puts forward the relevant research questions. The rest of the thesis is organized as follows:

Chapter 2 discusses similar research efforts on this topic.

Chapter 3 outlines the hardware and software experimental setups which were required for this work. These setups were especially useful in taking calibrated observations which were fundamental to this research.

Chapter 4 presents an evaluation of the conventional detection approaches namely standalone timing sensors and in situ dual complementary flip-flop EM sensors. These methods are discussed and analyzed for their efficiency as a countermeasure against EMFI attacks on FPGAs.

Chapter 5 proposes an in situ timing sensor-based detection method as an EM Injection countermeasure.

Chapter 6 presents our understandings about the nature of EM attacks and summarizes the thesis.

Chapter 2

Related Work

Previous efforts in this area have shown that electromagnetic injections can induce timing constraint violations in the integrated circuits and these effects are local and around the injection probe. In an effort to understand EM attacks, they have proposed some EM attack fault models.

2.1 EMFI Fault Models

The task of defining a precise fault model for the faults induced by electromagnetic pulses forms a sizable amount of past research in this area. These models help the researchers understand the nature and type of attacks that can be performed by an attacker using EMFI techniques. This understanding subsequently helps in creating better countermeasures for such attacks. Until now, the researchers have put forward different fault models based on

several ideas in their previous works. The principal ideas for EMFI fault models that come up frequently in the literature can be categorized as follows.

- **Timing Fault Model:** In the work by Zussa et al. [19], they have discussed attacks that involved timing constraint violations. Following this work, he also presented a study on EMFI countermeasure based on the timing fault model in [18]. The research by Dehbaoui et al. on the EMFI fault model on a 32-bit microcontroller [11], they reaffirm that EMFI faults are generated due to setup time constraint violations making them timing faults. Also, the recent work on EM injection by Marjan G. et al. [9] also stems from the idea of inducing timing faults in an ASIC system.

- **Bit Set/Reset Fault Model:** Ordas el al., in his 2014 work on the evidence of a larger EM-induced fault model [13], proposed that in addition to timing faults, EMFI can also produce bit-set and bit-reset faults. They stopped the target's clock and performed EMFI injection to generate such errors. This work also suggested the idea of EM injection fault model can have both timing and bit-set/bit-reset fault models together.

- **Sampling Fault Model:** In contradiction to the previous two models, Ordas et al. in his 2015 paper [12] proposed that EM pulse injections correspond to a different fault attack model which is neither a bit-set/bit-reset fault model nor a timing fault model. They called their model the sampling fault model. They mention that the sampling faults result from the incorrect sampling of the D flip-flops.

All the above fault models agree on the locality of EMFI attacks, but their ideas do not converge concerning the exact type of these attacks. Through our experiments, we try to gain a better perspective on the kind of faults in an EMFI fault model. In the next section, we will discuss the detectors that were proposed previously to counter the EMFI attacks.

2.2 EMFI Detectors

2.2.1 Delay-based Glitch Detector

In 2014, Zussa et al. [18] published a study regarding their investigation on the efficiency of a traditional glitch detector for detecting electromagnetic fault injection. They implemented their delay-based countermeasure as a hard macro to find out its spatial limitation and found out that a single delay-based glitch detector is indeed not sufficient to detect all EM glitches. There is some area which is not triggering any alarms even after placing five detectors. They have no claims regarding the number of detectors or their optimal placement that would lead to a complete EMFI detection. Also, the delay in these sensors needs to be tuned since they are prone to process variations.

2.2.2 Fully-Digital EM Pulse Detector

To avoid the difficulties with the tuning of delay according to the IC timings in the delay based glitch detector, El-Baze et al. [5] introduced an EM detector which is entirely digital. This detector takes a cue from the sampling fault model and therefore mainly features DFFs which are most sensitive to variation due to EM. This detector comprises of a total of 5 DFFs, six inverters, two XOR gates and one AND2 gate. In their experiment, they have mapped 37 hard macros of these EMP detectors geometrically on an FPGA. Their test experiments show a high number of false positives and even an increase in the instrumented design by 13%. They have mentioned their detection rate to be around 85%, but there is no mention regarding the precision or true negative rate. The optimal placement of the detectors is also an open question.

2.2.3 PLL-based Detector

In 2016, Miura et al. [10] proposed a PLL-based sensor circuit to detect EMFI. This sensor comprises of a single PLL and one Ring Oscillator. They take advantage of the fact that it takes several cycles for the PLL to lock the clock state. When EMI affects the internal clock which unlocks the PLL creating disturbances, it can be detected. They have compared their sensor to the delay-based glitch detector where they put forward the point that their sensor is much more immune to process variations. They have assumed that all chips will have a PLL circuit which is not the case especially in low-power FPGAs. Also, if the resources are

low, it is not possible to implement this countermeasure on an ASIC. Their design requires the routing of the RO to be long enough to envelop the target logic for good detection. They have a detection rate of 100% but they have no mention regarding the precision of detecting these faults. The high-frequency RO circuit and the high false alarm rate can lead to a very high power consumption.

2.2.4 Hogge Phase Detector

Breier et al. [3] proposed a digital sensor circuit to detect the EMFI-induced phase shift using a watchdog Ring Oscillator in 2017. The sensor consists of 3 functional modules namely a Ring Oscillator to sense the transient frequency variations induced by EMFI, a DFF modulator with the prolonged routing to derive a synchronized oscillation signal Data from CK and a phase detector to check for any change in phase shift between Data and CK. The prolonged routing introduces a delay factor on the signal line of data such that any disturbance on Data or CK will cause a phase shift change which will be detected. They have compared their countermeasure against the PLL-based EMFI detector. In order to latch the alarm, their sensor used 3 flip-flops and two LUTs. The sensitivity factor will also not be immune to process variations due to existence of delay element. Their detection rate is 93.15% but their precision or false positive rate is only 9.45%.

2.2.5 Dual complementary flip-flop Detector

This paper [7] proposes a low-cost, fully digital and cycle-accurate mechanism. The proposed sensor circuit consists of an existing main flip-flop and an additional shadow flip-flop that capture complementary values at every clock cycle. If there is an EMFI attack, either of the flip-flops gets affected based on the induced current polarity. This effect can be detected by applying an XNOR operation between the two flip-flops. They have implemented this detection scheme by an example of AES on a Cyclone IV FPGA. The detector can be integrated with the target circuit and take up one extra flip-flop and 1 combinational primitive for every existing flip-flop. This introduces some hardware overhead in the design but the false positive rate is extremely low which in turn reduces its power consumption.

Chapter 3

Experimental Setup and Methodology

This chapter provides an overview of the experimental setup that induces EM faults, and about the methodology that is used as the Device Under Test (DUT) for the experiments in this thesis.

3.1 The EMFI Setup

The EMFI setup is comprised of an EMFI transient Probe, a glitch controller (called VC Glitcher), an XYZ Motorized table, an oscilloscope and a computer running the Inspector software. This setup is then used to induce EM injections on a target embedded processor. We can divide the experimental setup into two parts namely the hardware configuration and the software configuration as discussed below.

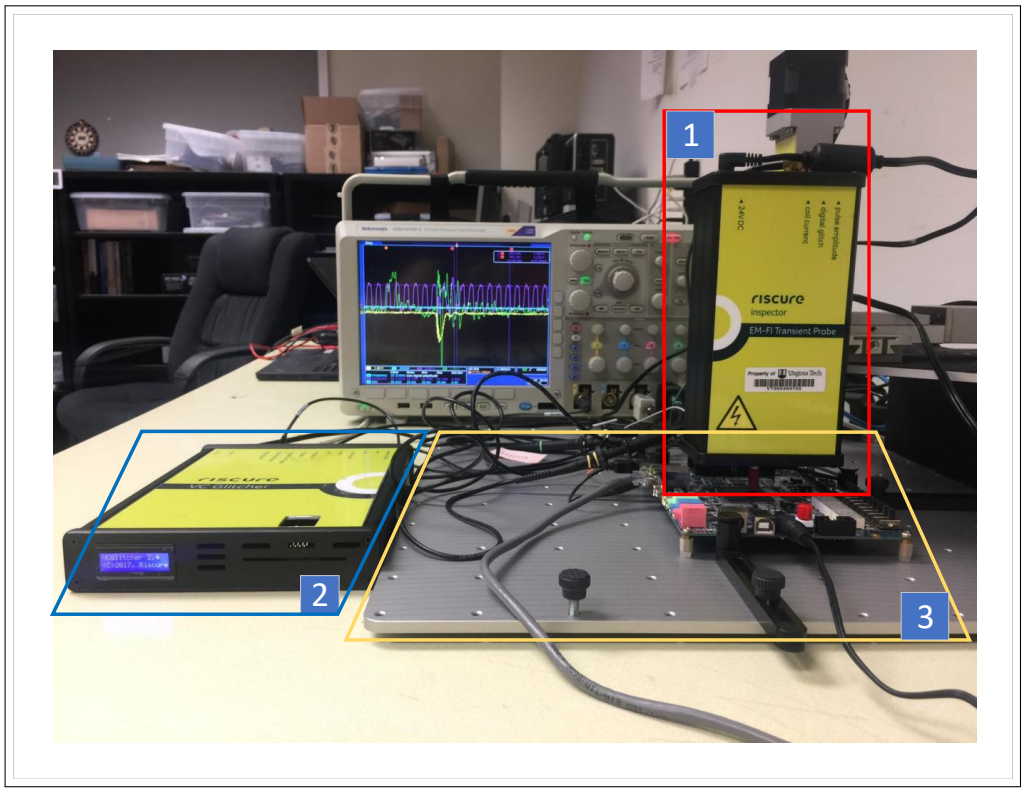


Figure 3.1: 1. EMFI Transient Probe 2. VC Glitcher and 3. XYZ Motorized table

3.1.1 Hardware Setup

We use the Fault Injection and Side Channel Analysis platform provided by Riscure Corporation to build our EMFI hardware setup. This consists of the Riscure EMFI transient probe, Riscure VC Glitcher, and the Riscure XYZ motorized table as shown in the Figure 3.1. This platform along with the Riscure Inspector software controls the entire process and coordinates communication between all the devices. This section describes the different components of the EMFI setup.

- **EMFI Transient Probe:** The EMFI probe generates high power, fast and predictable

electromagnetic pulses which are used for EM fault injection in embedded processors. The resulting current induced in the probe coil has two characteristics:

1. the pulse amplitude measured in volts.
2. the pulse duration measured in nanoseconds.

Riscure provides different probe tips which are of different polarities and diameters to generate EM fields over the surface of the chip. The diameter of the EM metal coil is 1.5 mm for two probes and 4mm for the other two probe tips. The EM pulse generated by the 4 mm coil probe tip is much stronger than the 1.5 mm coil probe tip. Each diameter tip comes in both positive and negative polarity. The start time, duration and magnitude of this pulse depend on the attack command it receives from the VC Glitcher. We have used the 4 mm probe tip with positive polarity in our experiments.

- **VC Glitcher:** The VC Glitcher [14] is a Riscure proprietary device for fault injection using voltage and clock glitches. It is used to send two signals, namely, Digital Glitch and Pulse Amplitude to the EMFI probe. Using the Digital Glitch signal, VC Glitcher conveys the instant and the duration of the EM pulse to the EMFI probe. The amount of current that should flow in the EM coil inside the probe is determined by the Pulse Amplitude signal that is sent by VC Glitcher. This can be used to control the intensity of the EM pulse. The VC Glitcher is connected to the computer, the EMFI probe, and the target device. It receives the attack parameters from the Inspector software, waits for the trigger signal from the target device and then sends the attack signals to the

EMFI probe.

- **XYZ motorized table:** The target embedded processor (DUT) is fixed onto a XY-axis motorized table. The EMFI probe is attached to the table such that it sits precisely above the DUT with its EM coil very near to the DUT surface. The probe can be moved and calibrated in all the three dimensions because of this motorized table. The motorized table is then used to position the probe at a fixed point or automatically scan the chip using the Inspector software. Here, scanning the chip refers to injecting EM pulses within an area along a grid specified in the inspector software.

3.1.2 Software Setup

The Inspector software from Riscure is used for automation of the fault injection tests, observation of the results for easy analysis and refinement of the test cases. This software runs on the PC that controls the XYZ motorized table. We can use it to specify and configure the attack parameters, transmit them to the VC Glitcher and communicate with the target embedded processor. We discuss these three roles of the Inspector below.

1. **Configuring the Attack Parameters**

   The software configures the attack parameters that can be divided into the below two groups.

   (a) **Location parameters:** Inspector can control the position of the EMFI probe on the XYZ table using the window shown in the Figure 3.2. We can either set

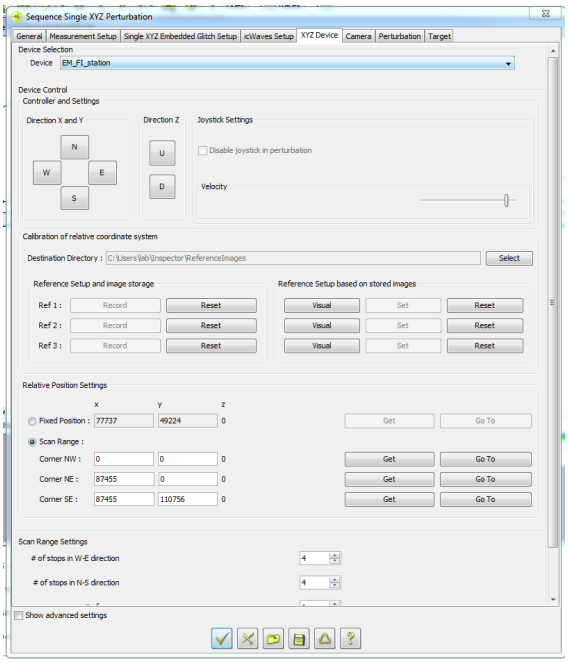


Figure 3.2: Inspector Software Location Parameters

the point of the Fault injection to be a fixed point or scan the whole or part of the chip using three reference coordinates.

(b) **Perturbation parameters:** These can be defined as physical quantities that influences the behavior of the EM injection which can be varied to perform a successful glitch. They are illustrated in the Figure 3.3. These are set using the Inspector window shown in the Figure 3.4 and include the following:

- **Glitch offset:** We can define glitch offset as the time between the reception of the trigger and the insertion of the glitch. Glitch offset can be specified as any value greater than 4 ns. Increasing the glitch offset can be used to insert glitches at different phases of the clock.

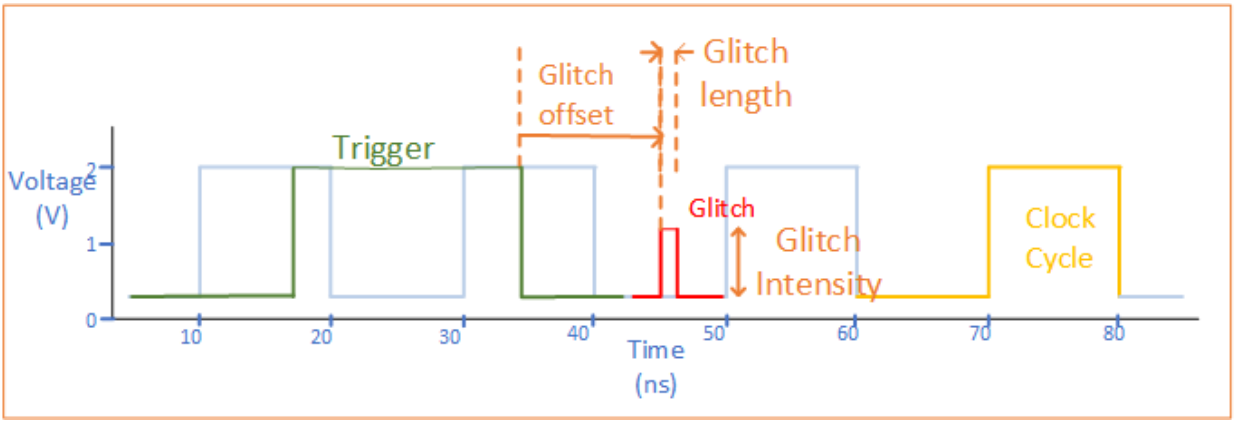


Figure 3.3: Glitch Parameters Illustration

- **Glitch source power:** This is also referred to as glitch intensity and specifies the relative strength of the EM pulse. This parameter is determined as a percentage of the highest EMFI voltage which is 450 V, eg: a glitch source power of 50% is 225V.
- **Glitch length:** This parameter defines the duration of the glitch. It can be set as a multiple of 2 ns. This duration translates to the amount of time for which the current is passed through the EM coil. We use the glitch length of 50 ns as recommended by Riscure in all our experiments.
- **Glitch repetition:** This is used to specify the number of pulses the EMFI probe should generate for each attack.

The perturbation parameters like glitch repetition, glitch source power, glitch offset and glitch length can be "fixed", "random" or in a "range".

2. **Transmitting the glitch parameters to the VC Glitcher** In addition to config-

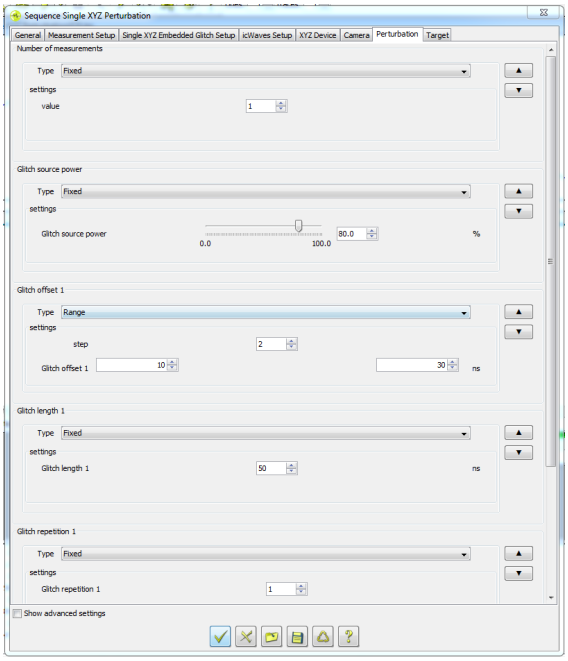


Figure 3.4: Inspector Software Perturbation Parameters

uring the parameters, the Inspector software send those parameters to the VC Glitcher to arm it. We can then select a custom program to load to the VC Glitcher. This custom program can be any Java module that is derived from a basic glitcher program.

3. **Communicating with the target** The Inspector interfaces with the DUT using serial interface. The default read/write protocol is the ReadAll protocol which reads all the data the device sends to Inspector. Instead of using predefined read/write protocols, a user can write his custom protocol. This custom protocol is should appropriately read the bytes sent by the DUT. It should be defined in a custom Java module program.

3.2 Attack Characterization

In this section, we characterize an attack depending on the relation between the faults and alarms and categorize them in various groups and well-known detection metrics.

3.2.1 Attack Analysis

What is an attack? Attacks may or may not produce faults. Consider an EM injection with a very low intensity that has no chance of causing a fault. Is this still considered an attack? Should the detector report such cases? We believe the answer to these questions is 'no'.

An attack becomes a threat when it affects the circuit to create faults. We call such attacks as viable attacks. Viable attacks are a subset of all attacks. We consider an attack 'viable' if it has its attack parameters configured such that it will induce a fault. Our goal is to detect only such attacks. If every attack including the benign one is detected, it will result in massive performance reduction. This response can be detrimental to many critical and real-time applications.

3.2.2 Detection Metrics

We use well-known data analysis techniques [8] to evaluate our detection schemes. For this purpose, we categorize each attack in one of the four groups as shown in Table 3.1. A

		Alarm	
		Yes	No
Fault	Yes	True Positive (TP)	False Negative (FN)
	No	False Positive (FP)	True Negative (TN)

Figure 3.5: Confusion Matrix

detection is 'true' if the alarm matches the existence of the fault. Otherwise, it is a 'false' detection. On the other hand, the existence of the alarm means the detection is 'positive'. Otherwise, it is 'negative'. The acronyms TP, FP, TN and FN in the Figure 3.1 represent the total number of True Positives, False Positives, True Negatives and False Negatives respectively.

In addition to the above four metrics, we have also used the following classification metrics for a better comparison of countermeasures against the attacks.

- **Detection Rate:** The percentage of faults that were detected is the detection rate. This number should be high for any efficient countermeasure. The detection rate should ideally be 100%.

$$Detection\ Rate = \frac{TP}{TP\ +\ FN} * 100$$

- **Precision:** In addition to having a high detection rate, it is essential that the detection be precise. A detection technique is not precise if raises an alarm in the absence of the

fault. The precision is defined below and should ideally be 100%.

$$Precision = \frac{TP}{TP\ +\ FP} * 100$$

- **Specificity:** The specificity (also known as the true negative rate) is a metric that informs about the percentage of negatives that were rightly identified as being no faults. A high value of true negatives will increase this metric. The specificity should ideally be 100%.

$$Specificity = \frac{TN}{TN\ +\ FP} * 100$$

- **Accuracy:** The Accuracy is the percentage of the true results out of the total number of observed cases. So, a high accuracy implies high precision as well as high trueness. This number should ideally be 100%.

$$Accuracy = \frac{TP\ +\ TN}{TP\ +\ TN +\ FP +\ FN} * 100$$

3.3 Target Device Setup

The overall setup and procedure of Electromagnetic injection followed in our experiments is illustrated in the block diagram shown in the Figure 3.6. We used two different types of FPGA boards shown in the Figure 3.7 to carry out our experiments. In our experiments, we implemented two existing EMFI detectors namely standalone Timing Sensors and in situ EM Sensors and analyzed them. For the purpose of our experiments, we attack the cryptographic algorithm called AES [4] which is a widely accepted symmetric-key encryption

algorithm. Here, we discuss the setup and methodology used to perform experiments on the two development boards.

3.3.1 DE1-SoC Implementation

The DE1-SoC development kit by Terasic has the Altera Cyclone V SoC which has an FPGA with a dual-core ARM Cortex A9. Serial communication is used as an interface between DE1-SoC and the PC. The board also provides support to connect the Cyclone V to Ethernet. We discuss the hardware and software implementations on this FPGA in the following sections.

1. **Hardware for Standalone Timing Sensor**

   The hardware logic used on the FPGA on DE1-SoC for this detector comprises of three different parts described below.

   - **AES logic:** We used the AES co-processor used in the FAME ASIC [17] as the starting point of the hardware logic that was implemented on the FPGA. The logic design software by Altera namely Quartus 17.1 was used as the design tool. The communication between the Inspector and the FPGA used the on chip UART module. Therefore, it was essential to keep the UART logic as well as a few other module like the reset controller module and Parallel Input Output (PIO) modules separate from the AES logic. This is because we intend to inject EM faults only on the AES module. Also, to see the effect of proximity of timing sensors to the

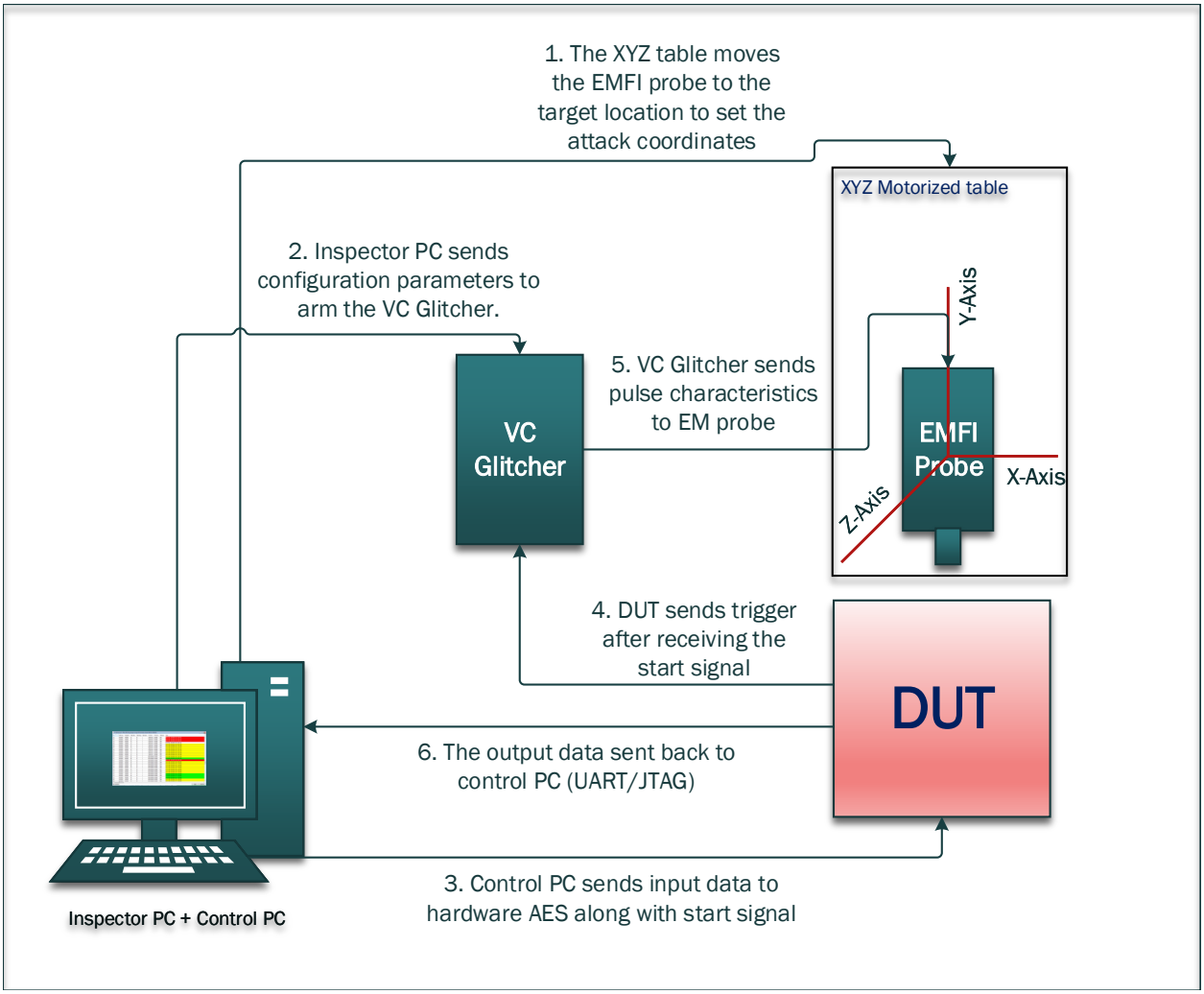


Figure 3.6: Block Diagram of EMFI Setup

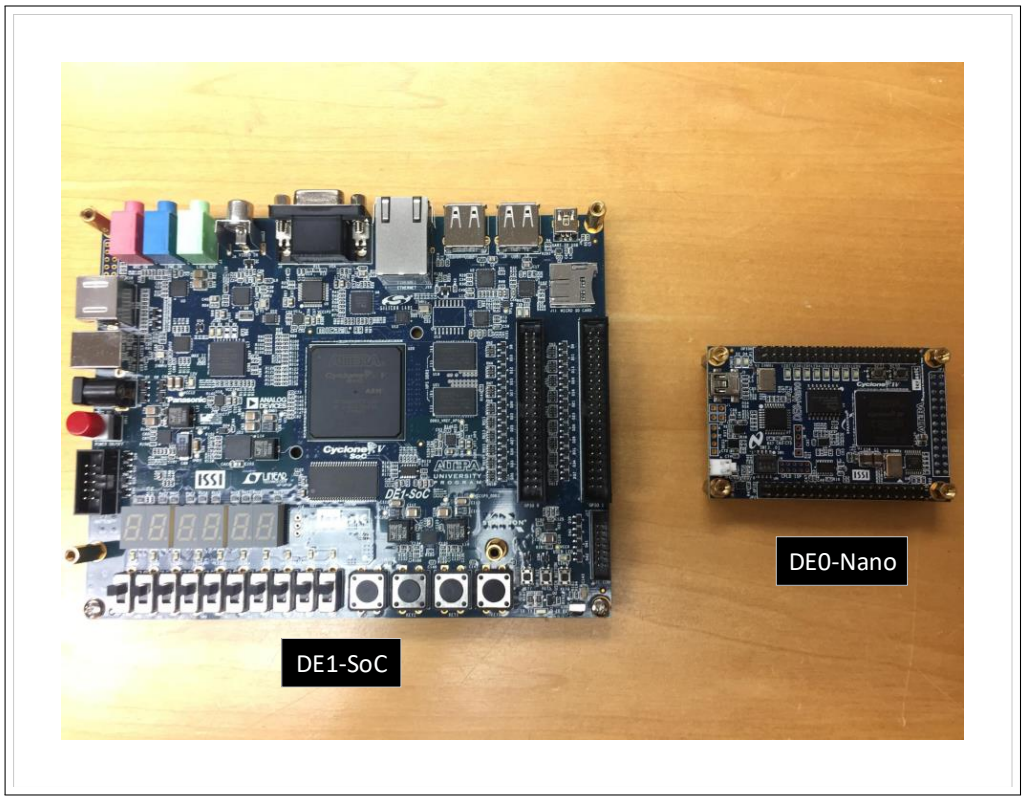


Figure 3.7: DE1SoC and DE0Nano FPGAs

AES logic, it was essential to make it a single macro-like entity. In Quartus, we can achieve this functionality by using a technique called logic locking. To design with this technique, we have to create a contiguous rectangular logic-locked region and assign logic to it such that the logic is placed within the locked boundaries. One can modify a logic-locked region's size, width, height, state and origin using either the LogicLock Region Properties window as shown in Figure 3.8 or directly modify them using the chip planner.

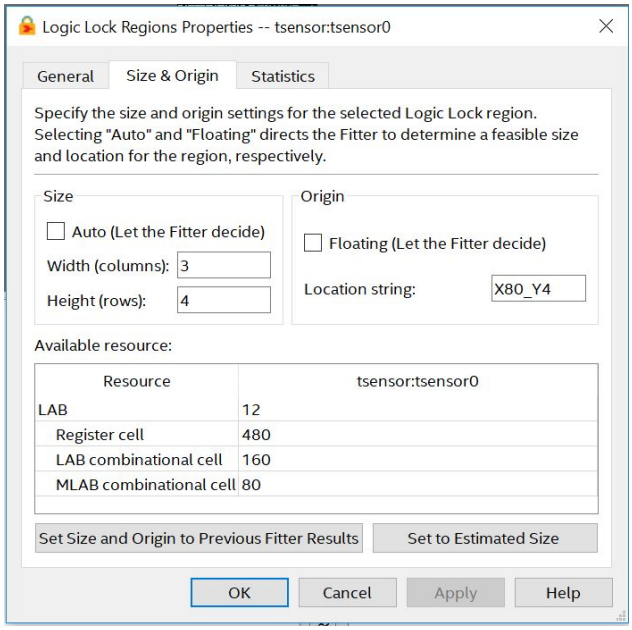


Figure 3.8: LogicLock Region Properties Window

- **Timing Sensor logic:** We used a timing sensor design [6] similar to the Figure 4.1 and implemented it on the FPGA on the De1-SoC development board. This timing sensor compares the difference between a signal and the delayed version of the signal to determine whether there was an attack. In a successful clock

glitch attack, there will be setup time violations which can cause the flip-flop with the delayed signal to latch a wrong value hence giving an alarm. This delay is usually generated using an internal delay chain implemented using NOT gates. The number of NOT gates in these chains decide the sensitivity of the sensor. However, on implementing this detection sensor on the FPGA using Quartus, it was observed that the design software would optimize away the NOT gates during synthesis assuming them to be redundant. Therefore, we chose to use Lcells in order to implement the delay.

The Lcell [1] is a low-level primitive buffer which can be used to allocate one logic cell in the design. Also, when one instantiates an Lcell buffer, Quartus II software does not remove it during the synthesis and preserves its alignment. Altera mentions that we can use Lcell primitives to create an intentional delay. However, the delay of these elements varies with temperature, power supply voltage, and the device fabrication process. We achieved the required delay in the timing sensors by using Lcell chains. The Figure 3.9 shows how Lcells were integrated into the RTL netlist of our design. It was necessary to find the ideal number of Lcells which gives the right sensitivity for the FPGA to create an alarm for any clock glitch. The DE1-SoC runs on 50 MHz and so we experimented by varying the Lcells from 40 to 100 and observe their delay on an oscilloscope. At 80 Lcells the delay was high enough to always give an alarm even without a clock glitch. The sweet spot just right to see an alarm even on any clock glitch was 70 Lcells. So,

we have used 70 Lcells in all our experiments as the optimum number to create the ideal delay.

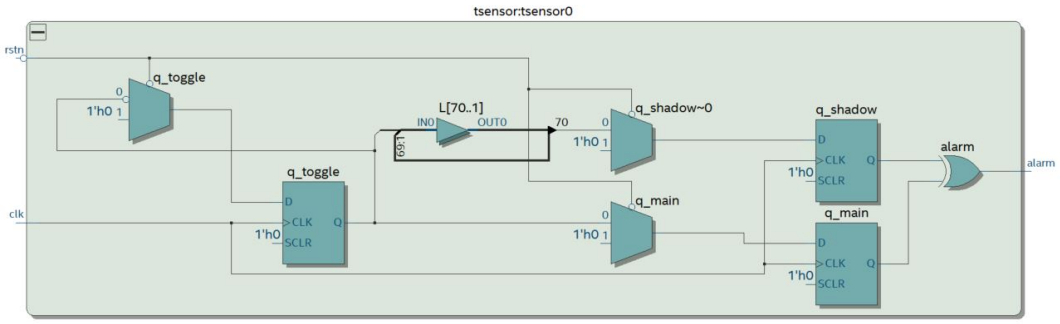


Figure 3.9: Timing Sensor schematic with Lcells

- **Wrapper logic:** Our design instantiates four timing sensors inside a timing macro (shown in the Figure 3.10) and the AES logic together into a single wrapper. Input data is written into the wrapper, then the wrapper asserts a start trigger. After the AES operation is complete, the output data as well as the value of the alarm can be read back from the wrapper. These read/write operations on the addresses were performed using Avalon Memory-Mapped interfaces. This wrapper also contained the latching logic to latch the alarm value at every clock cycle and reset it with a software reset. The final implementation of the design on the chip planner is shown in Figure 3.11.

2. **Hardware for in situ EM Sensor**

The baseline design used for our experiments is taken from the FAMEv2 fortified AES co-processor [16] and it was modified to cater to our requirements. The hardware logic for in-situ EM sensor used on the FPGA on DE1-SoC for this detector comprises of

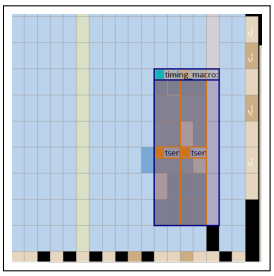


Figure 3.10: DE1-SoC Timing Sensor macro

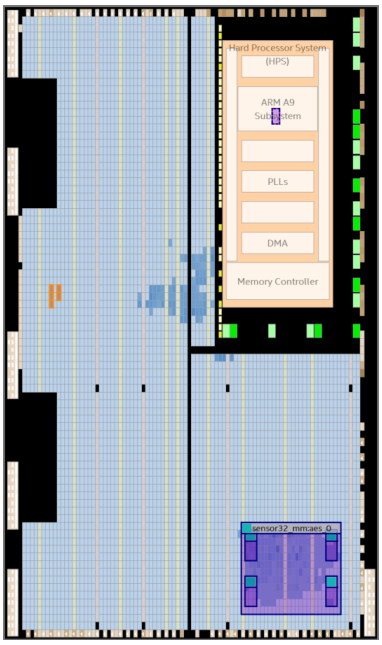


Figure 3.11: DE1-SoC Timing Sensor Chip Planner

three different parts described below.

- **Hardware**

  AES and EM sensor logic: The alarms from all the in situ EM sensors are ORed together to give a one-bit alarm output. The main flip-flops along with their complementary partners are placed in a 'flop' module which is logic-locked separately. We do this logic locking of the two flops in a macro as [7] has mentioned

that the MFF and SFF should be placed extremely close to each other for the sensor to detect bit-set/bit-reset faults. Figure 3.12 shows the DE1-SoC chip planner having the AES implementation with the logic locking of the in situ EM flops. We created a separate module 'flop' with just the Main Flip-Flop (MFF) and Shadow Flip-Flop (SFF) inside it. These flip-flops can be logic-locked together to fulfill the proximity requirement of the dual-complementary Flip-Flops. These physical constraints are necessary as the sensor assumes that both the flip-flops should be affected in the same manner by the EM fault injection.

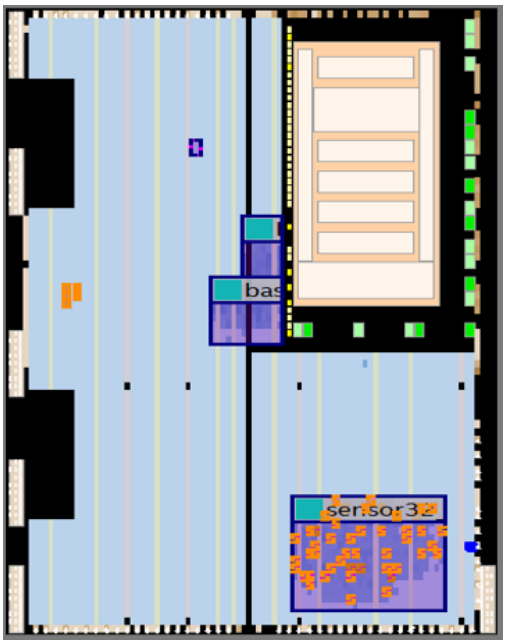


Figure 3.12: DE1-SoC EM Sensor Chip Planner

Wrapper logic: The wrapper logic only instantiates the AES logic inside it and handles the read and write of its inputs and outputs respectively. The latching of the alarm is taken care inside the design. We do not need to latch it separately in

the wrapper. The output alarm signal is exported to the one of the GPIO pins of the FPGA. The read and write are done using memory mapped addressing similar to the experiments done with AES and timing sensors. There is a start trigger which starts the AES operation, and this signal is exported out on a GPIO pin. This pin is used as the trigger to signal the EMFI setup to inject an EM injection on the surface. The outputs are read out from memory mapped addresses after reading the done signal signifying the completion of the AES.

3. **Software:**

This section describes the software that runs on the dual core ARM on the DE1-SoC. This software is used to interface the hardware co-processors on the FPGA which have the standalone timing sensors and the in situ EM sensors. The synchronization between the ARM on the DE1-SoC and the Inspector is done using a system call which is a request to the operating system to perform some activity. The system call method does a read() of a specified number of bytes from a file descriptor into a buffer. We use this system call to read a '1' that is sent from the Inspector using serial communication. After the software receives this set command, it loads the input data into the AES and triggers to start the AES operation. The program then waits for a done signal which indicates AES has completed. The signals used in these steps can be observed on the oscilloscope as shown in Figure 3.13. The green signal is the trigger, the blue represents the done, the purple one shows the clock and the yellow signal is the electromagnetic coil current.

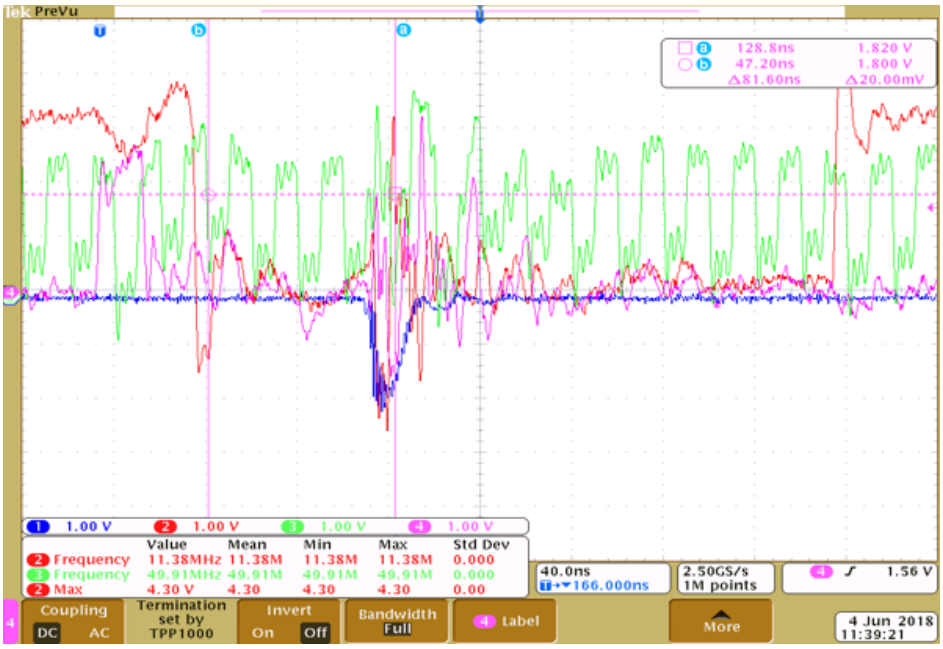


Figure 3.13: AES logic signals on Oscilloscope

Following this, the software reads the output data and alarm values and compares them to the expected result to determine whether there was a fault. The software c program used is attached in the Appendix A for reference. After every injection, we also send both the alarm and fault states back to the Inspector as seen in the Figure 3.14.

To send command and receive information, we build a custom user Java module that Inspector uses as a target for communication and synchronization. These values can be logged and further analyzed in the Inspector tool using perturbation logs as shown in the Figure 3.15. Using the Java module, it is also possible to give different verdiction or color to the displayed outputs depending on certain conditions. For example, the

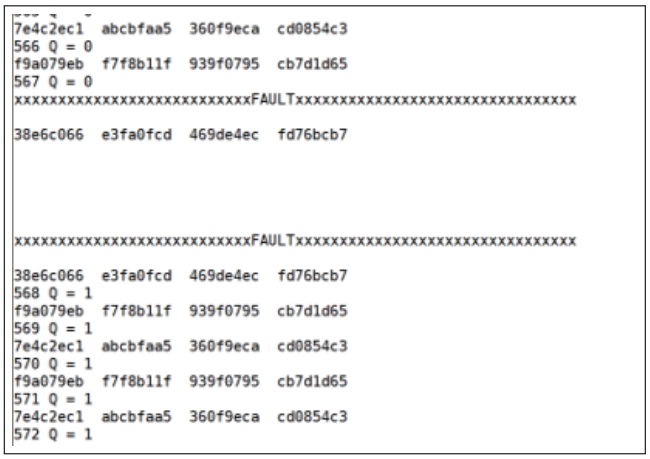


Figure 3.14: Seeing Alarms and Faults on MobaXterm

following verdiction scheme shown in the Figure 3.16 was used to display the results and count the number of scenarios for each of the four cases using counters. The code for the user-defined Java module can be found in the Appendix C.

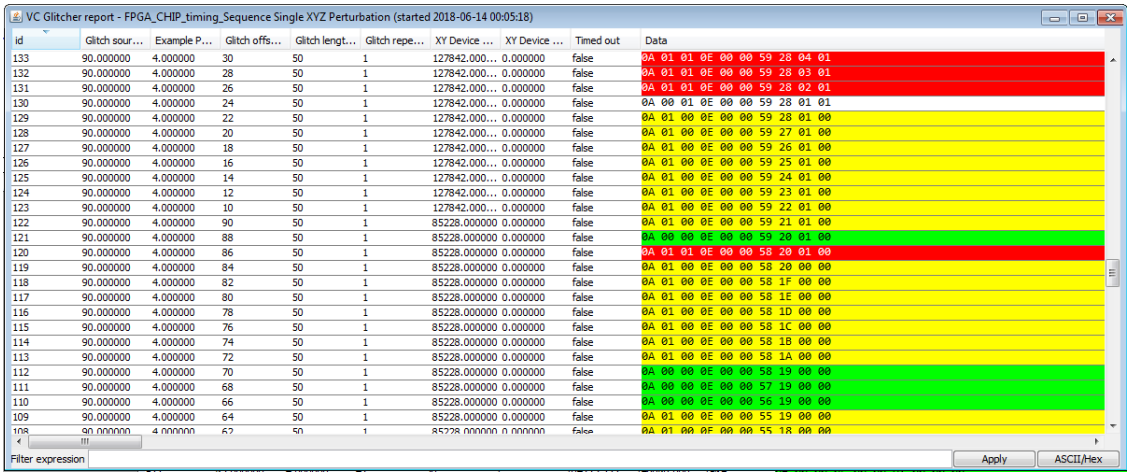


Figure 3.15: Perturbation Log

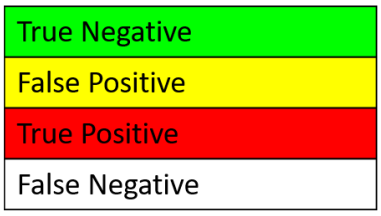


Figure 3.16: Verdiction

3.3.2 DE0-Nano Implementation

The DE0-Nano development kit by Terasic has the Altera Cyclone IV FPGA. The DE0-Nano has no interface for a direct communication link with the PC as the Inspector software can only access COM ports attached to the PC. The Figure 3.17 Shows a block diagram of this communication interface which comprises of a memory-mapped bus master (JTAG to MM Master), a memory-mapped slave (hardware AES) and an Altera Avalon bus. The communication between the computer and the DE0-Nano is performed using TCL scripts through the system-console application. For DE0-Nano, the majority of the hardware implementation is similar to DE1-SoC while the communication and software are significantly different. We discuss this in the following sections.

1. **Hardware for Standalone Timing Sensor** To reinforce the observations we obtained with the EMFI experiments on DE1-SoC FPGA, we used a different FPGA platform namely the DE0-Nano development board. Since the delay generated using the Lcells is process dependent, it would change on using a different FPGA. The AES, timer sensor and wrapper logic are the same as the one used for DE1-SoC. The placement done on Chip Planner is different and is shown in Figure 3.18.

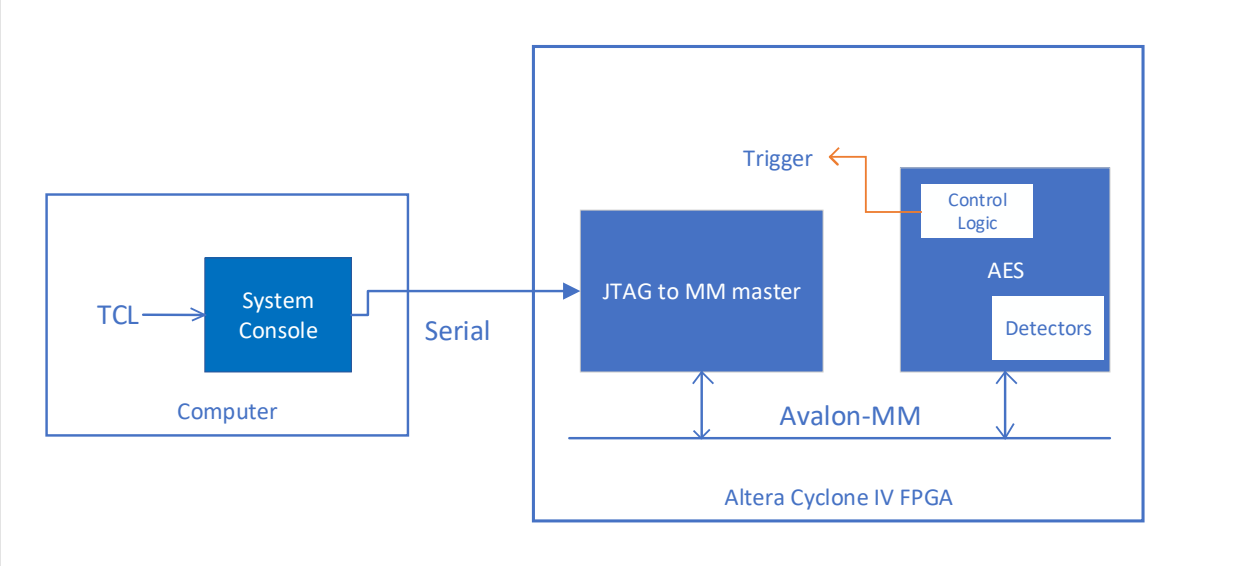


Figure 3.17: Communication Interface with DE0-Nano

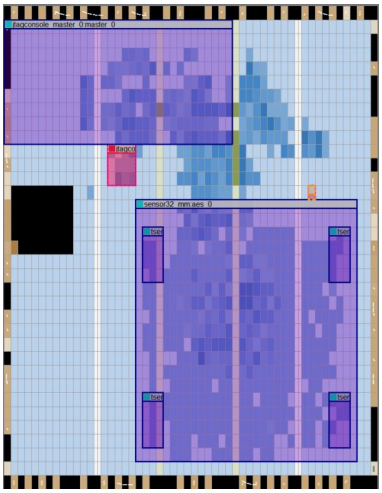


Figure 3.18: DE0-nano Timing Sensor Chip Planner

2. **Hardware for In situ EM Sensor** The same hardware implementation as the one used for DE1-SoC is used for DE0-Nano as well with the only difference being that the Quartus version used is 16.0 instead of 17.1 for compatibility. The chip planner after implementing the AES with in situ EM Sensor can be seen in the Figure 3.19.

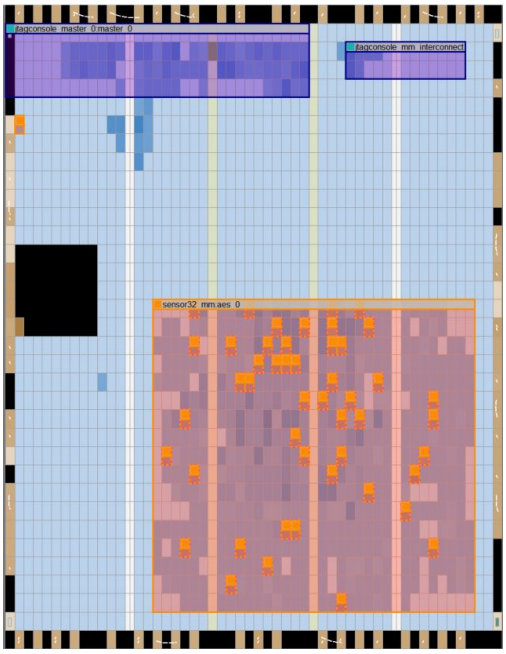


Figure 3.19: DE0-nano EM Sensor Chip Planner

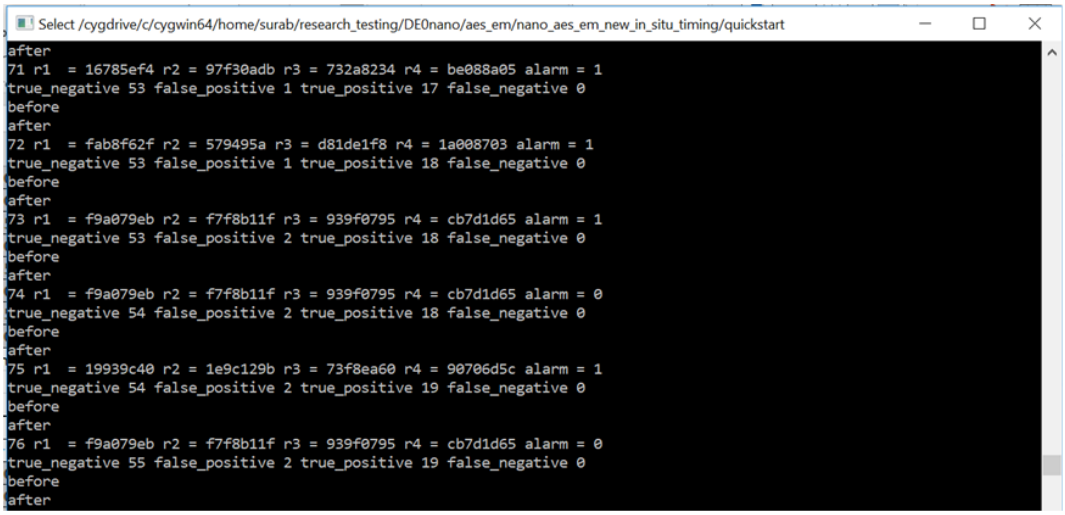


Figure 3.20: DE0-Nano System Console Results

3. **Software** Since there is no UART module present in this FPGA, communication with Riscure's Inspector software was not possible using the mechanism which is used in DE1SoC. Instead, the TCL script along with the system-console application is used as the software program. The software has access to the JTAG to MM Master through

USB. This block is connected to the Avalon-MM bus on which the AES coprocessor is attached. Through this interface, we can access the internal registers of the coprocessor. This method is used to write the input data, provide a start trigger and read the output data as well as the value of the alarm. The results for the detection metrics can be seen using the system console application shown in the figure 3.20. The TCL script that was used is attached in the Appendix B for reference.

Chapter 4

Experimental Evaluation of Existing Detection Mechanisms

Our study on electromagnetic injections confirmed that EMFI is local and can create ground bounces or voltage drops on a logical wire which falls in its EM attack range. We will now briefly discuss the two main existing detection mechanisms for EMFI attacks and perform their comparative analysis in the following sections.

4.1 Standalone Timing Sensor

We use the timing sensor proposed by C. Deshpande, B. Yuce et al. in [6] which is illustrated in the Figure 4.1. This 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. All the flip

flops that are shown for this sensor share the same clock. We study the effectiveness of the standalone timing sensor while varying the sensors' location as well as the attack location, intensity and offset.

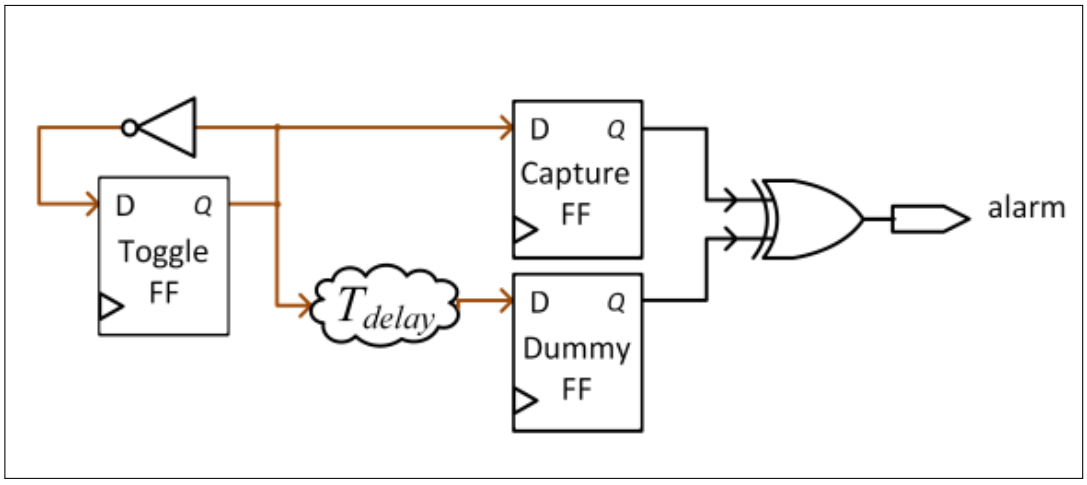


Figure 4.1: Timing sensor structure [6]

- **Effect of sensors' Location:** To evaluate the effect of sensors' location, we carry out two experiments. In the first one, we implement one standalone timing sensor and simply change its location while in the second experiment we use a carpet of sensors. In these experiments, we are only observing the sensors and there is no actual circuit (such as AES) on the board. This means there are no faults to be detected. The sensors are simply trying to detect the attacks. The Figure 4.2 shows the number of alarms at a point in the DE1-SoC chip. The top row shows the different chip planners with their timing sensors locations and the bottom row shows the number of alarm raised when scanning the chip in a 10x10 grid. The darker the shade of a block, the higher the number of alarms at that point and vice-versa. We observed that the location of the sensor has a significant effect on how often and at what locations it can detect the

attack. We also observe that regardless of its location, none of the sensors can detect all the attacks. Similar results were observe on the DE0-Nano board.

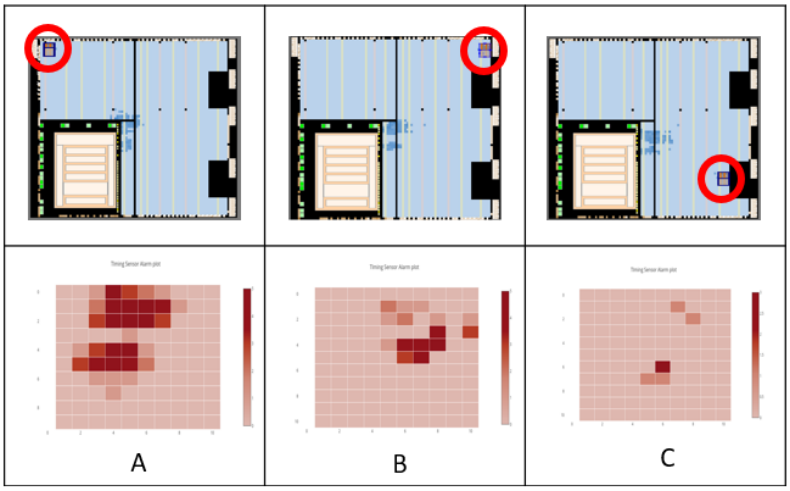


Figure 4.2: One Timing Sensor at A. Top Left B. Top Right and C. Bottom Right

In the second location-related experiment, a carpet of timing sensors was placed on the DE1-SoC chip as shown in Figure 4.3. We saw that timing sensor 3 had a vast area of detection scope. Also, other timing sensors in the left top quadrant (1, 2, 4, 5, 6, 7, 8, 9) were able to detect EMFI and give alarms for a good range of area too. However, for timing sensors on the right half of the chip, it was tough to find the exact location where the sensor would raise an alarm. These detection areas were very specific and small compared to the top left quadrant sensors. Since we do not know the exact clock distribution in the FPGA, with these experiments, one can estimate that the clock might originate closer to top left. Similar to the previous experiment, this one confirms the importance of timing sensor location on its effectiveness. A timing sensor placed in the wrong location can be simply useless.

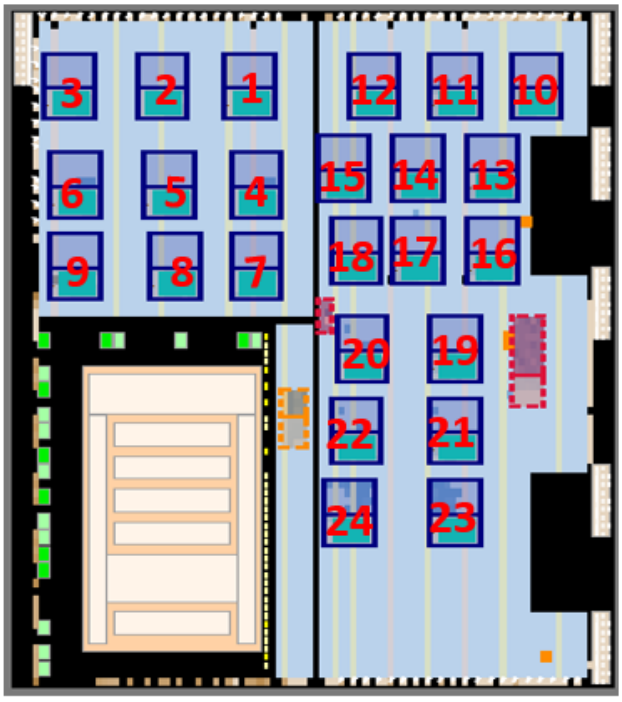


Figure 4.3: Timing Sensor Carpet

Intensity	True Negatives	False Positives	True Positives	False Negatives	Precision	Specificity	Detection Rate	Accuracy
70	260	115	3	6	2.5	69.3	33.3	68.5
75	235	114	13	22	10.2	67.3	37.1	64.6
80	187	125	48	24	27.7	59.9	66.7	61.2
85	160	128	66	30	34	55.6	68.8	58.9
90	132	121	99	32	45	52.2	75.6	60.2

Figure 4.4: DE1-SoC Standalone Timing Sensor Intensity Sweep

- **Effect of Attack Intensity:** To observe the effect of EM attack intensity on the behavior of the standalone timing sensor, we vary the intensity from 30% to 90% with steps of 10%. Unlike the previous location study, we have included a hardware AES circuit on the board for this experiment. The result of this experiment is summarized in the Figure 4.4. We categorized the total number of EM injections on a chip into the

detection and the characterization metrics as described in Section 3.2.2. The different colors are used to differentiate the various metrics from each other. Also, the color scales are darker for higher values in the cells and lighter in shade for lower values. For the DE0-Nano board, the intensity sweep is shown in the Figure 4.5. It can be seen that the results are in line with the observations from DE1-SoC intensity experiments. The inference from these tables will be explained in the next section.

Intensity	True Negatives	False Positives	True Positives	False Negatives	Precision	Specificity	Detection Rate	Accuracy
50	75	62	38	0	38	54.7	100	64.6
60	44	85	46	0	35.1	34.1	100	51.4
65	52	93	30	0	24.4	35.9	100	46.9
70	39	100	36	0	26.5	28.1	100	42.9
75	30	108	37	0	25.5	21.7	100	38.3
80	20	117	38	0	24.5	14.6	100	33.1
85	12	130	33	0	20.2	8.5	100	25.7
90	7	127	41	0	24.4	5.2	100	27.4

Figure 4.5: DE0-Nano Standalone Timing Sensor Intensity Sweep

- **Effect of Attack Offset:** The Figure 4.6 shows the results for the offset sweep. Using the Oscilloscope, the rising edge of the clock is observed at an offset value of 16 ns and the clock period is 20 ns. Faults were observed when performing EM injections just before or during the rising edge of the clock. Different rising edges give different outputs and on some rising edges, we see no faults. This is because at some specific locations, the change in the values of some of the flip-flops might not give rise to a fault. For DE0-Nano FPGA as well, the alarm is also seen before or during rising edge of the clock shown using the offset sweep in the Figure 4.7. The results from these

tables are analyzed in detail in the next section.

Alarm Map	Offset (ns)										
intensity	10	12	14	16	18	20	22	24	26	28	30
30	10	7	2	0	0	2	8	10	10	10	10
40	10	10	6	2	2	7	10	10	10	10	10
50	10	10	8	3	4	8	10	10	10	10	10
60	10	10	9	9	7	9	10	10	10	10	10
70	10	10	10	10	8	8	10	10	10	10	10
80	10	10	10	10	7	6	8	10	10	10	10
90	10	10	10	10	9	9	8	10	10	10	10

Fault Map	Offset (ns)										
intensity	10	12	14	16	18	20	22	24	26	28	30
30	0	0	0	0	0	0	0	0	0	0	0
40	0	0	0	0	0	0	0	0	0	0	0
50	0	0	0	0	0	0	0	0	0	0	0
60	7	1	0	0	0	0	0	0	8	10	8
70	3	4	0	0	0	0	0	3	6	5	6
80	4	6	9	3	0	0	2	7	10	10	8
90	10	10	3	0	0	0	2	10	10	10	10

Figure 4.6: DE1-SoC Timing Sensor Faults vs Alarms Map

Alarm Map	Offset (ns)										
Intensity	10	12	14	16	18	20	22	24	26	28	30
40	0	5	20	25	28	27	15	17	12	3	0
50	1	4	18	15	28	31	17	13	9	2	0
60	1	4	18	24	29	31	17	13	7	6	0
70	2	11	20	27	32	27	20	17	13	7	0
80	10	11	18	26	32	25	18	13	13	13	16
90	15	17	20	27	32	29	17	13	15	18	19

Fault Map	Offset (ns)										
Intensity	10	12	14	16	18	20	22	24	26	28	30
40	0	0	1	14	24	22	1	0	0	0	0
50	0	0	1	3	24	25	3	0	0	0	0
60	0	0	1	6	24	25	1	0	0	0	0
70	0	1	2	10	27	20	1	0	0	0	0
80	0	1	1	10	27	10	3	0	0	0	0
90	0	2	2	10	29	15	3	0	0	0	0

Figure 4.7: DE0-Nano Timing Sensor Faults vs Alarms Map

- **Effect of Delay:** The effect of varying delay values can be observed from the Figure 4.8. These delay values are varied by changing the number of Lcells where a higher number of Lcells signifies a higher delay. At 80 Lcells, we see alarm even without any EM injections. So, we perform the delay sweep uptil 78 Lcells. We observe that the false positives increase and the false negatives decrease as the delay value increases.

This resulted in the detection rate to rise with higher Lcells but with the increasing number of false positives, the precision and accuracy were negatively affected.

Lcells	True Negatives	False Positives	True Positives	False Negatives	Precision	Specificity	Detection Rate	Accuracy
25	537	81	11	25	95.6	86.9	30.6	83.8
35	539	69	13	34	93.9	88.7	27.7	84.3
45	524	91	7	20	92.1	85.2	25.9	82.7
55	541	78	6	29	90.8	87.4	17.1	83.6
65	315	271	25	43	82.9	53.8	36.8	52
75	63	495	85	12	45.7	11.3	87.6	22.6

Figure 4.8: DE1-SoC Standalone Timing Sensor Delay Sweep

4.2 Standalone Timing Sensor Advantages and Disadvantages

A delay-based timing sensor widely appears in the previous endeavors to build EMFI countermeasures. These efforts show that the timing sensor is considered as an established candidate for EMFI countermeasure. We discuss its advantages briefly below:

1. **Good detection rate:** Timing sensors show an above average detection rate of faults if they are placed close to the logic that is attacked. We can get a 100% detection rate at particular points on the chip especially the ones that are closer to the timing sensor. However, attackers never attack just one point but instead will mostly attack at various points on the chip.

2. **Less hardware overhead:** One timing sensor is made up of only two flip-flops, one XOR gate, and one delay circuit. This logic is a minimal hardware overhead compared to other countermeasures. For DE1SoC, the hardware overhead comparison will be shown in the Chapter 5.

3. **Design independent:** The timing sensor is an independent module which is not attached to any design. This module can be just placed as the starting point in any FPGA/ASIC to protect it against EMFI attacks without worrying about the design. However, since the EMFI attacks are local, the location of the timer sensor around the chip matters which can make this property a disadvantage for detecting EMFI.

Despite the advantages, we believe there are certain important shortcomings listed below that renders standalone timing sensors insufficient for EMFI detection.

1. **High False Positives:** High delay values give higher detected faults, but they also have an extremely high number of false positives as seen form the alarm fault map in the Figure 4.6. If one tries to lower the delay values to decrease the false positives and the false negatives how an increasing trend as seen in the Table 4.8. The false positives also increases with an increase in intensity. If we look at the intensity sweep in the Table 4.4, it can be seen that even though the detection rate is reasonable, the number of high false positives makes this sensor very imprecise. A lower precision means a very high energy consumption.

2. **Location & Time Dependent:** The EM attack in an FPGA is highly localized.

Every viable EM attack in the chip has a unique relationship pair of glitch offset and location. Faults are observed when EM attack is performed at the rising edge of the clock, but not all rising edges lead to a fault. This behavior shows that EM attack is localized in space and time. Also, some locations on the FPGA where the clock is not propagating, placing timing sensors will never give rise to an alarm.

3. **Clock Tree Knowledge:** We have no visibility or knowledge about the clock tree in an FPGA since it is propriety to the FPGA vendors. So, while creating designs on an FPGA, one cannot know where does a clock originate and how does it propagate throughout the chip. This hidden clock tree makes it difficult to optimally place the timing sensors for best detection in an FPGA. In contrast, when designing an ASIC, since the clock layout is known, it is easier to decide the placement of the timer sensors. With the rise in low-power FPGA options on the market, reduced fabrication costs and increasingly reconfigurable designs, the companies using off-the-shelf FPGAs to build their products is now common. Therefore, no visibility of the clock tree in FPGA designs is a significant disadvantage for standalone timing sensors.

4. **Delay-Dependent:** The timing sensors have a delay circuitry as the basis for detecting alarms whose delay is dependent on the fabrication process. Therefore, it has to be adjusted for every separate FPGA as the number of delay units required to make this delay circuit may be different. We saw that the ideal delay we found for DE1-SoC was 70 Lcells whereas for DE0-nano it was 55 lcells. On increasing the delay, the true positives rise, the true negatives reduce, and the false positives increase to a considerable

extent.

In conclusion, from the above experiments, we saw that the average accuracy for standalone timing sensor for a chip scan is around 70%. This detection rate is not the ideal 100% because of the high number of false negatives which are seen with these timing sensors. If we increase the number of timing sensors, the detection rate has an increasing trend. Due to the limitation of limited clock tree visibility in FPGAs, determining the ideal position and number of timing sensors is difficult. This ideal location can change when the designs' location in the FPGA changes. The delay acts as a sensitivity control and needs to be set to an ideal value for detecting faults correctly. Even with an above average detection rate, we can see high false positives or less precision due to the dependence on delay and location. All these factors show that even though a standalone timing sensor has been determined to detect EMFI faults, it is not perfect. There is a need to have a detection mechanism which is robust regarding its location and has both high precision and accuracy.

4.3 In-situ EM Sensor

The top-level module for AES with EM sensor integrates the complementary flip-flop (SFF) along with the main flip-flop (MFF) in situ as proposed in [7]. Since there is an extra SFF for every design flop (MFF), the sensor is inbuilt into the design and is therefore called in situ. The alarm is an XNOR of the MFF, and the SFF flops inside the design for every in situ em sensor as shown in Figure 4.9. In this section, we study the efficiency of the In situ

EM sensor by performing experiments on the DE1-SoC and DE0-Nano boards.

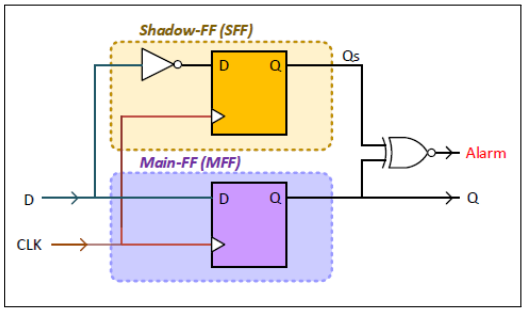


Figure 4.9: EM Sensor Diagram [7]

- **Effect of Attack Intensity**: These observations are taken in a similar manner to the timing sensor observations. The intensity observations can be seen from the Figure 4.10. To verify the results, these experiments were repeated with the DE0-Nano board, and the intensity observations can be seen from the Figure 4.11. The inferences and conclusions from these results will be elaborated in the next section

Intensity	True Negatives	False Positives	True Positives	False Negatives	Precision	Specificity	Detection Rate	Accuracy
70	384	0	0	0	NA	NA	NA	NA
75	364	18	2	0	10	95.3	100	95.3
80	295	32	57	0	64	90.2	100	91.7
85	236	38	110	0	74.3	86.1	100	90.1
90	213	21	149	1	87.6	91	99.3	94.3

Figure 4.10: DE1-SoC In Situ EM Sensor Intensity Sweep

- **Effect of Attack Offset**: The fault and alarm map with varying offset can be seen from the Figure 4.12. The observations regarding the DE0-Nano offset can be seen form the Figure 4.13. These observations are further analyzed in the next section.

Intensity	True Negatives	False Positives	True Positives	False Negatives	Precision	Specificity	Detection Rate	Accuracy
50	133	0	42	0	100	100	100	100
60	127	0	48	0	100	100	100	100
65	138	2	35	0	94.6	98.6	100	98.9
70	133	2	42	0	95.5	98.5	100	98.9
75	138	3	37	0	92.5	97.9	100	98.3
80	141	6	34	1	85	95.9	97.1	96.2
85	130	2	44	1	95.7	98.5	97.8	98.3
90	133	1	42	2	97.7	99.3	95.5	98.3

Figure 4.11: DE0-Nano In Situ EM Sensor Intensity Sweep

Alarm map	Offset (ns)										
Intensity	10	12	14	16	18	20	22	24	26	28	30
50	0	0	0	0	0	0	0	0	0	0	0
55	0	0	0	0	0	0	0	0	0	0	0
60	0	0	0	0	0	0	0	0	0	0	0
65	0	0	0	0	0	0	0	0	0	0	0
70	0	0	0	0	0	0	0	0	0	2	2
75	3	0	0	0	0	0	0	0	0	5	7
80	9	3	0	0	0	0	0	0	1	7	9
85	9	6	0	0	0	0	0	0	5	7	10
90	10	7	2	0	0	0	2	6	10	10	10

Fault Map	Offset (ns)										
Intensity	10	12	14	16	18	20	22	24	26	28	30
50	0	0	0	0	0	0	0	0	0	0	0
55	0	0	0	0	0	0	0	0	0	0	0
60	0	0	0	0	0	0	0	0	0	0	0
65	0	0	0	0	0	0	0	0	0	0	0
70	0	0	0	0	0	0	0	0	0	0	0
75	3	0	0	0	0	0	0	0	0	3	4
80	8	0	0	0	0	0	0	0	1	7	9
85	9	5	0	0	0	0	0	0	5	7	10
90	10	7	0	0	0	0	0	3	6	10	10

Figure 4.12: DE1-SoC EM Sensor Fault Alarm Map

- **Effect on Bit flips:** To perform the bit flip evaluation, all the flip-flops in each of the 12 cycles in the AES were read using separate read flops. These flops were logic locked separately from the actual design such that the EMFI should not affect them a shown in Figure 4.14.

  The initial C program was modified to calculate the total number of main flop bit

Fault Map	Offset (ns)										
Intensity	10	12	14	16	18	20	22	24	26	28	30
40	0	0	1	10	24	20	1	0	0	0	0
50	0	0	1	6	24	25	3	0	0	0	0
60	0	0	1	6	24	25	1	0	0	0	0
70	0	1	2	10	27	20	1	0	0	0	0
80	0	1	1	10	27	17	3	0	0	0	0
90	0	2	2	10	29	23	3	0	0	0	0

Alarm Map	Offset (ns)										
Intensity	10	12	14	16	18	20	22	24	26	28	30
40	0	0	1	10	24	20	1	0	0	0	0
50	0	0	1	6	24	25	3	0	0	0	0
60	0	0	1	6	24	25	1	0	0	0	0
70	0	1	2	10	27	20	1	0	0	0	0
80	0	1	1	10	27	17	3	0	0	0	0
90	0	2	2	10	29	23	3	0	0	0	0

Figure 4.13: DE0-Nano EM Sensor Fault Alarm Map

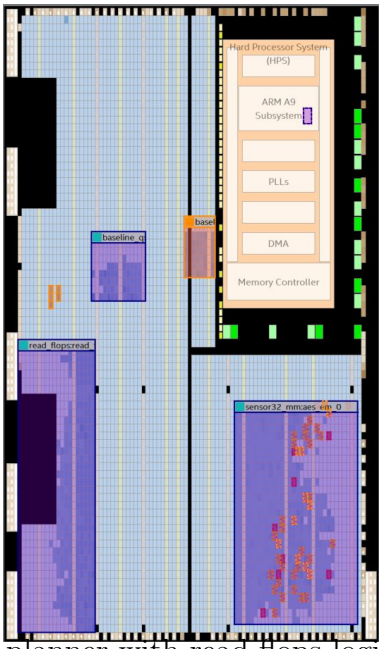


Figure 4.14: Chip planner with read flops logic locked separately

sets/resets and shadow flop bit sets/resets. The Figures 4.15 and 4.16 show how the results of this experiment are displayed. Figure 4.17 summarized the results for different offsets and fixed intensity of 80%. It was seen that EM attacks did not show only one type of behavior (bit sets/bit resets) instead the flops were affected by both bit sets/resets in the same cycle. It was also seen that the number of shadow bit flips were always higher than the number of main bit flips. This observation was possible

only if somehow shadow flop was affected with a higher probability. This led to the question of whether the not gate before the complementary flop behaved like a small delay.

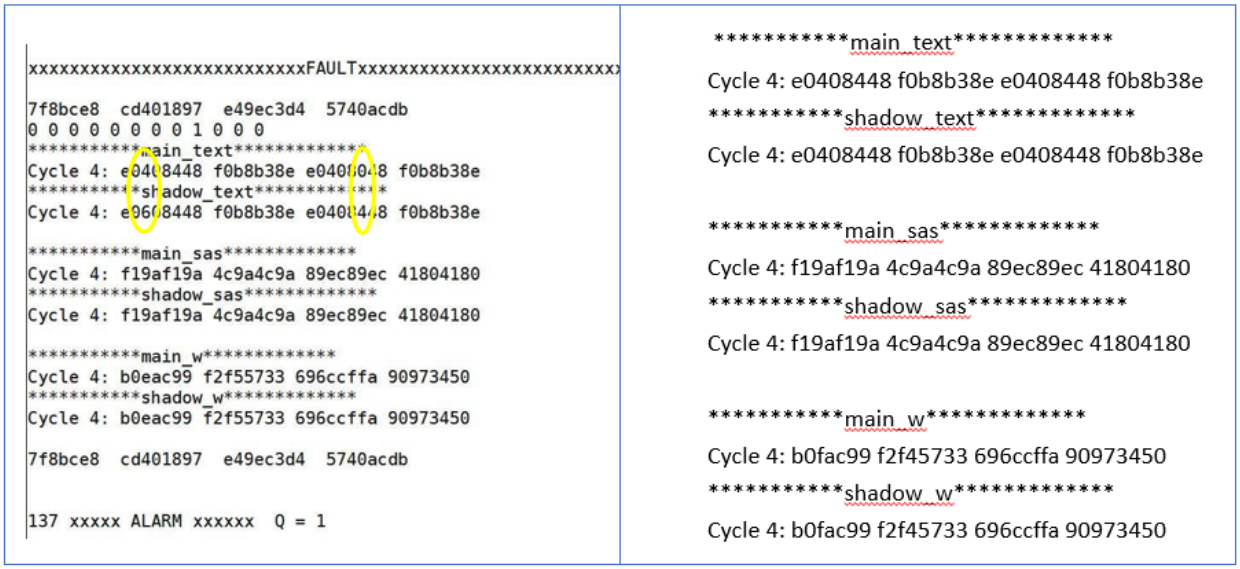


Figure 4.15: Bit Flip Observation for a True Positive case

Figure 4.16: Observing the Main and Complementary Bit Flips

Offset	Type	Total Sets	Total Resets	Main Bit Flips	Shadow Bit Flips
10	FP	16	6	8	14
10	TP	135	65	70	130
12	TP	90	21	27	84
12	TP	278	177	190	265
30	FP	51	41	6	86
30	TP	217	136	123	230

Figure 4.17: Shadow Flop Bit Flips vs Main Flop Bit Flips

4.4 In situ EM Sensor Advantages and Disadvantages

The in situ EM sensor was developed after the standalone timing sensor in order to detect EMFI. It addresses the shortcomings of standalone timing sensor (for EMFI detection) like the question of location, number of sensors required, clock dependence. This makes it a strong competitor for the role of EMFI detector. It's advantages can be listed as follows.

1. **Delay independent:** Since there is no delay in the design of the EM sensor with dual complementary flip-flops, it is free from the process and temperature variations that come along with it. Also, this saves us the effort in changing the delay if the design uses different FPGAs. Moreover, we do not need to spend extra time to find the optimal delay.

2. **Location Independent:** Since these sensors are in-situ with the design, their placement around the design is not a relevant constraint. Also, there is no requirement of

identifying the ideal number of in situ EM sensors as all the flops need to be duplicated just once for ideal detection.

3. **High precision:** We obtain higher precision values with these sensors compared to the timing sensors. This precision is mainly because the false positives are comparatively lesser in this countermeasure design.

Despite the significant advantages, our experiments exposed some drawbacks which are explained below.

1. **High hardware overhead:** It has a comparatively higher hardware overhead. We see from the design compilations that the in situ EM sensor has a 18% register overhead compared to a basic no-countermeasure design and 6% register overhead in comparison to the standalone timer sensors design.

In conclusion, though the in situ EM detection technique gives an almost 100% detection rate for DE0-Nano board, it is not the case for DE1-SoC board especially for higher intensities. Here, we see increasing false negatives as we increase the glitch source power but it is not as high as seen for the standalone timing sensor. The in-situ nature of this sensor is proposed to be the plausible reason for the low false positives. The in situ EM sensor was built on the assumption of detecting bit sets and bit resets. Even though recently, the authors in the previous works have mentioned that EM attack faults are similar to the bit set/reset fault model and sampling fault model, we felt there was more to the story. We probed further by performing the bit set/reset analysis of the dual complementary flip-flops and our

observations are not in sync with bit set/reset fault model. We suspect that the in situ EM sensor is, in fact, very similar to the design of standalone timing sensor. To validate this argument, we combined the ideas of the above two countermeasures and put forward the in situ timing sensor design which is discussed in the following chapter.

Chapter 5

In Situ Timing Sensor

In the previous chapters, we studied two of the existing EMFI countermeasures in detail. We found out that both the standalone timing sensor and the in situ EM sensor have their respective shortcomings.

5.1 Design

To build a better countermeasure that overcomes these limitations, we propose the idea of an in situ timing sensor for detection of EMFI attacks. We replaced the complementary flip-flop and the XNOR gate present in the in situ EM sensor by a corresponding delayed shadow flip-flop and XOR gate to get an in situ timing sensor as shown in the Figure 5.1. The delay in the shadow flip-flop helps in catching any glitches in the clock by keeping track of the delayed signal as shown in the figure 5.2. This delay has to be configured such that

when a glitch occurs, the main and shadow flops have different values. This difference will be detected by the XOR gate which will lead to an alarm. While performing experiments with in situ timing sensor, the only difference in hardware logic was the sensor design module.

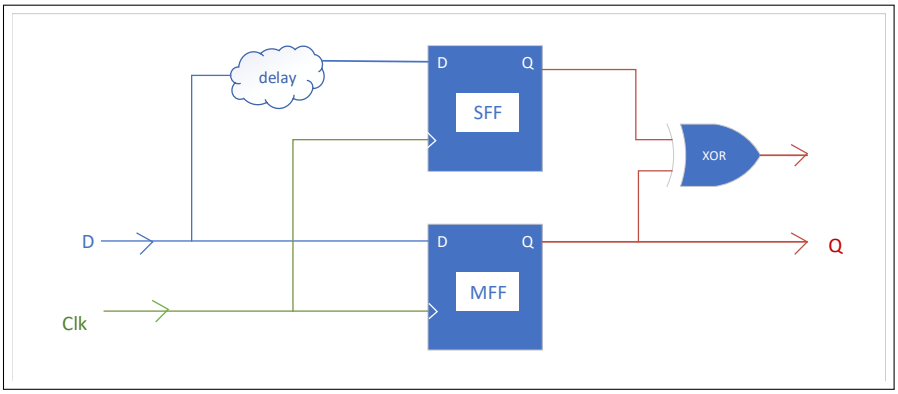


Figure 5.1: In Situ Timing Sensor Diagram

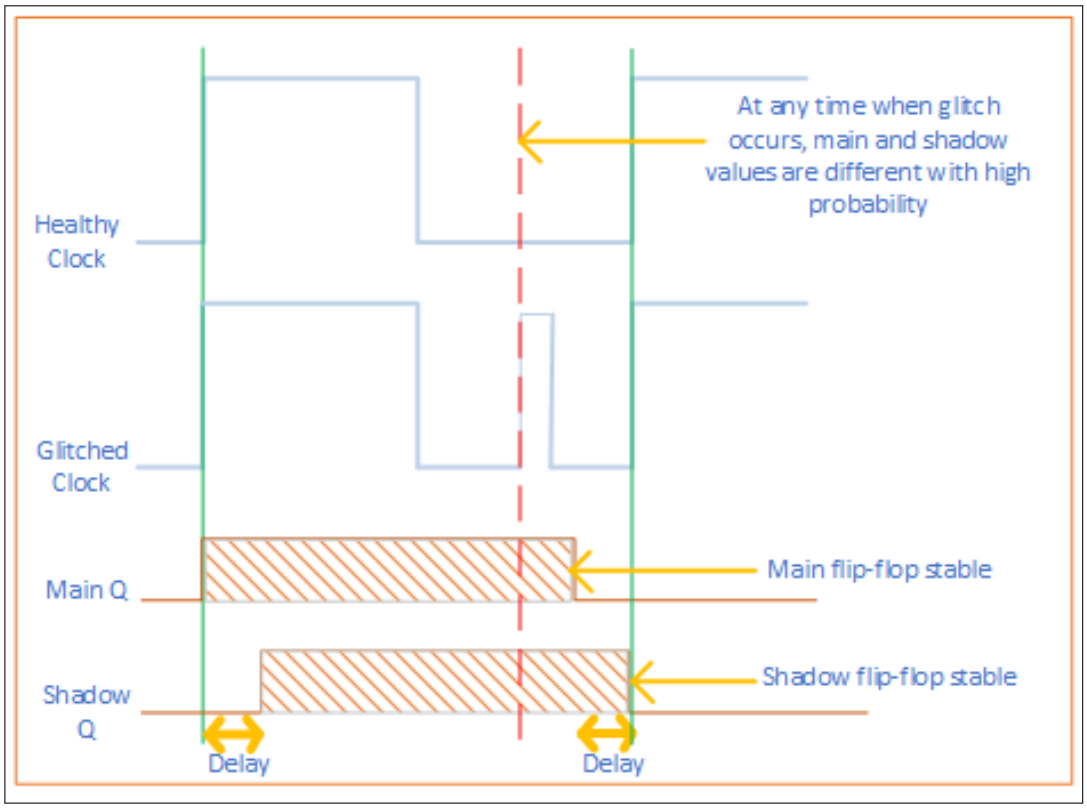


Figure 5.2: Delayed Shadow Signal to detect clock glitches

The Figure 5.3 summarizes our EMFI detection findings in both the in situ EM sensors and the in situ timing sensors with varying delay. The delay for a single NOT gate was found to be 0.4 ns, and the delay for the Lcell is observed to be around 0.4 to 0.5 ns using oscilloscope measurements. Therefore, similar delay values are expected for these techniques as both the NOT gate and Lcell behave like buffers with one having its input inverted. These results are also represented in the Figure 5.4. As can be seen from this graph, the detection statistics for an in situ timing sensor with one Lcell delay is very close to the those of an in situ EM sensor. Through these observations, one can argue that the in situ EM sensor, in fact, behaves like an in situ timing sensor having one Lcell delay.

Metric	In situ EM	In situ timing 1 Lcell	In situ timing 2 Lcells	In situ timing 4 Lcells	In situ timing 6 Lcells	In situ timing 12 Lcells
False Positives	32	26	17	7	4	1
True Positives	57	60	72	70	75	76
False Negatives	1	1	1	0	0	0
True Negatives	294	297	294	307	305	307
Detection Rate %	97	96.3	94.4	100	100	100
Precision %	64	69.8	80.9	90.9	94.9	98.7
Specificity %	90.2	92	94.5	97.8	98.7	99.7
Accuracy %	91.4	93	95.3	98.2	99	99.7

Figure 5.3: Comparison of in situ EM sensors with in situ timing sensors with varying Lcells

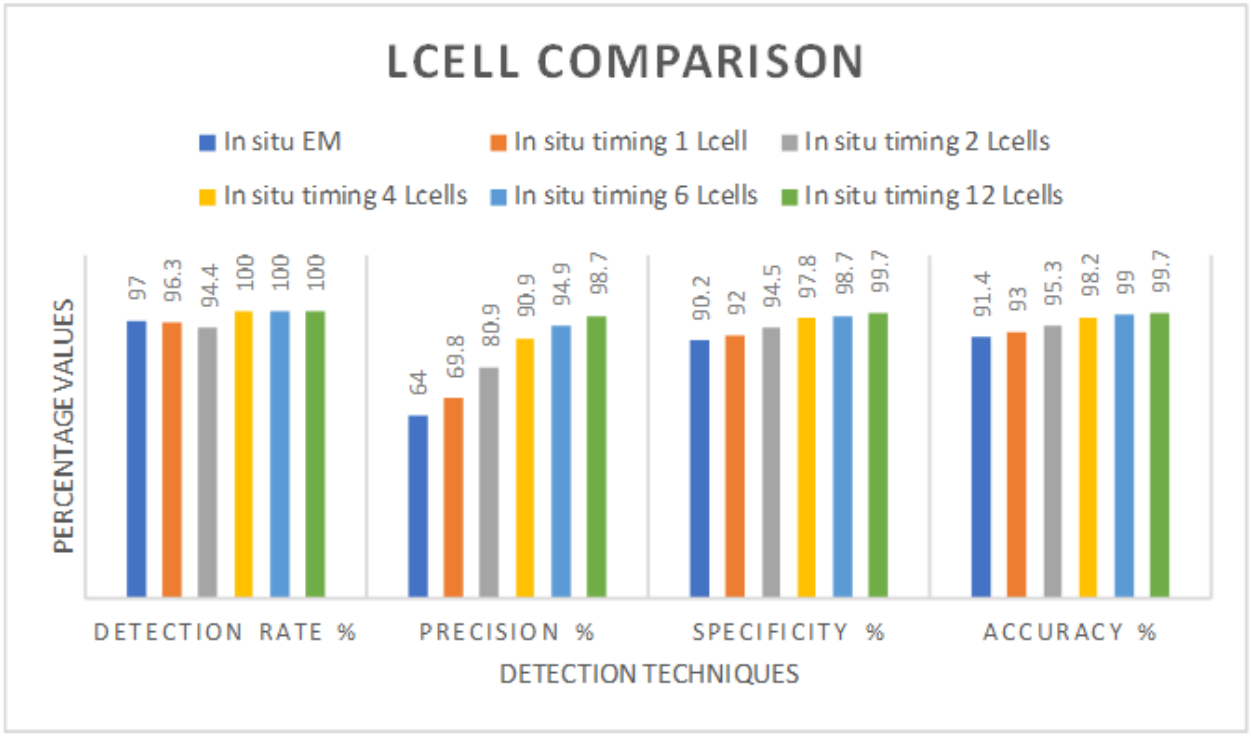


Figure 5.4: Comparing Detection Techniques with varying LCells

5.2 Experimental Results

- **Effect of attack Intensity:** The intensity sweep experiments showed promising results for the in situ timing sensor which can be seen from the Figure 5.5 for the case of DE1-SoC and the Figure 5.6 for the case of DE0-Nano. We can observe that as the intensity increases, the false positives increase slightly but they do not scale a lot as compared to the standalone timing sensor. We also notice that unlike the EM sensor there are no cases of false negatives for this experiment.

- **Effect of attack Offset:** The correlation between the alarm maps and the fault maps

Intensity	True Negatives	False Positives	True Positives	False Negatives	Precision	Specificity	Detection Rate	Accuracy
70	358	4	22	0	84.6	98.9	100	99
75	342	2	40	0	95.2	99.4	100	99.5
80	307	1	76	0	98.7	99.7	100	99.7
85	276	2	106	0	98.1	99.3	100	99.5
90	250	1	133	0	99.3	99.6	100	99.7

Figure 5.5: DE1-SoC In Situ Timing Sensor Intensity Sweep

Intensity	True Negatives	False Positives	True Positives	False Negatives	Precision	Specificity	Detection Rate	Accuracy
50	132	1	42	0	97.7	99.2	100	99.4
60	123	0	52	0	100	100	100	100
65	122	2	51	0	96.2	98.4	100	98.9
70	127	2	46	0	95.8	98.4	100	98.9
75	132	3	40	0	93	97.8	100	98.3
80	124	4	47	0	92.2	96.9	100	97.7
85	123	1	51	0	98.1	99.2	100	99.4
90	120	1	54	0	98.2	99.2	100	99.4

Figure 5.6: DE0-Nano In Situ Timing Sensor Intensity Sweep

with on different offsets can be seen from The Figure 5.7. High correlation between faults and alarms is a desirable metric as it shows both high precision and high accuracy. We can see that the correlation shown by this sensor is close enough to the expected correlation.

- **Bit-Flip Evaluation:** We performed the bit set-reset analysis of the in situ EM sensors in the previous chapter. We saw that for the situ EM sensor, the number of total bit sets and resets (we also call this bit flips) for the dual complementary flip-flops is considerably higher than the main flip-flops. The in situ timing sensors

Fault map	Offset (ns)										
Intensity	10	12	14	16	18	20	22	24	26	28	30
50	0	0	0	0	0	0	0	0	0	0	0
55	0	0	0	0	0	0	0	0	0	0	0
60	0	0	0	0	0	0	0	0	0	0	0
65	0	0	0	0	0	0	0	0	0	0	0
70	0	0	0	0	0	0	0	0	0	0	0
75	0	0	2	0	0	0	0	0	0	0	0
80	2	3	6	0	0	0	0	0	0	0	0
85	3	5	6	4	0	0	0	0	0	0	2
90	4	7	7	6	0	0	0	0	0	1	3

Alarm map	Offset (ns)										
Intensity	10	12	14	16	18	20	22	24	26	28	30
50	0	0	0	0	0	0	0	0	0	0	0
55	0	0	0	0	0	0	0	0	0	0	0
60	0	0	0	0	0	0	0	0	0	0	0
65	0	0	0	0	0	0	0	0	0	0	0
70	0	0	0	0	0	0	0	0	0	0	0
75	0	0	2	0	0	0	0	0	0	0	0
80	2	3	6	0	0	0	0	0	0	0	0
85	3	6	7	4	0	0	0	0	0	0	2
90	5	9	8	6	0	0	0	0	0	1	4

Figure 5.7: In situ Timing Sensor Faults vs Alarms Map

showed a similar relationship. The number of bit flips for the shadow flip-flops are again significantly higher than those for the main flip-fops. This showed that for the in situ timing sensors, the shadow flip-flop is affected more than the main flip-flop after EM injection. This behavior is expected if the attack is a timing attack wherein the clock glitch causes the delayed shadow flip-flop to latch a wrong value more easily than the main flip-flop causing more bit flips.

5.3 Comparison

In this section, we compare all the three detection techniques that were analyzed through this work. A comparative idea about the advantages and the disadvantages of all the three sensors can be seen from the Figure 5.8. In order to compare more quantitatively, we use various analysis techniques like detection metrics, area and performance.

Standalone Timing	In Situ EM	In Situ Timing
Location Dependent	Location Independent	Location Independent
Quantity Dependent	Quantity Independent	Quantity Independent
Low Hardware Overhead	Higher Hardware Overhead than standalone timing	Higher Hardware Overhead than In Situ EM
Design Non-intrusive	Design Intrusive	Design Intrusive
Configurable Sensitivity	Non-Configurable Sensitivity	Configurable Sensitivity

Figure 5.8: Qualitative Comparison

5.3.1 Detection

We can compare the results obtained for the three detection schemes using the detection metrics discussed in the Chapter 3. The comparative analysis of precision, accuracy, detection rate and specificity for both the FPGA boards can be seen from the Tables 5.9, 5.10, 5.11 and 5.12 respectively. We can also graphically plot the detection results that were obtained in the Table 4.4, 4.10 and 5.5. These graphs shown in the Figure 5.13 make comparing the sensors easier by showing the various trends in the detection metrics. Based on these graphs, we prepare a comparison table shown in the Figure 5.14 shows a comparative analysis of the three previously discussed detection techniques.

Sensor	Standalone Timing		In Situ EM		In Situ Timing	
Board	DE1	DE0	DE1	DE0	DE1	DE0
Precision	2 - 45	20 - 38	10 - 87	85 - 100	98 - 100	92 - 100

Figure 5.9: Precision Results

The high and low comparison of the metrics in the Figure 5.14 are shown in relation to each

Sensor	Standalone Timing		In Situ EM		In Situ Timing	
Board	DE1	DE0	DE1	DE0	DE1	DE0
Accuracy	58 - 68	25 - 64	90 - 95	96 - 100	98 - 100	98 - 100

Figure 5.10: Accuracy Results

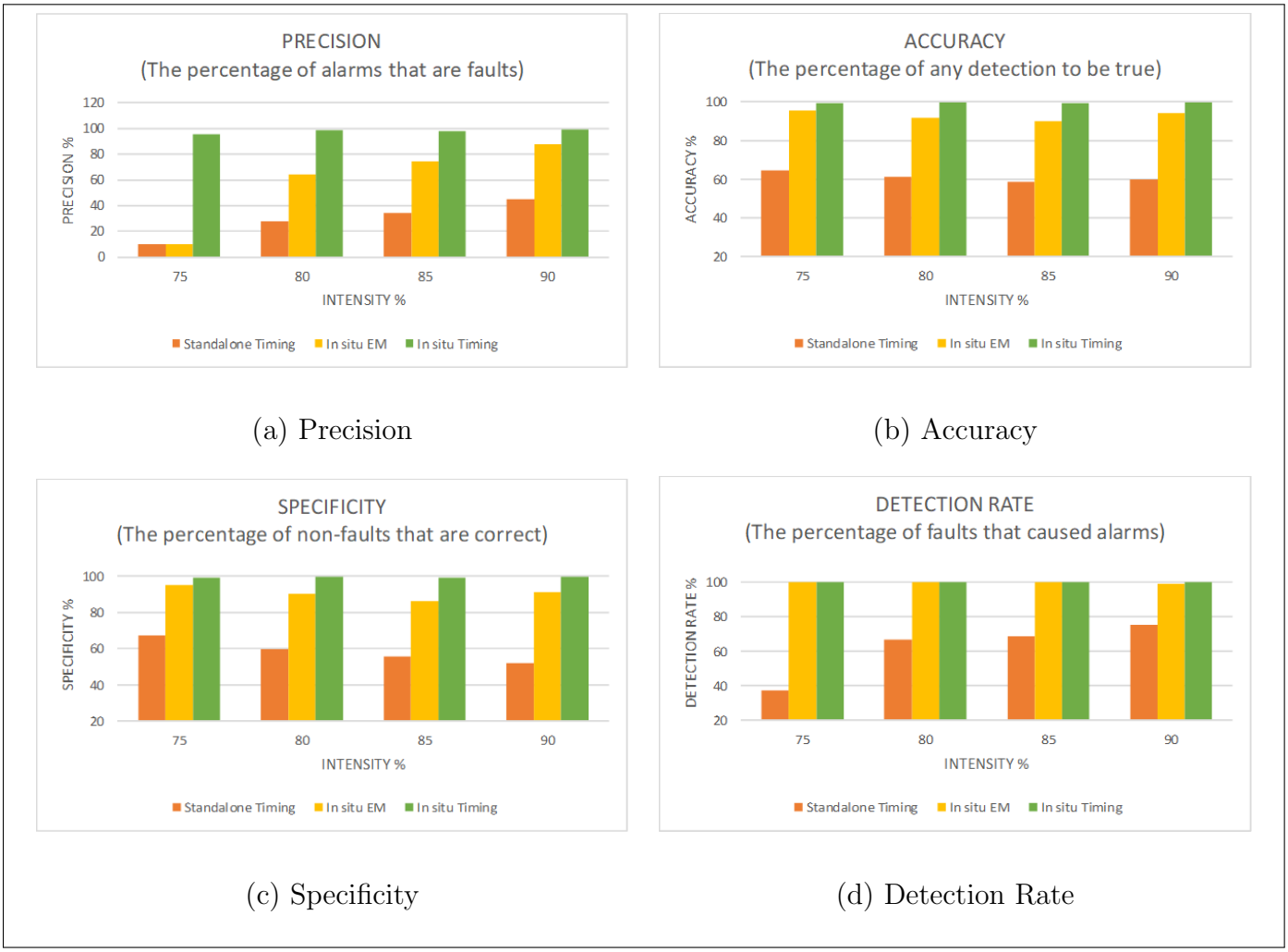
Sensor	Standalone Timing		In Situ EM		In Situ Timing	
Board	DE1	DE0	DE1	DE0	DE1	DE0
Detection Rate	33 - 75	91 - 100	99 - 100	95 - 100	100	100

Figure 5.11: Detection Rate Results

Sensor	Standalone Timing		In Situ EM		In Situ Timing	
Board	DE1	DE0	DE1	DE0	DE1	DE0
Specificity	52 - 69	5 - 54	86 - 95	95 - 100	98 - 100	96 - 100

Figure 5.12: Specificity Results

other. We saw that a high number of false positives leads to lesser precision whereas a high number of false negatives results in a lower detection rate. By varying the delay of the in situ timing sensors, we saw from the delay sweep analysis that it is possible to find the ideal delay where both the false positives and false negatives are low. We also observe from the Figure 5.14 that the overall accuracy of the in-situ timing sensor is higher than that of the standalone timing sensor and the in situ EM sensor.



(a) Precision

(b) Accuracy

(c) Specificity

(d) Detection Rate

Figure 5.13: Graphical Comparison of Detection metrics

5.3.2 Area and Power

We used the 16 standalone timing sensors for design area comparison, but the ideal number of timing sensors that are optimal can vary. The actual area overhead obtained for our designs in comparison to a baseline design with no detection technique is shown in the Figure 5.15. We can observe that the area overhead for in situ EM sensor is much higher compared to

Detection Metrics	Ideal Sensor	Standalone Timing Sensor	In situ EM Sensor	In situ Timing Sensor
False Positives %	0	High (29 - 33)	Average (4 - 10)	Low (< 1)
False Negatives %	0	Average (2 - 8)	Low (< 1)	Low (< 1)
Precision %	100	Low (2 - 45)	Above Average (10 - 87)	High (98 - 100)
Detection Rate %	100	Average (33 - 75)	High (99 - 100)	High (100)
Specificity %	100	Average (52 - 69)	Above Average (86 - 95)	High (98 - 100)
Accuracy %	100	Average (58 - 68)	Above Average (90 - 95)	High (98 - 100)

Figure 5.14: Overall Comparison of Detection Metrics

the standalone timing sensor. In the in situ timing sensor, the number of registers required is exactly same as the in situ EM sensor but the combinational logic overhead is slightly greater due to the extra number of Lcells required for the optimum delay. In small scale FPGAs designs, a lot of area is not utilized which can be used for implementing the in situ timing sensors making the design more secure.

Area Metric	Baseline	Standalone Timing	In Situ EM	In Situ Timing with 12 Lcells
Combinational LUT usage	1837	2770	6165	9959
Total Registers	2654	2750	2919	2919

Figure 5.15: Area Comparison

Power / Detection Techniques	Standalone Timing Sensor (mW)	In Situ EM Sensor (mW)	In Situ Timing Sensor (mW)
Total Thermal Power Dissipation	451.7	454.86	514.69
Core Dynamic Thermal Power Dissipation	14.5	17.59	77.04
Core Static Thermal Power Dissipation	411.42	411.43	411.79
I/O Thermal Power Dissipation	25.78	25.84	25.86

Figure 5.16: Power Comparison

The power comparison can be seen from the the Figure 5.16. The static and I/O thermal power dissipation values for all the three detection techniques are almost similar. We see the difference primarily in the dynamic power dissipation values. The total thermal power dissipation and the core dynamic thermal power dissipation of the standalone timing sensor detection technique is slightly lower than the in situ EM sensor detection technique. Whereas the total and dynamic power dissipation of the in situ timing sensor is higher than both the existing detection techniques. We believe that the slight area and power overhead of the in situ timing sensor from the in situ EM sensor is a small price to pay for a substantial improvement in accuracy.

5.3.3 Performance

We performed experiments to compare the maximum achievable frequency for the proposed in situ timing sensor and the previous in situ EM sensor. We saw that the in situ timing sensor can operate at a maximum frequency of 127 MHz without any timing violations and

the in situ EM sensor can go up to 164 MHz. This shows that the our sensor performs slightly lower than the previous in situ EM sensor techniques in relation to the maximum operating speed. This is due to the extra delays in the in situ timing sensor detection which reduces the critical path time. The overall performance of an EMFI sensor is also dependent on its detection rate, precision and accuracy. The in situ timing sensor incorporates the advantages of both the in situ EM sensor technique and the standalone timing sensor technique. This is especially true compared to the standalone timing sensor as our sensor overcomes the limitations of location and number of timing sensors and clock tree knowledge. Accuracy is a valuable metric in terms of performance and we show that the accuracy for our sensor is higher than the in situ EM sensor due to the lesser false detections. The minimal number of false positive and false negatives also helps in reducing the time to completion. In all, our results show that even though the in situ timing sensor performs better detection than both the standalone timing sensor and the in situ EM sensor.

Hence, it can be seen from the above comparisons that with some cost on the area, the power and the operational frequency; the in situ timing sensor performs better EMFI detection than the other techniques which are discussed.

Chapter 6

Conclusion

Hardware security is a continually evolving field wherein with the growth in technology there is also a rise in the threats to the hardware. It is critical to catch up with the times and promptly detect these attacks. It is not enough to merely identify an attack but in fact it more valuable to detect it with high precision and accuracy. We discussed the growing area of Electromagnetic (EMFI) attacks in this work. We extensively evaluated two of the primary existing countermeasures on FPGA platforms which also gave us a better perspective regarding the nature of electromagnetic attacks. By carefully analyzing the advantages and disadvantages of these detection methods, we combined the ideas learned from them and proposed a new EMFI countermeasure. We believe that this countermeasure will not only help in identifying attacks but also detect viable attacks with better confidence. The comparison of all the three techniques helped in studying the EMFI countermeasures on the same reference ground to decide their efficacy without any bias.

Appendix A

Softwares Code DE1-SoC

C Program

```
#include <stdint.h>

#include <string.h>

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>          // close

#include <fcntl.h>           // O_RDWR, O_SYNC

#include <sys/mman.h>        // PROT_READ, PROT_WRITE


#include "socal/socal.h"     // alt_write

#include "socal/hps.h"

#include "socal/alt_gpio.h"
```

```
#include "hps_0.h"          // definitions for LED_PIO and SWITCH_PIO

#define HW_REGS_BASE ( ALT_STM_OFST )
#define HW_REGS_SPAN ( 0x04000000 )
#define HW_REGS_MASK ( HW_REGS_SPAN - 1 )

int main(int argc, char **argv) {

  void *virtual_base;
  volatile unsigned long *led_pio, *switch_pio, *trigger_pio, i;
  volatile unsigned *em_base;
  int fd; int k;
  if( ( fd = open( "/dev/mem", ( O_RDWR | O_SYNC ) ) ) == -1 ) {
    printf( "ERROR: could not open \"/dev/mem\"...\n" );
    return( 1 );
  }
  virtual_base = mmap( NULL,
            HW_REGS_SPAN,
            ( PROT_READ | PROT_WRITE ),
            MAP_SHARED,
```

```
            fd,
            HW_REGS_BASE );
 if( virtual_base == MAP_FAILED ) {
    printf( "ERROR: mmap() failed...\n" );
    close( fd );
   return(1);
 }
 led_pio = virtual_base +
    ( ( unsigned long )( ALT_LWFPGASLVS_OFST + LED_PIO_0_BASE) & (
        unsigned long)( HW_REGS_MASK ) );
switch_pio = virtual_base +
    ( ( unsigned long )( ALT_LWFPGASLVS_OFST + SWITCH_PIO_1_BASE) & (
        unsigned long)( HW_REGS_MASK ) );
 trigger_pio = virtual_base +
    ( ( unsigned long )( ALT_LWFPGASLVS_OFST + PIO_0_BASE) & ( unsigned
        long)( HW_REGS_MASK ) );


 printf("hello\n");
em_base = mmap( NULL,
      AES_EM_0_SPAN,
      ( PROT_READ | PROT_WRITE ),
```

```
    MAP_SHARED,
    fd,
    0xC0000000 + AES_EM_0_BASE);
if (em_base == MAP_FAILED) {
  printf("ERROR: mmap() failed ...\n");
  return 1;
}
int ufd;
  ufd = open("/dev/ttyS0", O_RDWR | O_NOCTTY | O_SYNC);
if (ufd == -1) {
  printf("Couldn't open serial port\r\n");

  return EXIT_FAILURE;
}

char c = 0;
char finalc[8] = {0};
int tnum = 0;
unsigned int count = 0;
char set = '1';
int value, countvalue;
```

```
int hw_counter = 0, hw_counter_prev = 0;
int o_alarm;
printf("hello\n");
int ia;
int out0, out1, out2, out3;
char fault = 0;
int a = 0;


while (1) {
   fcntl( ufd ,F_SETFL ,0 );
   char rbuf[80];
   int rdlen;
   do {
   rdlen = read(ufd, rbuf, sizeof(rbuf) - 1);
   if (rdlen > 0){
   }
   } while (rbuf[0] != set);
   *led_pio = *switch_pio;
  fault = 0;
 if (a == 0){
```

```
  em_base[1] = 0xCAFEBABE;
  em_base[2] = 0xDEADBEEF;
  em_base[3] = 0xCAFEBABE;
  em_base[4] = 0xDEADBEEF;

  em_base[5] = 0x00000000;
  em_base[6] = 0x00000000;
  em_base[7] = 0x00000000;
  em_base[8] = 0x00000000;

em_base[0] = 1;
em_base[0] = 0;

int main[48], sas_main[48], w_main[48];
int shadow[48], sas_shadow[48], w_shadow[48];

while (em_base[4] != 1);
out0 = em_base[0];
out1 = em_base[1];
out2 = em_base[2];
out3 = em_base[3];
```

```
c = em_base[5];


//read textout main flops
for (i = 0; i < 48; i++){
  main[i] = em_base[10+i];
}
//read textout shadow flops
for (i = 0; i < 48; i++){
  shadow[i] = em_base[58+i];
}
//read sa main flops
for (i = 0; i < 48; i++){
  sas_main[i] = em_base[106+i];
}
//read sa shadow flops
for (i = 0; i < 48; i++){
  sas_shadow[i] = em_base[154+i];
}
//read w main flops
for (i = 0; i < 48; i++){
  w_main[i] = em_base[202+i];
```

```
}
//read w shadow flops
for (i = 0; i < 48; i++){
  w_shadow[i] = em_base[250+i];
}
if ( out0 != 0xf9a079eb || out1 != 0xf7f8b11f || out2 != 0x939f0795 || out3 !=
    0xcb7d1d65){
  printf("xxxxxxxxxxxxxxxxxxxxxxxxxxxxFAULTxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx\n\n");
    printf("%x %x %x %x\n", out0, out1, out2, out3);
fault = 1;
*led_pio = 0x3FF;
}
int fault_i = 33333;
i = 0;
int flag = 0;
while( i<48 ){
  if ((main[i] == (shadow[i])) && (main[i+1] == (shadow[i+1])) && (main[i+2] ==
      (shadow[i+2])) && (main[i+3] == (shadow[i+3]))
  && (sas_main[i] == (sas_shadow[i])) && (sas_main[i+1] == (sas_shadow[i+1])) &&
      (sas_main[i+2] == (sas_shadow[i+2])) && (sas_main[i+3] == (sas_shadow[i+3]))
    && (w_main[i] == (w_shadow[i])) && (w_main[i+1] == (w_shadow[i+1])) &&
```

```
        (w_main[i+2] == (w_shadow[i+2])) && (w_main[i+3] == (w_shadow[i+3]))){
    printf("0 ");
  }
  else {
    printf("1 ");
    flag = 1;
    fault_i = i;
  }
  i = i + 4;
}
printf("\n");
if (flag == 1){
  printf("**********main_text**********\n");
  printf("Cycle %d: %x %x %x %x \n", (12 - (fault_i/4)),
      main[fault_i],main[fault_i+1],main[fault_i+2],main[fault_i+3]);
  printf("**********shadow_text**********\n");
  printf("Cycle %d: %x %x %x %x \n\n", (12 - (fault_i/4)),
      (shadow[fault_i]),(shadow[fault_i+1]),(shadow[fault_i+2]),(shadow[fault_i+3]));

  printf("**********main_sas**********\n");
  printf("Cycle %d: %x %x %x %x \n", (12 - (fault_i/4)),
```

```
        sas_main[fault_i],sas_main[ fault_i +1],sas_main[ fault_i +2],sas_main[ fault_i +3]);
    printf (”**********shadow_sas**********\n”);
    printf (”Cycle %d: %x %x %x %x \n\n”, (12 − (fault_i/4)),
        (sas_shadow[fault_i]),(sas_shadow[fault_i+1]),(sas_shadow[ fault_i +2]),(sas_shadow[ fault_i +3]));


    printf (”**********main_w**********\n”);
    printf (”Cycle %d: %x %x %x %x \n”, (12 − (fault_i/4)),
        w_main[fault_i],w_main[fault_i+1],w_main[fault_i+2],w_main[fault_i+3]);
    printf (”**********shadow_w**********\n”);
    printf (”Cycle %d: %x %x %x %x \n\n”, (12 − (fault_i/4)),
        (w_shadow[fault_i]),(w_shadow[fault_i+1]),(w_shadow[fault_i+2]),(w_shadow[fault_i+3]));
    }
for (ia = 0; ia < 1000000; ia++);
    }
printf (”%x %x %x %x\n\n\n”, out0, out1, out2, out3);
    count = count + 1;
    if (c == 0) {
      value = write(ufd, &c, 1);
      printf (”%d Q = %d\n\n”, count, c);
      } else {
      value = write(ufd, &c, 1);
```

```
        printf("%d xxxxx ALARM xxxxxx Q = %d\n\n", count, c);


    }
  value = write(ufd, &fault, 1);
  }
  if( munmap( virtual_base, HW_REGS_SPAN ) != 0 ) {
    printf( "ERROR: munmap() failed...\n" );
    close( fd );
    return( 1 );
  }
  close( fd );
  return 0;
}
```

Appendix B

Softwares Code DE0-Nano

TCL Script

```
set jtag_master [lindex [get_service_paths master] 0]

open_service master $jtag_master



puts "hello"

set i 0

set fn 0

set fp 0

set tp 0

set tn 0

while {$i<500} {

  incr i
```

```
puts "before"

master_write_32 $jtag_master 0x0 0x1
master_write_32 $jtag_master 0x0 0x0

puts "after"

set r [master_read_32 $jtag_master 0x00 5]
scan $r "%lx %lx %lx %lx %lx" result1 result2 result3 result4 alarm
puts [format "%d r1 = %lx r2 = %lx r3 = %lx r4 = %lx alarm = %lx" $i $result1
    $result2 $result3 $result4 $alarm]

if {$result1 == 0xf9a079eb && $result2 == 0xf7f8b11f && $result3 == 0x939f0795
    && $result4 == 0xcb7d1d65 && $alarm == 0x0} {
    incr fn
    puts [format "false_negative %d false_positive %d true_positive %d true_negative
        %d" $fn $fp $tp $tn]
  } elseif {$result1 == 0xf9a079eb && $result2 == 0xf7f8b11f && $result3 ==
      0x939f0795 && $result4 == 0xcb7d1d65 && $alarm == 0x1} {
    incr fp
```

```
        puts [format "false_negative %d false_positive %d true_positive %d true_negative
            %d" $fn $fp $tp $tn]
    } elseif {$result1 != 0xf9a079eb || $result2 != 0xf7f8b11f || $result3 !=
        0x939f0795 || $result4 != 0xcb7d1d65 && $alarm == 0x1} {
        incr tp
        puts [format "false_negative %d false_positive %d true_positive %d true_negative
            %d" $fn $fp $tp $tn]
    } else {
        incr tn
        puts [format "false_negative %d false_positive %d true_positive %d true_negative
            %d" $fn $fp $tp $tn]
  }

  after 600

}
```

Appendix C

Inspector Software Code

Java Module

```
public class FPGA_CHIP_timing extends BasicSequence {


    private PIC_P_RSettings settings = new PIC_P_RSettings();
    private RawIODevice rawIODevice1;
    private ResetLineDevice resetLineDevice1;
    public static final int READ_TIMEOUT_MS = 500;
    private static final int BOOT_TIME_MS = 100;


    @Override
    protected void init() {
        // Get all devices from hardware manager
```

```
        rawIODevice1 = getRawIODevice(settings.getRawIODevice1());
        resetLineDevice1 = getResetLineDevice(settings.getResetLineDevice1());

        // Set default devices
        setDefaultDevice(rawIODevice1);
        setDefaultDevice(resetLineDevice1);

        // If any error occur the sequence will fail
        onError(FAIL);
        // Open all devices
        open(rawIODevice1);
        open(resetLineDevice1);

        setProperty(rawIODevice1, "baudrate", 115200);
        connect(rawIODevice1);
         sleep(20);

}
        byte[] g_counter = hex("00");
        byte[] y_counter = hex("00");
        byte[] r_counter = hex("00");
```

```
    byte[] w_counter = hex("00");
@Override
public void run() {
boolean timeout = false;
    sleep(100);//Post−arming time delay in ms for the scope to get ready
    // Set the default verdict to inconclusive
    verdict(INCONCLUSIVE);

    // Arm the measurement setup
    arm();
    sleep(100);//Post−arming time delay in ms for the scope to get ready

    onError(IGNORE);

    byte[] expected_result = hex("00 00");
    byte[] false_positive = hex("01 00");
    byte[] true_negative = hex("00 01");
    byte[] no_fault = hex("00 00");

    byte[] command_set_pt = {(byte) 49}; //ascii 1, hex 31
```

```
 int Size1 = 10;


write(rawIODevice1, command_set_pt, NO_LOG);
byte[] response = readAll(rawIODevice1, Size1 , READ_TIMEOUT_MS, 0);
String res1 = new String (response);
System.out.println(res1);


BufferedWriter output = null;
try {
    File file = new File("H:/Surabhi/FPGA_test.txt");
    output = new BufferedWriter(new FileWriter(file, true));
    output.append(res1);
    output.close();
} catch(IOException e) {
    e.printStackTrace();
}


softTrigger();
appendLog(hex(" A "));
appendLog(response);  //s_g
appendLog(hex(" E "));
```

```
    appendLog(expected_result);
    appendLog(g_counter);
    appendLog(y_counter);
    appendLog(r_counter);
    appendLog(w_counter);

    //no fault  no alarm
    if  (Arrays.equals(Arrays.copyOfRange(response, 0, 2), expected_result)) {
        verdict(NORMAL);
       g_counter[0]++;
    }
  //alarm but not fault
   else  if  (Arrays.equals(Arrays.copyOfRange(response, 0, 2),  false_positive )) {
        verdict(INCONCLUSIVE);
        y_counter[0]++;

    }
  //fault  but no alarm
     else  if  (Arrays.equals(Arrays.copyOfRange(response, 0, 2), true_negative)) {
        verdict(UNKNOWN);
        w_counter[0]++;
```

```
        }
         else if (response.length == 0) {
            response = new byte[Size1];
            //timeout = true;
            verdict(UNKNOWN);
            System.out.println("string is empty");
        }
        //both fault and alarm
          else {
            verdict(SUCCESSFUL);
            r_counter[0]++;
        }
    }
    //Reset the board (using the soft reset line)
    protected void softReset() {
        assertReset();
        sleep(100);
        deassertReset();
        sleep(BOOT_TIME_MS);
    }
```

```
//Powercycle the board (using the Vcc line)
@Override
protected void onError(Throwable t) throws RuntimeException {
    if (t instanceof TimeoutException) {
        // Ignore
    } else {
        throw new RuntimeException(t);
    }
}
@Override
public void close() {
    // Close all devices, note that closing does not mean powering down
    close(rawIODevice1);
    close(resetLineDevice1);
}
@Override
public PIC_P_RSettings getSettingsBean() {
    return settings;
}
@Override
```

```
    public void setSettingsBean(Object settings) {
        this.settings = (PIC_P_RSettings) settings;
    }

    public static class PIC_P_RSettings {
  //kept as default

        @Reference(RawIODevice.class)
        @DisplayName("Raw I/O Device 1")
        @NotNull
        private ServiceReference rawIODevice1;


        @Reference(ResetLineDevice.class)
        @DisplayName("Reset Device 1")
        @NotNull
        private ServiceReference resetLineDevice1;

        public ServiceReference getRawIODevice1() {
            return rawIODevice1;
        }
```

```
public void setRawIODevice1(ServiceReference rawIODevice1) {
    this.rawIODevice1 = rawIODevice1;
}

public ServiceReference getResetLineDevice1() {
    return resetLineDevice1;
}

public void setResetLineDevice1(ServiceReference resetLineDevice1) {
    this.resetLineDevice1 = resetLineDevice1;
}

// Custom parameter examples
@DisplayName("Example Parameter1")
@Unit("Unit")
@DecimalMin("0")
@DecimalMax("10000000")
@DecimalStep("4")
@Presentation(slider = false)
@Perturbation // This makes the parameter shows up in Perturbation tab.
```

```
private BigDecimal perturParam = BigDecimal.valueOf(4);

public BigDecimal getPerturParam() {
  return perturParam;
}

public void setPerturParam(BigDecimal perturParam) {
  BigDecimal old = this.perturParam;
  this.perturParam = perturParam;
  pcs.firePropertyChange("perturParam", old, this.perturParam);
}

@DisplayName("Example Parameter2")
@Unit("Unit")
@DecimalMin("4")
@DecimalMax("1000000")
@DecimalStep("4")
@Presentation(slider = false)

private BigDecimal targetParam = BigDecimal.valueOf(4);
```

```
public BigDecimal getTargetParam() {
  return targetParam;
}

public void setTargetParam(BigDecimal targetParam) {
  BigDecimal old = this.targetParam;
  this.targetParam = targetParam;
  pcs.firePropertyChange("targetParam", old, this.targetParam);
}

/*
 * Property change support
 */
private PropertyChangeSupport pcs = new PropertyChangeSupport(this);

public void addPropertyChangeListener(PropertyChangeListener listener) {
    this.pcs.addPropertyChangeListener(listener);
}

public void addPropertyChangeListener(String propertyName,
```

```
            PropertyChangeListener listener) {
            this.pcs.addPropertyChangeListener(propertyName, listener);
        }

        public void removePropertyChangeListener(PropertyChangeListener listener) {
            this.pcs.removePropertyChangeListener(listener);
        }

        public void removePropertyChangeListener(String propertyName,
            PropertyChangeListener listener) {
            this.pcs.removePropertyChangeListener(propertyName, listener);
        }

    }
}
```

Bibliography

[1] Altera. Designing with low-level primitives. 2018.

[2] D. Boneh, R. A. DeMillo, and R. J. Lipton. On the Importance of Checking Cryptographic Protocols for Faults. In *Advances in CryptologyEUROCRYPT97*, pages 37–51. Springer, 1997.

[3] 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.

[4] J. Daemen, V. Rijmen, and A. Proposal. Rijndael. In *Proceedings from the First Advanced Encryption Standard Candidate Conference, National Institute of Standards and Technology (NIST)*, 1998.

[5] E.-B. David, J.-B. Rigaud, and P. Maurine. A fully-digital em pulse detector. In *DATE: Design, Automation and Test in Europe*, number ID164, 2016.

[6] 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.

[7] 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.

[8] T. Fawcett. An introduction to roc analysis. *Pattern Recogn. Lett.*, 27(8):861–874, June 2006.

[9] M. Ghodrati, B. Yuce, S. Gujar, C. Deshpande, L. Nazhandali, and P. Schaumont. Inducing local timing fault through em injection. In *Proceedings of the 55th Annual Design Automation Conference*, DAC '18, pages 142:1–142:6, New York, NY, USA, 2018. ACM.

[10] 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.

[11] 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.

[12] S. Ordas, L. Guillaume-Sage, and P. Maurine. Em injection: Fault model and locality. In *Fault Diagnosis and Tolerance in Cryptography (FDTC), 2015 Workshop on*, pages 3–13. IEEE, 2015.

[13] S. Ordas, L. Guillaume-Sage, K. Tobich, J.-M. Dutertre, and P. Maurine. Evidence of a larger em-induced fault model. In *International Conference on Smart Card Research and Advanced Applications*, pages 245–259. Springer, 2014.

[14] Riscure. Vc glitcher datasheet. 2018.

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

[16] B. Yuce. Fault attacks on embedded software: New directions in modeling, design, and mitigation (ph.d). Virginia Tech, 2018.

[17] 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.

[18] 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.

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