Hardware Fault Attack Detection Methods for Secure Embedded Systems

Chinmay Deshpande

Dissertation submitted to the Faculty of the

Virginia Polytechnic Institute and State University

in partial fulfillment of the requirements for the degree of

Masters of Science

in

Computer Engineering

Leyla Nazhandali, Chair

Patrick R. Schaumont

Zeng, Haibo

Sep 15, 2017

Blacksburg, Virginia

Keywords: Fault Attack, Countermeasure, Hardware Detection

Hardware Fault Attack Detection Methods for Secure Embedded Systems

Chinmay Deshpande

ABSTRACT

In our daily life, we are increasingly putting our trust in embedded software applications, which run on a range of processor-based embedded systems from smartcards to pay-TV units. This trend expands the threat model of embedded applications from software into hardware. Over the last 20 years, fault attacks have emerged as an important class of hardware attacks against embedded software security. In fault attacks, an adversary breaks the security by injecting well chosen, targeted faults during the execution of embedded software, and systematically analyzing softwares fault response.

In this work, we propose cycle-accurate and fully digital techniques that can efficiently detect different types of fault attacks. The detection methods are low-cost regarding the area and power consumption and can be easily implemented using the standard cell based VLSI design flow. In addition to the architecture of the detectors, we present a detailed analysis of the design considerations that affect the cost and accuracy of the detectors. The functionality of the detectors is validated by implementing on ASIC and FPGA platforms (Spartan-6, Cyclone IV). Additionally, the proposed detection methods have demonstrated to successfully detect all of the injected faults without any false alarm.

Hardware Fault Attack Detection Methods for Secure Embedded Systems

Chinmay Deshpande

ABSTRACT (GENERAL AUDIENCE)

Embedded systems nowadays play a very crucial role in day to day life. They are always gathering sensitive and private data of the users. So they become an attractive target for the attackers to steal this important data. As a result, the security of these devices has become a grave concern.

Fault attacks are a class of hardware attacks where the attacker injects faults into the system while it is executing a known program and observes the reaction. The abnormal reactions of the system are later analyzed to obtain the valuable data. Several mechanisms to detect such attacks exist in the literature, but they are not very effective. In this work, we first analyze the effect of different types of fault attacks on the embedded processor. Then we propose various low-cost digital techniques that can efficiently detect these attacks.

Acknowledgments

I want to thank my committee Dr. Leyla Nazhandali, Dr. Patrick Schaumont, and Dr. Haibo Zeng, for mentoring me and overseeing my work. Thank you to everyone in the Secure Embedded Systems Lab for helping me, including Bilgiday Yuce, Abhishek Bendre, Nahid Ghalaty, Conor Patrick, and Yuan Yao. Thank you to Dr. Nazhandali for helping me come up with new ideas and setting a good example for how to write and convey ideas to others. I've learned a lot during my time in grad school.

Thank you to my friends and family for supporting me.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The daily life of the modern society relies on a range of secure embedded devices such as smartcards, smart phones, implantable medical devices, and pay-TV systems. These devices store, transfer, and process the sensitive data of both users (e.g., passwords, personal data) and vendors (e.g., intellectual property, cryptographic keys). Therefore, secure embedded systems employ security mechanisms to protect confidentiality, integrity, and availability of the sensitive data. The employed security mechanisms are efficient against traditional software-oriented attacks. However, because of their pervasive nature, the secure embedded systems are subject to hardware-oriented attacks that exploit vulnerabilities in the physical implementation of the security mechanisms.

An important class of hardware-oriented attacks is fault attacks, which use fault injection as a hacking tool [1, 3]. In a fault attack, an adversary has physical access to the target de-

vice, and s/he can control the operating conditions of the apparatus. To break the security, the adversary injects well-chosen, engineered faults by actively manipulating the operating conditions during the execution of the target security mechanism. Common low-cost fault injection techniques include clock glitching, voltage glitching, voltage underfeeding, and overheating [1, 3]. These methods create failures by temporarily violating the timing constraints of the target system [20]. Another technique for fault injection is using electromagnetic fault injection (EMFI) technology, which directs powerful electromagnetic pulses towards the device using an EM probe [15]. These pulses create sudden current flow in power/ground networks in the device under attack switching transistor(s) ON (or OFF). Also, EMFI can penetrate through nonmetallic surfaces, allowing an adversary to create local faults without decapsulation of the target device. This significantly reduces the cost and time needed to mount a successful attack. Therefore, it is vital to design efficient detection methods.

Software countermeasures such as algorithm-level, instruction-level fault detection mechanisms are generic solutions against fault attacks. The most straightforward method for the fault detection is running an algorithm twice and comparing the outputs of both executions. Another approach is using error detection codes or parity bits for the critical data of an algorithm and checking them at the end of the algorithm. However, it has been shown that these algorithm-level countermeasures are insecure against multiple fault injections and adaptive adversaries. Defending against fault attacks from software is therefore difficult as the faults do not originate in the software, but rather in the underlying processor hardware. Moreover, modern embedded systems need to satisfy various performance and flexibility re-

quirements. Therefore, current embedded systems require low-cost and flexible mechanisms for fault detection.

In this research, we propose various hardware fault attack mechanisms to detect different types of attacks suited for constrained embedded systems. These detection devices use the most sensitive elements of the design as detectors and monitor their response to the injected fault. The detectors consist of circuit elements that do actual computation. Thus, it can achieve 100% detection rate with no false positives. As the detectors are tightly coupled with the logic, they are hard to bypass. Because of its all-digital implementation, the detection methods can be easily integrated into VLSI design process for both ASIC and FPGA technologies.

The thesis is organized as follows. Chapter 2 gives an overview of common fault injection techniques and how they can be practically applied. Chapter 3 provides an outline of current Fault attack detection technologies. Chapter 4 presents Ring Oscillator based detector for detecting setup time violation attacks. Chapter 5 presents detection method for Electromagnetic Fault Injection Attacks. Chapter 6 ends the thesis with the conclusion of presented work.

Chapter 2

Fault Injection

Hardware Fault Injection (FI) refers to a variety of techniques for inducing errors in the device and measure the response to those mistakes. This chapter introduces the commonly applied fault injection mechanism and fault injection properties. Subsequently, we demonstrate how fault attack can be conducted on supposedly secure function.

2.1 Fault Injection Methods

There are different ways an adversary can introduce errors into the target device. The below list shows a few of prevalent methods for how an attacker can tamper with a processor to induce fault

Clock glitching reduces the clock period of a digital circuit during selected clock cycles. If

the instantaneous clock period decreases below the critical path of the circuit, then a faulty value will be captured in the memory or state of the circuit. An adversary can inject a clock glitch by controlling the clock line of the digital circuit, triggering a fault in the critical path of the circuit. If the adversary knows the circuit structure, s/he will be able to predict the location of the circuit faults. Glitch injection is one of the least complicated methods of fault injection, and therefore it can be considered as a broad threat to secure circuits.

Voltage Starving can be used to artificially lengthen the critical path of a circuit, to a point where it extends beyond the clock period [2]. This method is similar to injection of clock glitches, but it does not offer the same precise control of fault timing.

Voltage Spikes cause an immediate change in the logic threshold levels of the circuit [1]. This changes the logic value held on a bus. Voltage spikes can be used, for example, to mask an instruction read from memory while its moving over the bus. Similar to clock glitches, voltage spikes have a global effect and impact the entire circuit.

Electromagnetic Pulses cause Eddy currents in a chip, leading to erroneous switching and isolated bit faults [19]. By using special probes, EM pulses can be targeted at specific locations of the chip.

Laser and Light Pulses cause transistors on a chip to switch with photo-electric effects [22]. Through focusing of the light, a minute area of the circuit can be targeted, enabling precise control over the location of the fault injection.

Hardware Trojans can be a source of faults as well. This method requires that the

adversary has access to the circuit design flow and that the design is directly modified with suitable trigger/fault circuitry. For example, recent research reports on a FPGA with a backdoor circuit which disables the readback protection of the design [21]. Another example is BlueChip, a processor design that considers the threat of hardware trojans inside of the processor [11].

2.2 Fault Injection Properties

An attacker should understand the impact of the fault and the target algorithm to mount a successful fault attack. Fault attacks can be designed to exploit specific weaknesses of the target algorithm which are introduced by the injection of a fault. Several attacks targeting a large number of algorithms were presented in the past, the most common being the attacks against AES, RSA, and ECC. Besides the algorithm type, it is important to characterize different properties that help distinguish an attack. These properties are listed below:

Fault Attack Timing refers to the time of the injected fault concerning the instructions running on the target hardware. With precise control, it is possible to affect a specific bit or variable in time. An attacker can use this to target a specific operation/instruction in the algorithm.

Fault Attack Location determines the position of the injected fault on the target device. An adversary with full control can modify the specific bit/s of a particular variable, given that s/he knows the physical layout of the device under attack. However, in practice, this is

rarely the case and is possible only to attack a particular variable or range of bits.

Fault Type is decided by the effect of the fault on the device and can be classified into: bit flip, bit set/reset and stuck-at-fault. A bit flip error always sets the target bits to complementary values. On the other hand, the bit set/reset fault sets the target bits to chosen values - one or zero regardless of their previous values. The stuck-at-fault is similar to bit set/reset fault. Nevertheless, in this case, a bit is permanently tied to a particular value.

Size of the affected bits refers to the timing of the injected fault to the instructions running on the target hardware. With precise control, it is possible to affect a specific bit or variable in time. An attacker can use this to target a specific operation/instruction in the algorithm.

2.3 Fault Attack Example

The power of fault attacks can be illustrated with a simple PIN verify program.

```
boolean verifyPin(char* userPIN) {
  charArray correctPIN = {1,2,3,4}
  for (i=0; i<length(correctPIN); i++) {
    if (pin[i] != correctPIN[]){
       reducePinTryCounter()
       return false
```

```
    }
  }
  return true
}
```

Listing 2.1: Pin Verification Code

A Personal Identification Number (PIN) is an alphanumeric passcode used for authenticated access. Listing 2.1 gives the pseudocode of PIN verification function. `verifyPin(char* pin)` function compares the entered pin `userPIN` to a secret value `correctPIN`. If the `userPIN` matches the stored pin `correctPIN`, the function returns true. Otherwise function returns false reporting an incorrect pin and decrementing the pin-try-counter.

The function may seem secure but only if the underlying device executes it correctly. Fault Attacks can modify the data and instruction execution. For the given example, an adversary can successfully complete `verifyPin(char* )` with an *invalid* PIN. An adversary can achieve this outcome in multiple ways. For instance, s/he can skip the `if` statements or calling of the `verifyPin(char* )` function. Alternatively, he can set the `correctPIN` to a known value such as zero by attacking when it is loaded from the memory. The adversary can also invert the result of the `verifyPin(char* )` function while attacking during its execution. Finally, s/he can try all possible PIN values and skip the execution of the `reducePinTryCounter()` with fault injection. All of these can be achieved using glitch injection attacks by manipulating either clock or the operating voltage.

2.4 Conclusion

In this chapter, we provided some background on fault attacks. We showed that various methods could be used to induce faults in a design from clock glitching, to voltage starvation, and to electromagnetic or laser injection. Although different in forms all these methods try to cause errors in the operation of the circuit and use that error to undermine the security of the design. Therefore, it is imperative to be able to detect these attacks. In the next chapter, we will review some of the existing work on detecting fault attacks for timing fault attacks and Electromagnetic Injection fault attacks.

Chapter 3

Related work

Here we will provide an overview of current fault attack detection techniques. We focus on detectors and sensors in the literature designed to detect timing violations and Electromagnetic fault attacks.

3.1 Detecting Electromagnetic Fault Injection Attacks

The Electromagnetic glitch Fault Injection (EMFI) has recently emerged as an effective fault injection method for conducting physical attacks against integrated circuits. Initial research efforts have demonstrated that electromagnetic glitch based failures are induced due to timing violations and that they are also located in the vicinity of the injection probe. In the literature, there are different approaches used to develop a dedicated EMFI detection method, from using a glitch based detector to a Ring Oscillator based sensor, which we

discuss in the following sections.

3.1.1 Delay Based Sensor

Zussa et al. proposed the use of delay-based glitch detectors to detect EMFI[25]. This method detects setup time violations by using fixed guarding delay which is set slightly greater than critical path of the design. The principle of this detection mechanism consists of detecting the breach of a guarding delay prior to any timing violation. The clock signal is used as a reference to draw comparisons between the guarding delay and the clock period. In normal operation, the guarding delay is set greater than the critical time, but smaller than the clock period. If clock period is decreased by inducing a timing violation, it will have to be shorter than the guarding delay. Hence, voltage disturbances will be detected, and an alarm will be issued.

The delay-based glitch detector is only partially successful with a best-case detection rate of about 32%. The second drawback of the detector is that the fixed guarding delay value during the pre-silicon stage might not be the same after the chip is fabricated. Specifically, the PVT variations can cause the guarding delay to deviate from its intended value. This change of guarding delay can cause the detection rate to decrease further or can create false positives.

3.1.2 PLL Based Sensor

Miura et al. introduced PLL-based sensor circuit to detect Electromagnetic Injection reactively [16]. The detection technique uses a Phase Locked Loop (PLL), which is a clock control circuit and is found in modern ASICs and FPGAs. A PLL needs multiple clock cycles before it generates a stable clock, after which it goes into 'locked' state. Whenever the parameters of PLL are changed, the lock is broken and takes multiple clock cycles to come back to the 'locked' state. This work shows that EMFI can affect the clock signal and break the PLL lock. So by monitoring the 'locked' state of PLL, this method determines if the device is under attack or not.

A drawback of this method is that the PLL may not always be available and adding a PLL just for fault detection might not be cost effective. Furthermore, PLL is an analog component sensitive to layout. Therefore, integrating it into a digital circuit remains a challenge. This method adds a lot of cost on the chip area and power consumption.

3.1.3 Full Detector

A method for detecting bit-fault using full detector was proposed by El-Baze et al. [8]. First, they initialize several flip-flops to represent different bits. Then, they monitor the bits at the rising edge of the clock. A fault is detected when the flops take unexpected values, and the alarm is then issued.

This detection mechanism uses 5 flip flops, 6 inverters, 2 XOR gates, and 1 AND gate per

detector block. Moreover, it is not clear how many of these detector blocks are required and where to physically place them on the chip to ensure 100% fault coverage.

3.1.4 Hogge Phase Detector

A novel sensor is introduced in [4] for detecting EM disturbances using Hogge Phase Detector. The sensor identifies if the circuit is under EMFI attack by measuring the variations in an internal ring oscillator (RO). The reported detection rate of this sensor is 93%.

However, this sensor has high false positives rate of about 56%, which may incur a significant performance penalty. Furthermore, using a power-hungry RO limits its applications on IOT and low power devices. It also requires some characterization before it can start working correctly.

3.2 Detecting Setup time Violation Attacks

The setup time violation attacks such as clock glitching, supply voltage variation, voltage underfeeding are the most commonly used physical fault injection methods. Clock glitching is the most practical and low-cost fault injection technique, and therefore clock glitch based DFA is more controllable and becomes a real security threat. Some high-level protection methods for cryptographic systems employ error detection and correction codes or redundancy [24, 13] to improve the reliability of circuits, but cannot resist clock glitch-based

DFAs. Other protection methods try to detect the fault injection sources and therefore prevent DFAs [12]. We discuss these hardware detection techniques in the literature used against setup time violation attacks.

3.2.1 Concurrent Error Detection (CED)

These methods detect faults in parallel with the normal operation of the circuit. They employ time (e.g., repetition), hardware (e.g., duplication), information (e.g., error detection codes), or hybrid (e.g., inverse computation) redundancy for fault detection [12].

The CED methods aim at detecting logical effects of the fault injection (e.g., clock glitching) on the data. Thus, their detection capabilities are independent of the fault injection means. However, they require modifications to the protected circuit. Depending on the performance and security requirements, they can bring large area and timing overhead on the protected circuit. Moreover, Guo et al. showed that an adversary could bypass most of the CED techniques by injecting the same faults into both actual and redundant operations [12].

3.2.2 Error Detection for Variation-Aware Design

In a variation-aware design, the operating frequency is dynamically scaled to increase the performance. The operating frequency is gracefully increased until a timing error occurs. Upon detection of a timing error, the frequency is reduced back to the previous value and circuit operates at this frequency until the next error occurs.

A well-known technique for error detection is using Razor Flip-Flops (FFs) [9]. A Razor FF consists of a regular datapath FF and a shadow latch that is fed by a delayed clock signal. If the regular FF and the shadow latch capture different values, an alarm is raised.

In a system, where the operating voltage is reduced gracefully, or the clock frequency is increased gradually to improve performance, it is expected that Razor FFs will be able to detect the first errors and raise an alarm that will initiate a recovery process. However, in a fault attack, an adversary can inject a clock glitch such that both the regular FF and the shadow latch of a Razor FF capture the incorrect value. In this case, the fault remains undetected. Thus, Razor FFs are not suitable for fault attack detection.

3.2.3 Voltage or Clock Line Monitors

These detection methods monitor the clock or voltage line of the device. They raise an alarm signal in case of an anomaly.

A method for detecting anomalies in the clock signal is using another external clock signal as a reference to monitor system clock line[14]. This reference signal is of higher frequency than the system clock and is used to measure the width of the system clock. The width of the system clock relative to this reference clock is measured in the normal scenario when it is not under attack. Any time this measurement changes, an alarm is raised and the glitch attack is detected.

This is a simple method to detect clock glitches, and the results have shown that the scheme

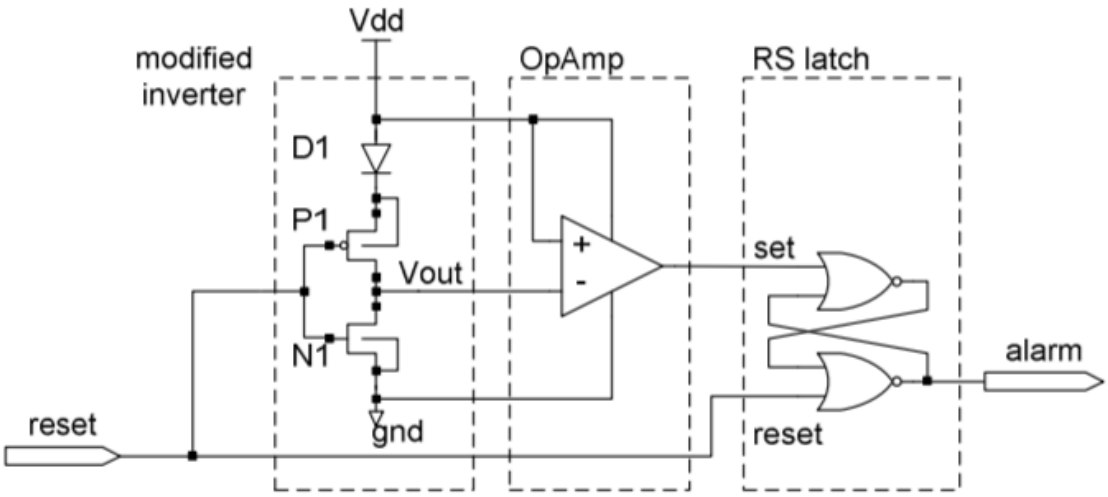

Figure 3.1: Analog Sensor to monitor voltage lines

can detect glitches efficiently with low resource overhead. However, the reference clock signal used to monitor the system clock can be manipulated. The adversary can control both the system and the reference clock to glitch the device yet not triggering the detection method. This is a major drawback of this approach and is only suitable in cases where reference clock lines could be trusted.

Analog sensors are employed to monitor the voltage line of a device [23]. The glitch sensors shown in Figure 3.1 detect the glitches by overseeing the impact of a glitch on the output of the modified inverter circuit. This method is designed to be used to detect positive glitches on Vdd.

However, these sensors are bulky and have high detection latency. They also require a complex analog design process.

3.3 Conclusion

In this chapter, we reviewed state of the art on timing and EMFI fault attack detection. We reported their effectiveness and their perceived weaknesses. In the next chapter, we provide our proposed low-cost cycle-accurate timing fault detector, which can be efficiently employed both in ASIC and FPGA designs.

Chapter 4

Detecting Timing Violation Attacks

In this chapter, we propose a countermeasure that can detect timing violation attacks, which we have previously presented in [6]. The proposed detection mechanism is based on monitoring the incoming clock and making sure it is pacing at a speed that is acceptable to the current state of the hardware. The detection mechanism measures the relative speed of the clock with respect to the hardware and resolves whether the design is under attack or not.

We understand the speed of the hardware depends on many variables including operating voltage and environmental temperature. Attackers can use these variables to inject faults. They can also directly inject faults by manipulating the clock. One can also imagine a combination of these methods. Regardless of the source, these faults happen because the design clock has become too fast for the circuit to operate. The detector measures the relative speed of the clock with respect to the hardware and resolves whether the design is under

attack or not. In an analogy, it works similar to a medical thermometer. Just like a medical thermometer can detect the presence of fever, but it cannot identify the underlying issues, our proposed detector can capture the presence of danger, but cannot isolate the cause (does not need to). The sensor can detect several forms of attacks, namely under-powering, clock glitching, or a combination of these two attacks.

4.1 Proposed Detection Mechanism

The proposed scheme detects if there is an anomaly in the operating conditions of the circuit and raises the alarm. The principle of this work is to detect fault attacks on the external clock by monitoring it using a Ring Oscillator (RO). RO clock is itself immune to glitches, direct change in frequency and clock manipulation by faults as it is internal to the device. This RO clock is then used as a reference to draw a comparison between the external clock and itself, efficiently detecting a change in the external clock.

4.1.1 Operation Overview

The basic idea of the proposed detector is to sample the external design clock (clk_d) with a faster internal Ring Oscillator clock (clk_{ro}) as illustrated in Fig. 4.1. The expected relative frequency difference between clk_d and clk_{ro} is stored in a secure non-volatile memory, which is a standard component in many applications. When an attacker tries to over-clock a design or underfeed it by lowering the operating voltage of the device (V_{DD}) for inducing a timing

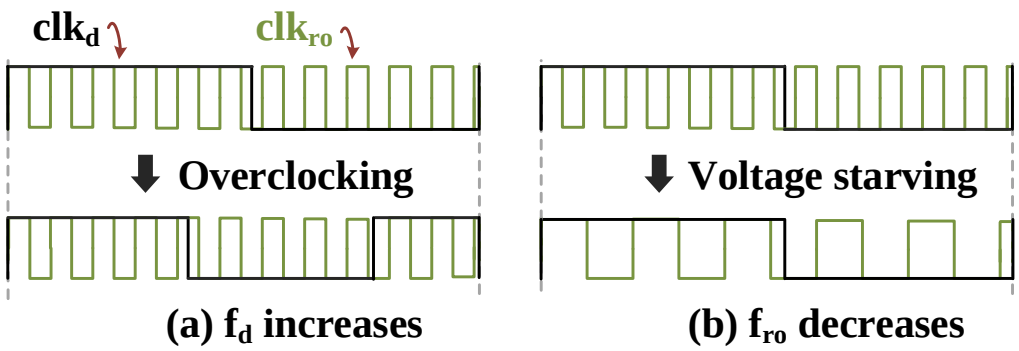

Figure 4.1: Detector Principle for (a) Over-clocking. (b) Voltage starving.

fault, the relative frequency difference deviates from the expected value, and an *alarm* is issued. Hence, by continuously monitoring the design clock every cycle, the detector can issue an *alarm*, *insitu*, to prevent an attack.

Fig. 4.1 illustrates the behavior of the detection technique under over-clocking and under-feeding. If the adversary tries to induce fault by increasing the clock frequency, the relative frequency difference decreases as shown in Fig. 4.1a. This change thus triggers an alarm notifying the user of timing violation. In case the circuit is attacked by reducing the supply voltage, the propagation delay of the circuit and the detector increases. This decreases the RO frequency and reduces the relative difference as depicted in Fig. 4.1b.

To monitor the design clock, we use two counters; one for each phase of clk_d. Although only a single counter is sufficient to sample the period of the design clock, we use two counters as it has distinct advantages. First, in attack scenarios like over-clocking, *alarm* can be issued earlier than in case of using single counter. This is necessary for sensitive designs which require immediate activation of response strategy after fault detection. Further, using

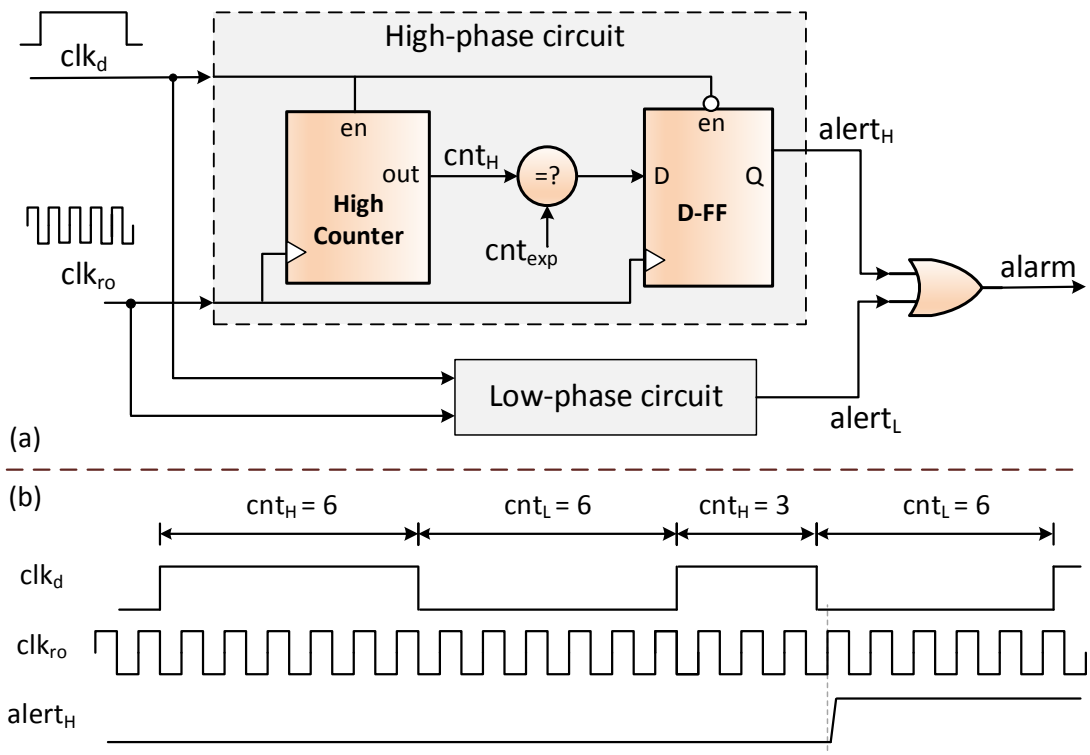

Figure 4.2: Timing Detector: (a) Block diagram. (b) Timing diagram

two counters allows for latching the comparator output and resetting the counter without borrowing time from the next clock cycle. Next, we present the architecture and the detailed operation of the detector.

4.1.2 Circuit Architecture

Fig. 4.2a shows the block diagram of the operation of the detector for the high phase of clk_d. The operation for the low phase is similar to that of the high phase, and it is not shown. As shown in Fig. 4.2a, the counter is enabled for the high phase of clk_d, and it increments on the

positive edge of clk_{ro}. The output of the counter (cnt_H) is compared against a predetermined expected threshold (cnt_{exp}) to generate the comparator output. However, the comparator output is latched to $alert_H$ only when the counter has stabilized after the high phase of clk_d. Hence, when $alert_H$ is asserted, it signals that an anomaly occurred during the high phase of clk_d. Similar to $alert_H$, $alert_L$ signals if an anomaly has occurred during the low phase of clk_d. A positive edge of either $alert_H$ or $alert_L$ issues an $alarm$, which can be acted upon to suspend the circuit operation, update the secret key, etc. From Fig. 4.2a, it can be observed that the comparator output is latched to $alert_H$ only on the $first\ valid\ clk_{ro}$ cycle during the low phase of clk_d. After $alert_H$ is latched, the high counter is cleared to zero on the next rising edge of clk_{ro}.

Fig. 4.2b shows the operation of the detector with the help of a timing diagram. The expected counter value cnt_{exp} for target clk_d and V_{DD} is computed to be 6. We can observe that the counter output, cnt_H, increments during the high phase of clk_d and stabilizes to value of 6 after the first high cycle. Since this counter value is the same as expected value, this cycle does not trigger an alarm. However, at the end of the second high cycle, the counter value cnt_H assumes the value of 3. Hence, the comparator output stays high as the counter value is less than the expected threshold. The first $valid\ clk_{ro}$ cycle in the following low phase of clk_d latches the comparator output to $alert_H$ which then triggers an $alarm$.

4.1.3 Timing analysis of the detector

In Fig. 4.2b, when the rising edges of clk_d and clk_{ro} are close to one another, it can result in a timing violation in the counter. To limit the effect of this violation, we use *asynchronous (ripple) counters* such that, in the case of a violation at either the rising edge of clk_d or the falling edge of clk_d, it only affects the least significant bit (LSB) of the counter. Hence, in the event of a violation, the counter misses the count by 1, in the worst case. The detector can be designed to tolerate this error by accommodating this error in setting the threshold for comparison (cnt_{exp}).

This design accommodation involves detector to require 2 clock cycles on each phase of clk_d. This enforces a lower bound on the relative frequency of the ring oscillator (f_{ro}). Additionally, our design choice involves using a single counter to monitor each phase of clk_d. Considering these constraints, the frequency of RO, f_{ro}, has to be at least four times the frequency of the design clock (f_d). This is easily achievable considering the target systems are secure embedded designs.

4.2 Detector Parameters

In this section, we define a new metric called **Leniency Factor** which captures the required resolution to distinguish acceptable safe variations from unacceptable attack behavior. Leniency Factor is defined as the difference between the delay of design when working in safe

Figure 4.3: Attack and safe operation windows

conditions compared to its delay when the first fault happens. This metric is also a function of area and power allowing a designer to trade-off between low area or low power while maintaining the required minimum resolution. In the following section, we analyze different attack scenarios and describe how to determine the parameters that affect both cost and effectiveness of the detector while mathematically formulating LF.

The most important parameter of the detector is the frequency of the internal ring oscillator. In other words, a fast enough RO provides sufficient resolution to discern faults from normal variations in environment but consumes more power. A slower RO, on the other hand, may not be able to identify an acceptable change in operating voltage from an attack and result in a false positive alarm. Nevertheless, such slow RO is going to consume less power. We formalize the above statements by calculating a lower bound for the frequency of the RO. First, we observe that for the detector to be able to distinguish safe operation from attack, the difference between the counter values for the two scenarios must be greater than or equal to one. This is shown in (4.1), which can be rewritten as (4.2).

$$f_{ro} \times t_{safe} - f_{ro} \times t_{attack} >= 1 \tag{4.1}$$

$$f_{ro} \times t_{safe} >= \frac{1}{1 - \frac{t_{attack}}{t_{safe}}} \tag{4.2}$$

In the above equations, f_{ro} is the frequency of the RO. t_{safe} is the period of the fastest clock, where the design can safely operate. t_{attack} is the period of the slowest clock that results in faults and therefore, constitutes a real threat. While t_{attack} is the property of the circuit, t_{safe} is designated by the designer and is greater than or equal to t_{attack}. Fig. 4.3 shows the relationship between t_{attack} and t_{safe}.

Equation 4.3 formulates Leniency Factor which captures the acceptable variations from irregular. The minimum leniency of a design is 0, while at its maximum, it can reach 1. We can rewrite (4.2) as shown in (4.4),

$$LF = 1 - \frac{t_{attack}}{t_{safe}} \tag{4.3}$$

$$\frac{f_{ro}}{f_{safe}} >= \frac{1}{LF} \tag{4.4}$$

This means the relative frequency of the RO with respect to a safe frequency of the design is inversely proportional to LF. In other words, if the designer decides on very low leniency by choosing t_{safe} very close to t_{attack}, which requires fast RO thus dissipating more power. On the other hand, if t_{safe} is chosen much larger than t_{attack}, leniency will be higher, and frequency of RO can be lower. The relative speed of the RO should never be less than 4, required for ensuring correct operation of the detector as described in Section 4.1.3.

The above equations describe how f_{ro} can be selected at nominal voltage, when the attacker is manipulating the clock. We continue our analysis by describing the scenario, where the

voltage is under attack. Equation 4.1 can be written as follows:

$$f_{ro}(V) \times t_{safe}(V) - f_{ro}(V) \times t_{attack}(V) >= 1 \tag{4.5}$$

This means when manipulating operating voltage, V, all three parameters in the equation become functions of V. The question is how to choose f_{ro} at nominal voltage such that it can still identify attack scenarios at lower voltages. In order to answer this question, we resort to modeling the delay of the circuit as well as RO with respect to operating voltage. We know that in superthreshold region frequency increases almost linearly with voltage. The slope of the increase depends on circuit characteristics, nevertheless with an error margin of less than 10%, we can assume both the circuit under attack and the RO experience the same relative slowdown. In other words, we can write:

$$f_{ro}(V) = f_{ro}(V_{nominal}) \times a \times (V - V_{threshold}) \tag{4.6}$$

$$f_{attack}(V) = f_{attack}(V_{nominal}) \times a \times (V - V_{threshold}) \tag{4.7}$$

Where a is a constant value. In our SPICE simulation, the error resulting from this model for voltages up to $0.9V$ is less than 6%.

If we rewrite (4.5), since all the voltage-dependent terms cancel out, we are back to (4.1), and the same bound for RO frequency holds for the underfeeding attack. Even if the first faults happen at near or subthreshold regions, despite the fact that the relationship is not linear anymore, it is still expected that voltage-dependent terms in (4.5) to cancel each other out. In other words, the RO experiences the same relative slow down as the rest of the circuit.

This enables the designer to choose a LF at nominal voltage and decide on a frequency for the RO without extensive analysis at lower voltages.

$$f_{ro} >= f_{safe} \times Max(4, \frac{1}{LF}) \tag{4.8}$$

To summarize, the designer can use (4.8) to choose a frequency for RO. From (4.8), both f_{safe} and LF are chosen by the designer. The choice of LF depends on what the designer deems as noise in the system compared to an attack. t_{safe} shown in Fig. 4.3 is the shortest delay, while the design is operating safely. For example, let us assume that t_{safe} for a device at nominal operating conditions is $100ns$ and -10% V_{DD} and 50°C is the upper limit of noise in the system. At these extreme conditions, let us assume that t_{safe} increases to $150ns$. If t_{attack} for such a design is $75ns$, using (4.3), LF becomes 0.5, whereas, for a design at nominal conditions LF is 0.25. The designer has to select f_{ro} based on LF of 0.5 to accommodate for extreme variations that he/she deems as noise.

4.3 Results

We validated the functionality of the proposed detector on a Xilinx Spartan-6 FPGA. We also synthesized our detector onto a 90nm ASIC technology to examine the effects of process variation and aging on the operation of the detector. Next, we will explain the details of our experimental work.

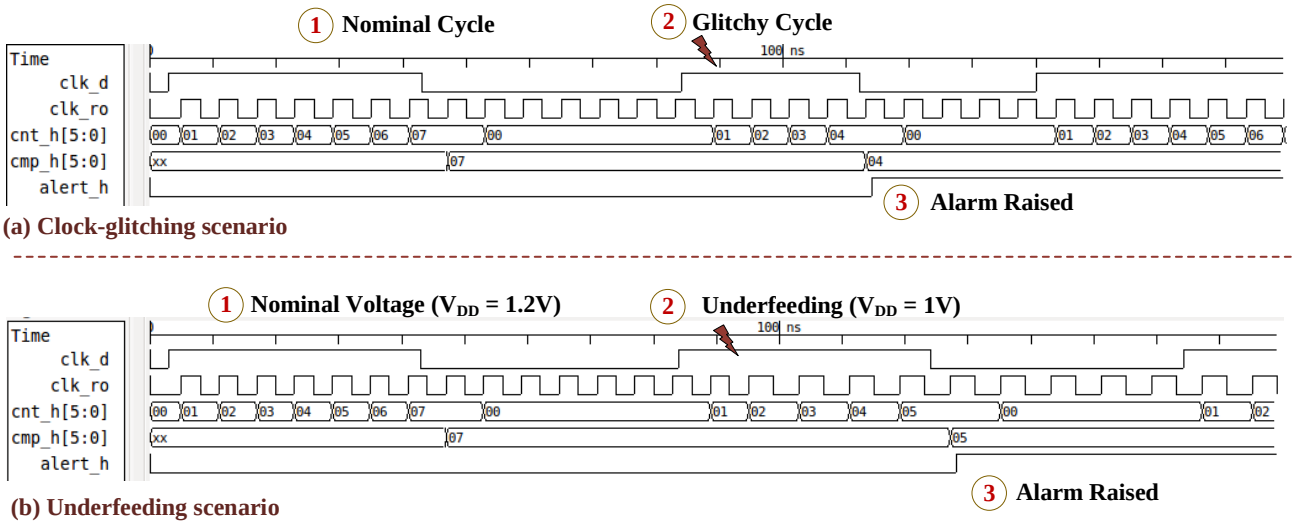

Figure 4.4: Experimental results for (a) Overclocking (b) Underfeeding : (1) Nominal cycle in which case cnt_H counts till 7 (2) Attacker induces faults by shortening the high phase of clock as in case (a) or underfeeding to 1V in case (b) (3) The cnt_H in this cycle assumes a different value raising an alarm

4.3.1 FPGA Implementation

The experimental setup (Fig. 4.5) consists of a SAKURA-G board, a pulse generator (Agilent 81110A), and a computer. On the main FPGA (Spartan XC6SLX75) of the SAKURA-G board, we implemented an AES-128 circuit and our detector. We can adjust the supply voltage of the main FPGA to be 0.5–1.5V by using an onboard trimmer. The clock signal of the main FPGA is fed by a clock glitcher module implemented on the control FPGA (Spartan XC6SLX9). To generate a glitchy clock, the clock glitcher processes a glitch-free clock signal that comes from the pulse generator [5]. It can cause glitches with a resolution of 100ps. The computer communicates with onboard components with a USB interface.

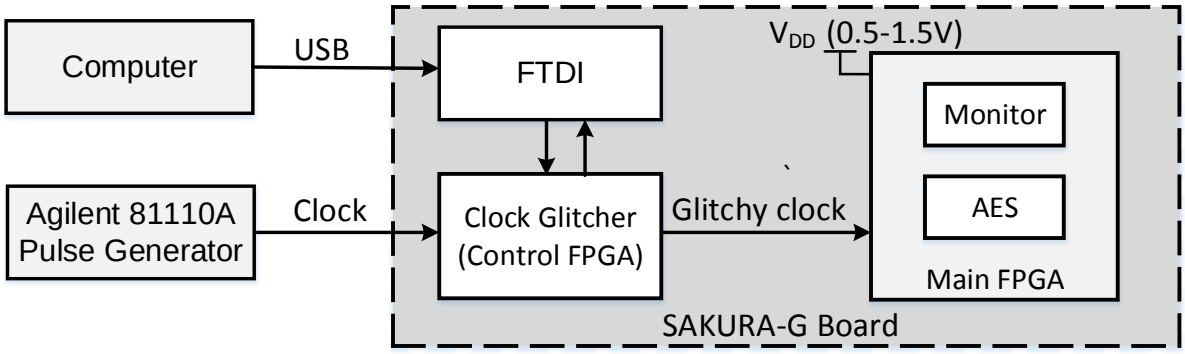

Figure 4.5: Block diagram of the experimental setup

We determined parameters of the detector by following the methodology explained in Section III. First, we measured t_{attack} value of the AES circuit as $38ns$. We obtained this value by injecting clock glitches while the circuit was processing a set of random inputs. Second, we chose a t_{safe} value of $40ns$. This corresponds to an f_{safe} value of $25MHz$. Then, we calculated the Leniency Factor (LF), which is 0.05, using (4.3). Finally, we chose an f_{ro} value of $200MHz$, which satisfies (4.4). Therefore, we set cnt_{exp} to 7.

Fig. 4.4a shows operation of the detector in case of clock glitching. In this figure, a nominal cycle followed by a glitchy cycle, where the high phase of the clk_d is shortened to $25ns$. For the nominal cycle, the cnt_H counts up to 7. However, cnt_H only reaches 4 for the glitchy cycle. As we set cnt_{exp} to 7, $alert_H$ is asserted, which results in issuing an $alarm$. Fig. 4.4b shows operation of the detector in case of underfeeding. We reduced the V_{DD} level from the nominal value of $1.2V$ to $1V$. As seen, this reduces the frequency of ring oscillator (f_{ro}) and, the cnt_H counts up to 5 instead of its nominal value 7. Thus, the detector asserts $alert_H$,

Table 4.1: Area and Power Overhead of the Proposed Detector

Circuit	***Area*** (μm^2)	***Power (mW)***
AES	28321	35.6
AES+Detector	28386	36.1

which then issues an *alarm*. Next, we present our analysis on an ASIC implementation.

4.3.2 ASIC Implementation

We synthesized the AES circuit with our detector onto IBM 90nm technology to compare its cost with the existing methods and analyze the effects of variation on the detector's operation. We used Synopsys tools for synthesis (DC), logic simulation (VCS), SPICE simulation (HSPICE), and reliability analysis (MOSRA). We organized results into four categories.

Overhead of the Detector: Table 4.1 shows post-synthesis area and power consumption results for *(i)* original AES *(ii)* original AES with the timing detector. As seen, our timing detector incurs 0.23% area and 1.4% power-consumption overhead. Also, it does not bring any timing overhead. In comparison, Igarashi et al. [12] and Zussa et al. [25] report 0.47% and 0.3% area overhead for their delay-based monitors. On the other hand, the CED techniques have up to 100% area and timing overhead [10]. As a result, the overhead of our detector is slightly better than the existing delay-based methods and significantly better than the CED techniques.

Effect of Leniency Factor (LF) A designer must consider different tradeoffs during the selection of Leniency Factor (LF). Higher LF values imply a slower f_{ro}, a higher number of inverters, and a smaller number of counter flip-flops. Therefore, each different LF value effects area and power consumption of the detector in a different way.

Fig. 4.6 shows variation in the power and area of the detector for different leniency factor (LF) values. For this simulation, we assume t_{attack} for a design to be $160ns$. The t_{safe} has been varied from $161ns$ to $200ns$ in steps of $5ns$. Then we computed the minimum value of f_{ro} at each step using (4.3) and (4.4). The area numbers reported are the post-synthesis total cell area, whereas, the power consumption numbers are the average power consumption collected from transistor-level SPICE simulations. For very low LF values, Fig. 4.6 shows that the power consumption is high due to large counter size and toggling in counter bits. For higher LF values, however, the area of the inverters becomes dominant. The total area and power consumption saturate as f_{ro} reaches the minimum design limit of 4 times frequency of design clock (f_d).

Effect of Process Variation: Process variation (PV) has a significant effect on the operation of timing detector. The frequency of ring oscillator (f_{ro}) can change from its predefined, intended value due to process variation. This can result in false alarms. To observe the effect of PV, we simulate the timing detector at typical, slow, and fast process corners. For a ring oscillator (RO) designed for $200MHz$ at typical corner, we observe that f_{ro} varies from $165MHz$ at slow corner to $246MHz$ at fast corner. As mentioned earlier cnt_{exp} is the product of f_{ro} and t_{safe}. As f_{ro} might deviate from expected value due to PV, we propose

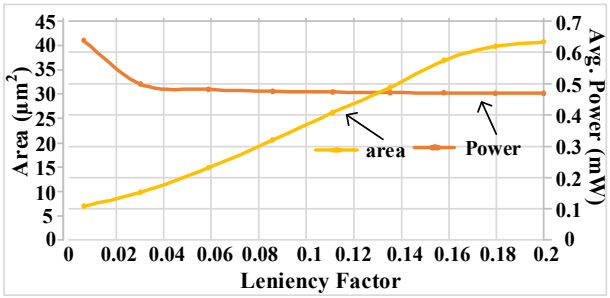

Figure 4.6: Power and area tradeoff with leniency factor

characterization of f_{ro} after fabrication. Based on post-fabrication f_{ro}, the actual value of cnt_{exp} is evaluated and stored in the non-volatile memory.

Having this flexibility of tuning the detector with respect to the requirements is an advantage of our detector over delay-based countermeasures.

Effect of Aging: Aging is a gradual phenomenon that slows down circuits with time. To simulate the effect of aging on the ring oscillator (RO), we design an RO with a nominal f_{ro} of $200MHz$. After operating the RO for 10 years, the value of f_{ro} drops to $193MHz$. Similar to the effect of process variation, the value of cnt_{exp} can be adjusted to accommodate changes in f_{ro} due to aging. To further mitigate the effect of aging, the RO can be composed of NAND gates with common enable input. This allows designers to reduce the impact of aging by disabling the detector when it is not needed.

4.4 Conclusion

In this chapter, we presented a cycle-accurate monitor that could efficiently detect timing-violation-based fault attacks. The proposed monitor could detect clock or voltage manipulations by monitoring the external clock using an internal Ring Oscillator. The monitor was low-cost concerning area and power consumption and could be easily implemented using the standard cell based VLSI design flow. In addition to the architecture of the timing monitor, we presented a detailed analysis of the design considerations that affect the cost and accuracy of the detector.

In the next chapter, we will present a different fault attack detector, which targets Electromagnetic Fault Injections. We will provide the architecture of the detector as well as an analysis of its cost and effectiveness.

Chapter 5

Detecting Electromagnetic Fault Injection Attacks

In this chapter, we propose a novel sensing technique that can detect EMFI, which we have previously presented in [7]. The technique relies on the observation that flip-flops are the most sensitive elements to EMFI and the effect of EMFI is based on the polarity of the applied EM pulse [18, 17]. The central idea of the detection is to monitor the vulnerable flip-flops for faults by pairing each of these flip-flops with a shadow flip-flop, which is designed in such a way that only one flip-flop of the pair gets affected by EMFI. The proposed sensor then measures the relative states of the main and shadow flip-flops, and resolves whether the design is under attack or not.

As opposed to the previous detectors, this EMFI detector uses the circuit elements that do

actual computation. Thus, it can achieve 100% detection rate with no false positives. As the detector is tightly coupled with the logic, it is hard-to-bypass. Because of its all-digital implementation, this sensor is easily integrated into VLSI design process for both ASIC and FPGA technologies. The sensor is lightweight as it incurs cost of integrating a shadow element into the logic. In addition, existing scan flip-flops can be used as shadow flip-flops to reduce the cost more.

5.1 Operation Overview

The proposed detector consists of an existing main flip-flop (MFF) and an additional sensing flip-flop called a shadow flip-flop (SFF) as shown in the Figure 5.1. The detection circuit is designed in such a way that nominally, the two flops capture complementary values at every clock cycle. So at a particular cycle, if the output of MFF (Q) is at logic-0, the output of SFF (Qs) will be at logic-1. In the case of EMFI, either the main flop (MFF) or the shadow flop (SFF) gets affected based on the polarity of the EM pulse. The injected fault is then detected by applying an XNOR operation on the flip-flop outputs Q and Qs. The area cost of the standalone detector is an additional FF, NOT and an $XNOR$ gate compared to a regular FF. It is possible to utilize design for test (scan-chain design) reuse methodology to reduce the area overhead of the monitor. However, we limit this article to monitor's use to detect EM fault attack.

Figure 5.2 illustrates the behavior of the sensor in a fault-free case and under different fault

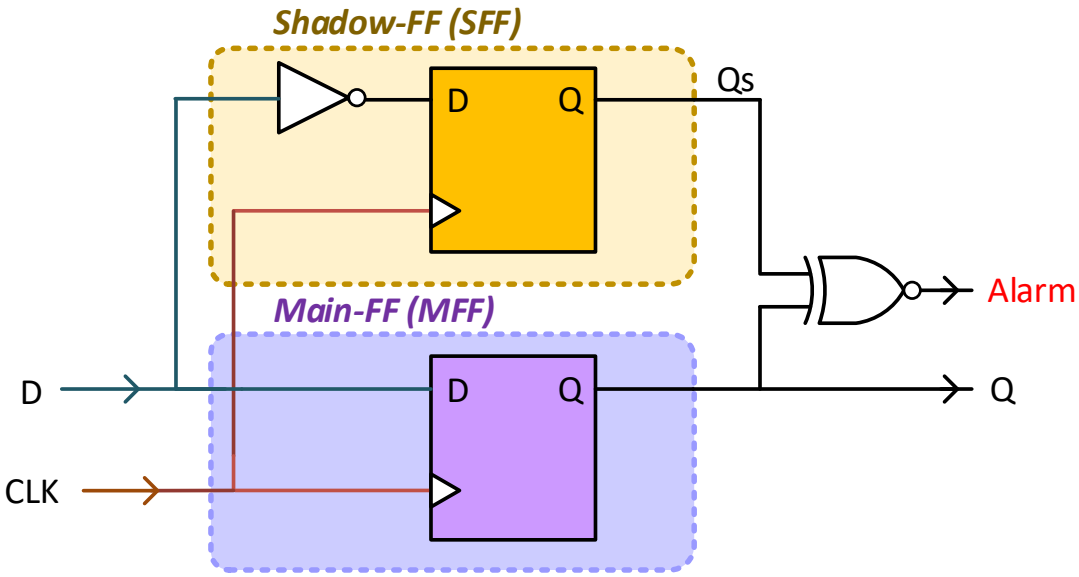

Figure 5.1: Block diagram of EMFI detector: During normal operation, MFF and SFF keep complementary values. EMFI can alter only one of the FFs. An alarm signal is generated if both FFs has the same value.

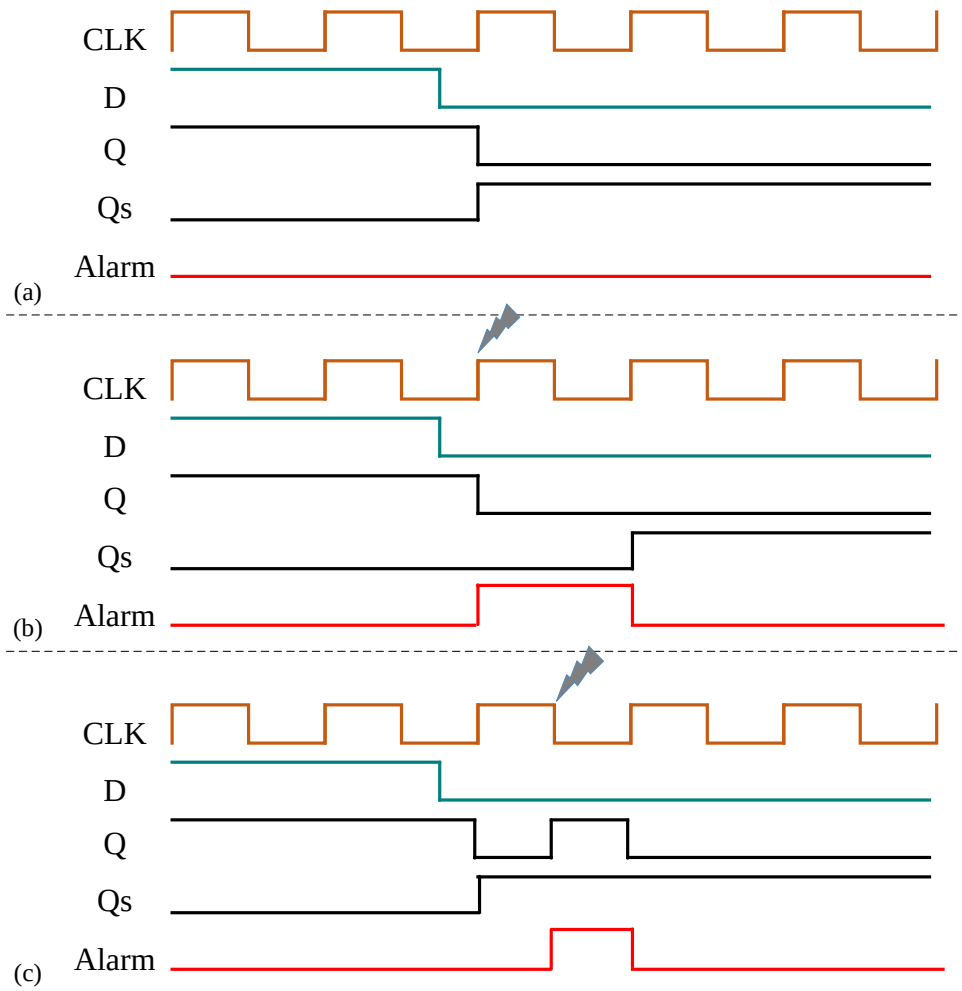

Figure 5.2: Timing diagram of the proposed detector: (a) Fault-free operation. (b) Fault injection case 1. (c) Fault injection case 2.

injection scenarios. It shows how the redundancy in the shadow flip-flop can be used for EMFI detection. Figure 5.2a depicts the first case in which the device is not under EMFI attack. We can observe that the flop outputs consistently assume opposite values at every clock cycle. For example, in the third clock cycle, the main flop output Q is at logic-0 and the shadow flop output Qs is at logic-1. Since the outputs of the registers are complementary, the alarm signal remains low.

Figure 5.2b shows the second case where the adversary tries to inject the fault during the the sampling window [17] of the flip-flops. In this scenario, the clock is switching, i.e. the fault occurs around the rising edge of the clock, maximizing the occurrence of a fault in the IC. We can see that until the point of attack, the behavior of monitor is same as in the first case, that is, flop outputs Q and Qs remain complementary to each other keeping alarm signal low. However, the fault in the third clock cycle causes one of the flops to change its value. In this case, the EMFI induces bit-reset faults in both the flops. Since Q was already at logic-0, it stays at logic-0. However, Qs value changes to logic-1 from logic-0 instantaneously. The alarm signal immediately goes high which can be used to trigger fault response of the device on the next clock edge.

In the third case, EMFI happens outside the sampling window as shown in Figure 5.2c. In this scenario, the fault is injected when the clock is not switching. As observed, the fault induces bit-set faults in the IC during the third clock cycle. This change switches the main flop output Q to switch to logic-1. The relative indifference in the logic outputs immediately gets detected triggering an alarm.

5.2 Implementation Details

In order to evaluate the effectiveness of the proposed mechanism, we integrate the detector with a target circuit. The detector along with the target logic was implemented on a Terasic Technologies DE0-Nano development kit. In the following section, we first discuss the standalone implementation of the detector on FPGA. Following that, we present the steps performed to integrate the detection mechanism with the logic and map it to the chip. Subsequently, we discuss the measurement system used for validating the faults.

5.2.1 Implementation of Detector

We implemented our EMFI detector on an Altera Cyclone-IV FPGA residing on the DE0-Nano board. In this family of FPGA, Logic elements (LE) are the smallest unit of logic block. Each LE consists of a 4-input lookup table (4-LUT) for implementing logic gates, and a register.

A macro(mon) was created for the detector to integrate it into the target circuit. For this purpose, we first synthesize the detector and map it into the FPGA. Then we place and route the design and the netlist is then extracted using the netlist writer. As it is seen from the EDA netlist writer, the detector macro takes up 2 FFs and 1 combinational primitive.

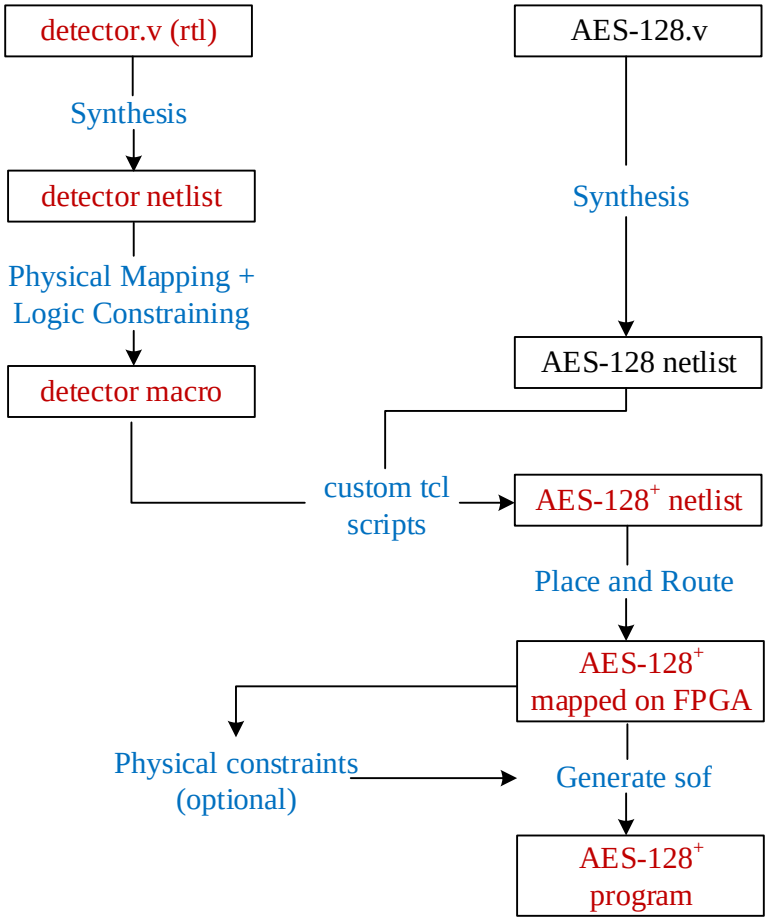

Figure 5.3: Flow chart to integrate the detector with the AES-128 coprocessor

5.2.2 Integration of Detector

To experimentally verify the operation of the EMFI detector, we use a hardware implementation of Advanced Encryption Standard (AES-128) as the target logic. We integrate the EMFI detector into the target logic by replacing all flip-flops of the target logic with our EMFI detector macro. We automated this process using Quartus-II 15.0 software from Altera and our custom Tcl scripts as shown in Figure 5.3. The detailed procedure of integrating the monitor is given below:

1. We synthesize the AES-128 logic and place&route the synthesized design. Then we extract the netlist using the Quartus netlist writer.

2. We also apply the synthesis, place&route, and netlist extraction steps for the EMFI detector.

3. We convert the extracted netlist of the EMFI detector into a macro as mentioned in the earlier subsection.

4. We replace the flip-flops of the original AES-128 netlist by the EMFI detector macros. For this purpose, we use a custom Tcl script which assigns 2 LE blocks per macro. The final netlist has EMFI detectors integrated within the logic. We call the final netlist *AES-128*$^{+}$.

5. Next, we place&route *AES-128*$^{+}$ netlist again.

6. Finally, we generate sof bitstream file to program the FPGA with the implemented circuit.

During *Step 5*, optionally, additional physical constraints can be applied to control the locations of the EMFI detectors. In this work, we applied additional constraints to refrain the the tool from placing the *AES-128*$^+$ circuit near the communication interface. This enables us to inject EM faults into the target circuit without affecting the communication interface. To apply the additional constraints, we use Altera's LogicLock feature. Figure 5.4 demonstrates the final design after LogicLock. We configure the Logiclock region to keep all the *AES-128*$^+$ circuit out of the black rectangle, which is a JTAG module to provide communication between the target logic and the outside world.

5.2.3 Communication Interface

In addition to the *AES-128*$^+$ circuit, we also implemented a communication interface to control the input/output of the AES circuit and observe the status of the EMFI detectors from a computer. Figure 5.5 shows the block diagram of the communication interface consisting of an Altera Avalon Bus fabric, a memory-mapped slave (*AES-128*$^+$), and a memory-mapped bus master (*JTAG to MM Master*). The computer can communicate with the AES circuit using Tcl scripts through Altera's *system-console* application.

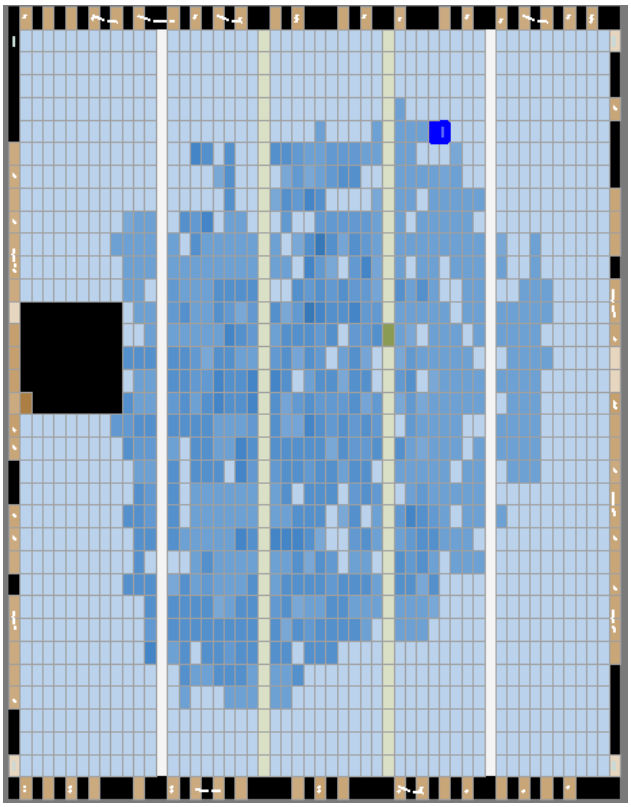

Figure 5.4: *AES-128*$^{+}$ circuit on the Altera Cyclone IV fabric after place&route. Dark blue cells correspond to LEs occupied by *AES-128*$^{+}$ logic, light blue cells are unmapped and black rectangle shows the dedicated JTAG core

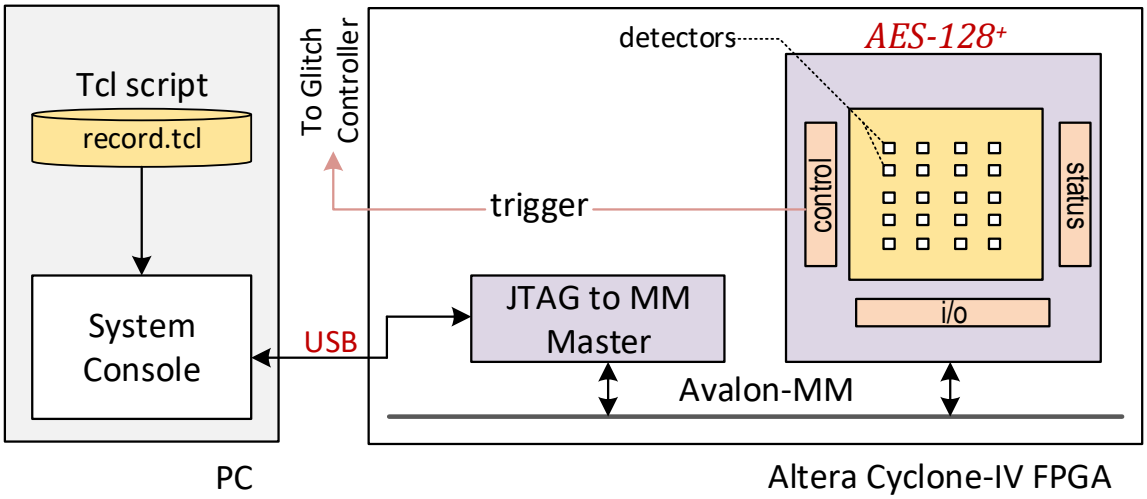

Figure 5.5: Communication interface of *AES-128*$^{+}$ circuit: A computer uses Tcl scripts and system-console to communicate with the AES circuit (a memory-mapped slave) through a JTAG bus master on the Avalon Bus.

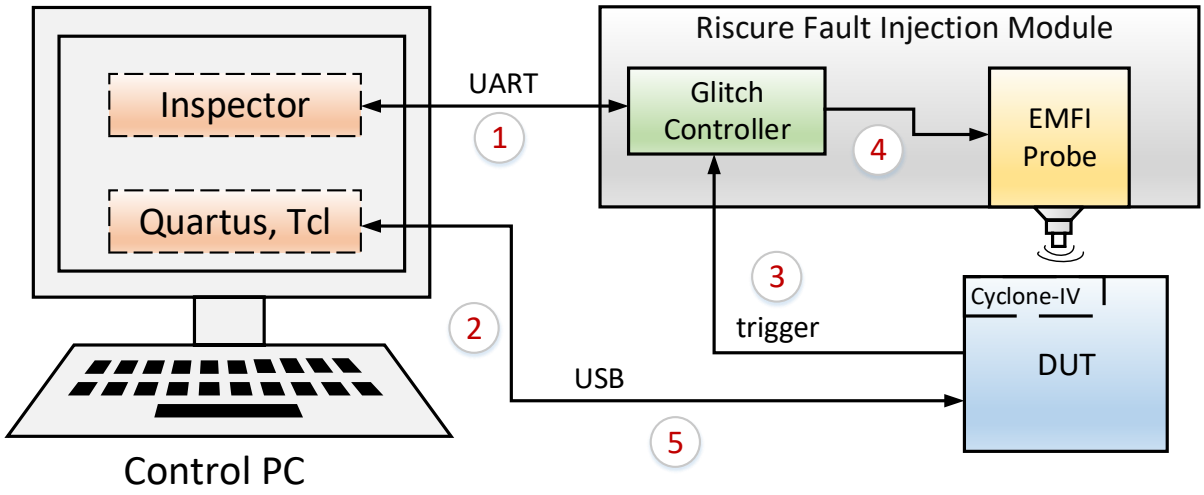

Figure 5.6: Block diagram of Fault Injection Setup. (1) Inspector transmits EMFI parameters to the Glitch Controller and arms it. (2) PC sends input to *AES-128*$^+$ and start encryption. (3) FPGA sends a *trigger* to the Glitch Controller. (4) Glitch Controller makes EMFI-probe to inject EM pulse (5) The output is sent back to PC.

5.3 Experimental Results

In this section, we provide an outline of our fault injection setup and the test procedure that enables us to verify the correctness of the detector and to establish the results.

5.3.1 Fault Injection Setup

Figure 5.6 gives an overview of the setup. It consists of a Control PC, Cyclone-IV FPGA, and Riscure Fault Injection module. The Control PC manages the entire injection process by controlling the fault injection module and the FPGA. The FPGA runs the *AES-128*$^+$ core

and communicates the results back to the PC using the communication interface described earlier. The FPGA also generates a *trigger* for the injection module a clock cycle before the attack.

The Fault Injection module uses two different components, Glitch Controller and EMFI probe. Glitch Controller enables the communication between the PC (Riscure's inspector software), FPGA, and EMFI probe. The Glitch Controller is realized using VCGlitcher module from Riscure. It receives *trigger* signal from the FPGA and generates EMFI parameters required by the probe. The probe creates a voltage pulse that sends current through the coil to produce magnetic field. This changing magnetic field causes voltage perturbation in the device producing EM glitch.

5.3.2 Finding EMFI Parameters

The first step of our experimental work is characterizing the target system to find the right EMFI parameters. These parameters include power of the EM pulse, duration of the EM pulse, location of the probe tip, and the distance between the FPGA surface and the probe tip.

For this particular experiment, we use a probe tip with positive polarity, which generates $+200V$ perturbation at maximum power. The range of the power is from 0% to 100%. Glitch offset refers to the timing of EMFI relative to the rising edge of the clock signal, and takes values between $0 - 20ns$. During our experiments, FPGA operates with a 50-MHz clock

signal and a 1.2-Volt supply voltage.

To determine the best glitch offset and EM pulse power, we conveyed a systematic search. We divided the chip area into a grid of 10 x 10 locations. Then we injected 200 EM faults at each location of the grid by randomly varying EM power and glitch offset parameters within their range. we used a constant EM pulse duration of 50 ns.

Figure 5.7 shows the effect of EM pulses with varying power and glitch offset. For any given offset, we observe three distinct behavior of the circuit, (a) faults (b) fault-free (c) resets/mutes. First, we notice that fault occurrence window is much larger when glitch offset is around 0 or 20ns. At these offset points, a fault can be induced even when the power is as low as around 35%. This can be attributed to the fact that EM susceptibility of a circuit is higher during the rising edge of the clock, which for this circuit corresponds to glitch offset of 0 or 20 ns. Second, we observe that fewer faults can be induced when glitch offset is around 10ns. The power required for such faults is also much higher. This is because the clock is stable and higher energy is needed to induce faults. We use this fine-tuned parameter data and inject repeated faults to establish the results.

5.3.3 Results

We experimentally verified the EMFI detector and measured its efficiency by comparing the number of injections that lead to a faulty output and the number of injections that were detected.

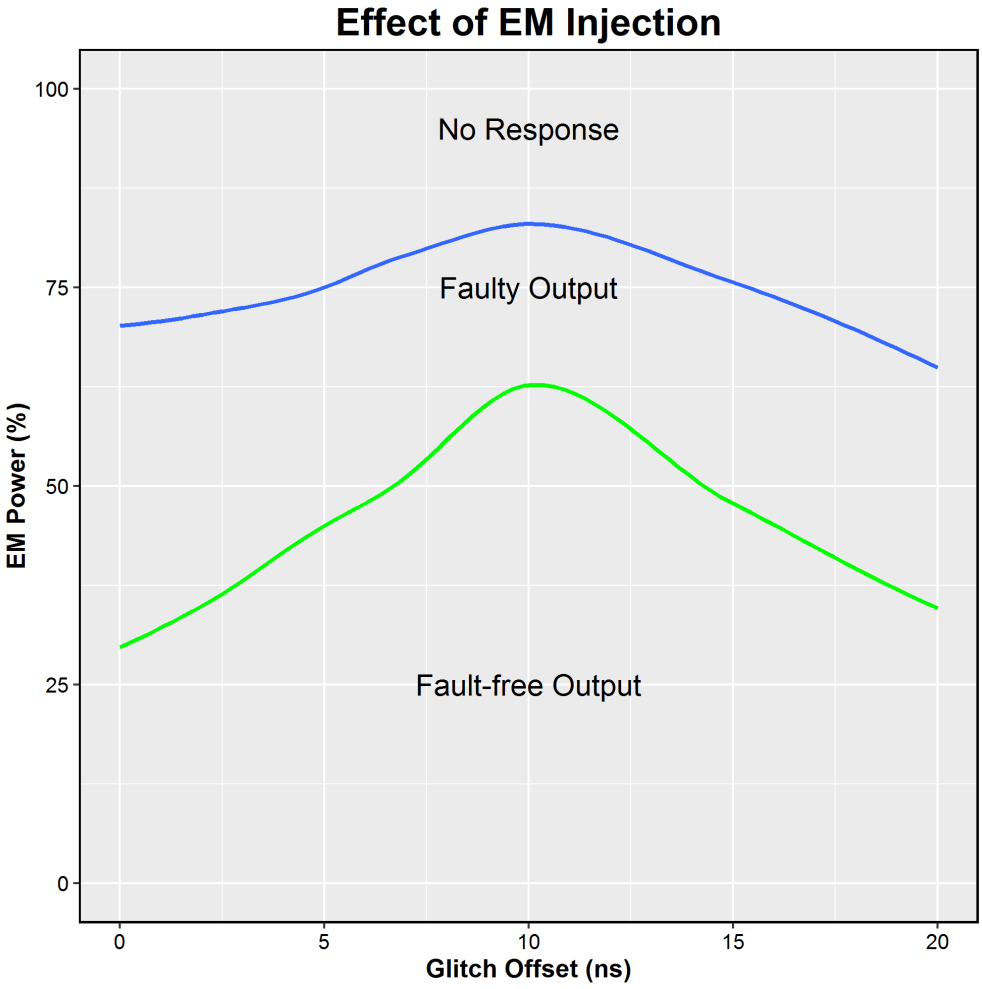

Figure 5.7: Effect of EMFI on the target circuit. There are three distinct regions: (a) fault-free output (under the green line), (b) faulty output (between green and blue lines), (c) no response (above the blue line).

Table 5.1: Hardware Overhead of the detection mechanism on AES-128

	LUTs	**Registers**	**Timing (ns)**
AES-128	4814	530	9.22
AES-128 with detector	5097	620	9.22

Figure 5.8 shows the result of EM injections on the AES-128 logic. We can observe the number of injections at each of location that result in an incorrect cipher output. Out of the 20000 EMFI attacks, only 7% injections lead to a fault. Some of these locations are more vulnerable to EMFI attacks and can produce repeated faulty outputs. We also notice that fault injection experiments at all the locations do not induce a fault in the target logic. The heatmap for the number of detected faults and injected faults are found to be exactly identical, providing a fault detection rate of 100%.

Based on the observed results, we can conclude that (a) All the EM injections that induced faults in the logic were detected. This validates the correctness of the detector and confirms the locality of EMFI attacks on Altera FPGA platform. (b) The detector triggers an alarm only when a fault is induced. This eliminates the unnecessary false positives that can be raised when the circuit is under attack, but no fault is being induced. The detector's alarm output can be reliably used by the controller to safely initiate device reset or other system calls.

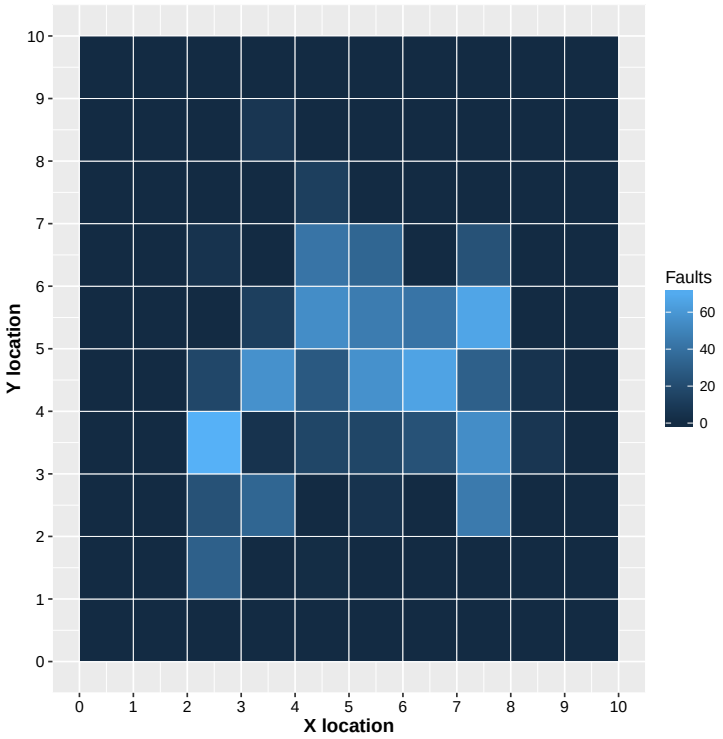

Figure 5.8: Heatmap showing the number of injections leading to a faulty output. All the induced faults are detected, giving 100% detection rate.

5.3.4 Hardware Overhead

We measure the hardware overhead of proposed mechanism by observing area and timing results for (a) AES-128 logic without detector (b) AES-128 logic with detector. The timing analysis reports maximum clock frequency for both the implementations as 108.39 MHz. This shows that the detector incurs no timing overhead. Table 5.1 shows the obtained area results. The area increase is found to be 0.05% and 17% in the number of LUTs and registers respectively.

5.4 Conclusion

In this chapter, we proposed a low-cost, fully-digital, and cycle-accurate mechanism to detect EMFI. We also presented a framework to integrate the proposed detector into any logic core. We implemented the proposed detection mechanism on an Altera Cyclone-IV FPGA and integrated it with a round-serial hardware implementation of AES-128. The area overhead of this design was 0.05% and 17.3% in the number of LUTs and registers, respectively. We also experimentally demonstrated the efficiency of the implemented design under actual EM fault injection. In our experiments, we applied 20000 EM pulses with different parameters (i.e., timing, location, pulse width, energy). In total, 7% of the applied pulses caused faults in the output of the AES. The proposed mechanism successfully detected all of the injected faults without any false alarms.

Chapter 6

Conclusion

Within this thesis, we introduced different fault attack detection methods in hardware. Software solutions are proven to be insecure against multiple fault injections and adaptive adversaries. We have designed and experimentally demonstrated novel low-cost and cycle-accurate detectors that can efficiently detect fault attacks without any false alarms. From a general point of view, our detectors can be applied to arbitrary logic and support protection against different attacks, namely EMFI, Clock glitching and Voltage starvation. The detectors can be added to any device without the need to change the rest of the design with an insignificant area and power overhead. They are fully digital and enable easy integration into both ASIC and FPGA platforms.

Bibliography

[1] H. Bar-El, H. Choukri, D. Naccache, M. Tunstall, and C. Whelan. The sorcerer's apprentice guide to fault attacks. *Proceedings of the IEEE*, 94(2):370–382, 2006.

[2] A. Barenghi, G. M. Bertoni, L. Breveglieri, M. Pelliccioli, and G. Pelosi. Injection technologies for fault attacks on microprocessors. In *Fault Analysis in Cryptography*, pages 275–293. Springer, 2012.

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

[4] 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.

[5] C. OFlynn and Z. Chen. Chipwhisperer: An open-source platform for hardware embedded security research. In *Proc. of COSADE*, pages 243–260.

[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 *VLSI (ISVLSI), 2016 IEEE Computer Society Annual Symposium on*, pages 461–466. IEEE, 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] D. El-Baze, J.-B. Rigaud, and P. Maurine. A fully-digital em pulse detector. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2016*, pages 439–444. IEEE, 2016.

[9] D. Ernst, S. Das, S. Lee, D. Blaauw, T. Austin, T. Mudge, N. S. Kim, and K. Flautner. Razor: circuit-level correction of timing errors for low-power operation. *IEEE Micro*, 24(6):10–20, 2004.

[10] X. Guo, D. Mukhopadhyay, and R. Karri. Provably secure concurrent error detection for advanced encryption standard. *ePrint Archive*, page 552.

[11] M. Hicks, M. Finnicum, S. T. King, M. M. Martin, and J. M. Smith. Overcoming an untrusted computing base: Detecting and removing malicious hardware automatically. In *Security and Privacy (SP), 2010 IEEE Symposium on*, pages 159–172. IEEE, 2010.

[12] H. Igarashi, Y. Shi, M. Yanagisawa, and N. Togawa. Concurrent faulty clock detection

for crypto circuits against clock glitch based dfa. In *Circuits and Systems (ISCAS), 2013 IEEE International Symposium on*, pages 1432–1435. IEEE, 2013.

[13] M. Karpovsky, K. Kulikowski, and A. Taubin. Differential fault analysis attack resistant architectures for the advanced encryption standard. *Smart Card Research and Advanced Applications VI*, pages 177–192, 2004.

[14] P. Luo and Y. Fei. Faulty clock detection for crypto circuits against differential fault analysis attack. *IACR Cryptology ePrint Archive*, 2014:883, 2014.

[15] P. Maurine. Techniques for em fault injection: equipments and experimental results. In *Fault Diagnosis and Tolerance in Cryptography (FDTC), 2012 Workshop on*, pages 3–4. IEEE, 2012.

[16] 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.

[17] 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.

[18] 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.

[19] J.-J. Quisquater and D. Samyde. Eddy current for magnetic analysis with active sensor. In *Proceedings of Esmart*, volume 2002, 2002.

[20] N. Selmane, S. Guilley, and J.-L. Danger. Practical setup time violation attacks on aes. In *Dependable Computing Conference, 2008. EDCC 2008. Seventh European*, pages 91–96. IEEE, 2008.

[21] S. Skorobogatov and C. Woods. Breakthrough silicon scanning discovers backdoor in military chip. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 23–40. Springer, 2012.

[22] J. G. Van Woudenberg, M. F. Witteman, and F. Menarini. Practical optical fault injection on secure microcontrollers. In *Fault Diagnosis and Tolerance in Cryptography (FDTC), 2011 Workshop on*, pages 91–99. IEEE, 2011.

[23] A. G. Yanci, S. Pickles, and T. Arslan. Characterization of a voltage glitch attack detector for secure devices. In *Bio-inspired Learning and Intelligent Systems for Security, 2009. BLISS'09. Symposium on*, pages 91–96. IEEE, 2009.

[24] C.-H. Yen and B.-F. Wu. Simple error detection methods for hardware implementation of advanced encryption standard. *IEEE transactions on computers*, 55(6):720–731, 2006.

[25] 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.