

Technical University of Crete
School of Electrical and Computer Engineering



Mitigating Side-Channel Attacks in the Context of Multi-Tenant FPGA Usage

Diktopoulos Christos

Thesis Committee:

Associate Professor Sotirios Ioannidis (supervisor) (ECE TUC)

Professor Apostolos Dollas (ECE TUC)

Dr. Konstantinos Georgopoulos

Chania, February 2022

Abstract

The rising use of multi-tenant FPGAs for cloud computing has created security concerns. Previous works have shown that malicious users can implement remotely, i.e., without physical access, voltage fluctuation sensors and mount successful power analysis attacks against cryptographic algorithms that share the same Power Distribution Network (PDN).

So far, masking and hiding schemes are the two main mitigation strategies against such attacks. One such work has shown that the use of an Active Fence of Ring Oscillators, which has a similar impact on the PDN as the cryptographic algorithm, if placed between two adversary users, can be an effective hiding countermeasure. Although this countermeasure is presented as platform independent, more recent platforms show different results against remote Side-Channel Attacks (SCAs).

This work presents the mapping of an intra-FPGA adversary scenario on two platforms, a ZedBoard and a Xilinx UltraScale+ MPSoC to assess the effectiveness of the Ring Oscillator Active Fence countermeasure. We compare different Active Fence configurations, with a varying number of Ring Oscillators, while using a new, resource efficient, activation method aiming to achieve noise injection hiding. The results show that by using our proposed Active Fence, which exhibits lower area overhead and, subsequently, lower power consumption than the algorithm under attack, the side-channel leakage is reduced to such a degree that the number of traces that need to be collected for a successful attack is more than ten times higher compared to no fence present. Moreover, this work presents quantitative results that FPGA cloud providers, may use to assess the benefits gained through the deployment of Active Fence mechanisms within their platforms prior to offering multi-tenant services.

Περίληψη

Η ανερχόμενη χρήση συστημάτων αναδιατασόμενης λογικής από πολλαπλούς χρήστες σε περιβάλλοντα που βρίσκονται στο νέφος, έχει εγείρει προβληματισμούς σχετικά με την ασφάλεια των χρηστών. Προηγούμενες μελέτες έχουν δείξει πως κακόβουλοι χρήστες μπορούν να υλοποιήσουν απομακρυσμένα, δηλαδή χωρίς φυσική πρόσβαση, αισθητήρες κατανάλωσης ενέργειας και να πραγματοποιήσουν επιθέσεις κατά κρυπτογραφικών αλγορίθμων που βρίσκονται στο ίδιο δίκτυο τροφοδοσίας.

Μέχρι τώρα, τεχνικές masking και hiding, αποτελούν τις δυο βασικές κατηγορίες αντιμετώπισης τέτοιων επιθέσεων. Μια αντίστοιχη μελέτη αποδεικνύει πως ένας ενεργός φράχτης από κυκλικούς ταλαντωτές, ο οποίος έχει αντίστοιχο αντίκτυπο στο δίκτυο τροφοδοσίας με τον κρυπτογραφικό αλγόριθμο και τοποθετείται μεταξύ των 2 χρηστών, αποτελεί αποτελεσματικό αντίμετρο κατά των απομακρυσμένων κακόβουλων επιθέσεων.

Η συγκεκριμένη εργασία παρουσιάζει την προσομοίωση ενός σεναρίου κακόβουλης επίθεσης, σε δύο πλατφόρμες, για την εκτίμηση της αποτελεσματικότητας του ενεργού φράχτη από κυκλικούς ταλαντωτές ως αντίμετρο. Συγκρίνουμε διαφορετικές υλοποιήσεις του ενεργού φράχτη, με διαφορετικό αριθμό από κυκλικούς ταλαντωτές, ενώ παράλληλα χρησιμοποιείται ένας νέος τρόπος ελέγχου του φράχτη με μικρό αριθμό πόρων, με σκοπό την απόκρυψη μέσω της εισαγωγής θορύβου στο σύστημα. Τα αποτελέσματα δείχνουν ότι με τη χρήση του προτεινόμενου ενεργού φράχτη, ο οποίος καταναλώνει μικρότερο αριθμό πόρων και επακόλουθα χαμηλότερη κατανάλωση ενέργειας από τον υπό επίθεση αλγόριθμο, η διαρροή δεδομένων μειώνεται σε τέτοιο βαθμό ώστε ο αριθμός των μετρήσεων που πρέπει να συλλεχθούν για μια επιτυχημένη επίθεση είναι πάνω από δέκα φορές υψηλότερη σε σύγκριση με την απουσία του ενεργού φράχτη. Επιπλέον, αυτή η εργασία παρουσιάζει ποσοτικά αποτελέσματα που οι πάροχοι υπηρεσιών νέφους με FPGAs μπορούν να χρησιμοποιήσουν για να αξιολογήσουν τα οφέλη που αποκομίζονται μέσω της χρήσης ενεργών φραχτών στις πλατφόρμες τους πριν από την προσφορά υπηρεσιών σε πολλαπλούς χρήστες.

Acknowledgements

First of all, I would like to thank Prof. Sotirios Ioannidis (TUC, FORTH) for entrusting me with this assignment and for being my supervisor. I would like to deeply thank him and Dr. Konstantinos Georgopoulos for their guidance and support during the work and writing of this thesis. I would also like to express my gratitude to Prof. Apostolos Dollas (TUC) for being the third member of the committee and for evaluating my thesis.

Furthermore, I would like to express my deepest gratitude to all the members of the Microprocessors and Hardware Lab (MHL) for supplying the means that were needed for conducting this thesis remotely, amidst the COVID-19 crisis. I would especially like to thank Dr. Georgios Chrysos, ECE MSc Andreas Brokalakis, ECE MEng Efstratios Koutroulakis, ECE MEng Morianos Ioannis, ECE MEng Thomas Kyriakakis and ECE MEng Vasileios Amourgianos for their valuable input and technical support during the work of my thesis.

Finally, I would like to thank my family and friends for their support over the years. This thesis is dedicated to them

Contents

Abstract	i
Περίληψη	ii
Acknowledgements	iii
Contents.....	iv
List of figures.....	vi
List of tables	viii
Acronyms - Abrevations.....	ix
Chapter 1: Introduction	1
1.1 Background.....	1
1.2 Thesis contributions	2
1.3 Thesis structure	2
Chapter 2: Related Work.....	3
2.1 Multi-Tenant Attack Model.....	3
2.2 Power Distribution Network	3
2.3 Electrical Level Attacks	4
2.4 Remote Power Side-Channel Attack Sensors.....	5
2.4.1 Ring Oscillator Sensors.....	5
2.4.2 Time-to-Digital Converter Sensors.....	5
2.5 Advanced Encryption Standard Algorithm (AES)	8
2.6 Corelation Power Analysis.....	11
2.6.1 Attack Methodology	11
2.7 Defence Mechanisms Against SCAs	13
Chapter 3: Platforms and Tools	16
3.1 Platforms	16
3.1.1 ZedBoard™	16
3.1.2 ZYNQ UltraScale+ MPSoC ZCU104	17

3.2	Basic Tools	19
3.2.1	Xilinx Vivado Design Suite 2020.2.....	19
3.2.2	Xilinx Vitis Unified Software Platform Documentation	19
3.3	SCAbox UI: Vitis Bare-Metal App and Python Corelation Analysis App.....	20
3.3.1	Vitis Bare-Metal Application	20
3.3.2	Python Corelation Analysis Application	23
3.4	The AXI protocol.....	24
3.5	UART serial connection	25
Chapter 4: System Architecture.....		27
4.1	Emulation Assumptions.....	27
4.2	Victim and Attacker Emulation	27
4.2.1	AES algorithm.....	27
4.2.2	TDC – sensor Bank.....	28
4.2.3	Memory.....	30
4.2.4	Active Fence Architecture.....	31
4.2.5	Design Clocking	33
4.2.6	Remaining Modules	33
Chapter 5: Experimental Procedure		39
Chapter 6: Results and Evaluation		40
6.1	ZedBoard Platform Results.....	40
6.2	ZCU104 Platform Results.....	47
Comparison of the two platforms.....		53
6.3	Comparison with Similar Works.....	54
6.4	Results' Evaluation Summarization	60
Chapter 7: Conclusions and Future Work.....		61
Chapter 8: References.....		63

List of figures

Figure 2.1.i: Adversary Model Block Diagram.....	3
Figure 2.4.i : Initial Delay Line of TDC and Timing Diagram.....	6
Figure 2.4.ii: Three Scenarios of TDC Output.....	7
Figure 2.4.iii: RemoteRemote Power Side-Channel Attack Sensors.....	8
Figure 2.5.i: AES Block Diagram	10
Figure 2.6.i: AES Last Round.....	12
Figure 2.7.i: Adversary Model With Active Fence Block Diagram	15
Figure 3.1.i: ZedBoard Platform.....	16
Figure 3.1.ii: ZedBoard Block Diagram.....	17
Figure 3.1.iii: ZCU104 Platform	18
Figure 3.1.iv: ZYNQ UltraScale+ MPSoC ZCU104 Block Diagram	18
Figure 3.2.i: Vivado Desing Suite High-Level Design Flow	19
Figure 3.2.ii: Xilinx Vitis Software/Hardware Build Process	20
Figure 3.3.i: SCABox Starting Screen.....	21
Figure 3.3.ii: SCABox TDC Senor Value Instruction	21
Figure 3.3.iii: SCABox Read FIFO Instruction	22
Figure 3.3.iv: SCABox Multiple AES Encryptions Instruction	23
Figure 3.3.v: SCABox Python Application UI	24
Figure 3.4.i: AXI Interconect Block Diagram	25
Figure 3.5.i: UART Connection Block Diagram.....	25
Figure 4.2.i CARRY4 primitive	29
Figure 4.2.ii CARRY8 primitive	30
Figure 4.2.iii: Active Fence Block Diagram.....	32
Figure 4.2.iv: Ring Oscillator Sensor with Libaw-Craig Counter	33
Figure 4.2.v: ZedBoard Design Block Diagram.....	34
Figure 4.2.vi: ZCU104 Design Block Diagram	35
Figure 4.2.vii: ZedBoard Floorplaning.....	36
Figure 4.2.viii: ZCU104 Floorplaning	37
Figure 6.1.i: Quantification vs. Time Samples. Average of 500 AES Iterations wihtout countermeasure (ZedBoard).....	41
Figure 6.1.ii: Pearson's Correlation Plots for the Second Key Byte with no Active Fence	42

Figure 6.1.iii Quantification vs. Time Samples. Average of 500 AES Iterations with Active Fence (ZedBoard)	43
Figure 6.1.iv: Pearson's Correlation Plots for the Fourth Key Byte with 1024 RO Active Fence	44
Figure 6.1.v: Pearson's Correlation Plots for the Fourth Key Byte with 4096 RO Active Fence	45
Figure 6.1.vi : ZedBoard CPA Results Plot	47
Figure 6.2.i: Quantification vs. Time Samples. Average of 500 AES Iterations without countermeasure (ZCU104)	47
Figure 6.2.ii: Quantification vs. Time Samples. Average of 500 AES Iterations with Active Fence (ZCU104)	48
Figure 6.2.iii: Pearson's Correlation Plots for the Second Key Byte without Active Fence (ZCU104)	50
Figure 6.2.iv: Pearson's Correlation Plots for the Fourth Key Byte with 5120 RO Active Fence (ZCU104)	51
Figure 6.2.v: ZCU104 CPA Results Plot	52
Figure 6.4.i Comparisson of Each Platform	59

List of tables

Table 4.2-1: ZedBoard Design Total Resource Utilization	38
Table 4.2-2: ZCU104 Design Total Resource Utilization	38
Table 6.1-1: ZedBoard CPA Results	46
Table 6.2-1 : ZCU104 CPA Results	52
Table 6.2-2: ZedBoard Vs ZCU104 implementations.....	53
Table 6.3-1: ZedBoard Implementations Vs. Glamocanin et. al. Work	55
Table 6.3-2: ZCU104 Implementations Vs. Glamocanin et. al. Work	56
Table 6.3-3: ZedBoard Implementations Vs. Krautter et. al. Work	57
Table 6.3-4: ZCU104 Implementations Vs. Krautter et. al. Work	58

Acronyms - Abbreviations

ACAP	Adaptive Compute Acceleration Platform
AES	Advanced Encryption Standard
AMBA	Advanced Microcontroller Bus Architecture
API	Application Programming Interface
ARM	Advanced RISC (Reduced Instruction Set Computer) Machines
AXI	Advanced eXtensible Interface
CPA	Correlation Power Analysis
FIFO	First In First Out
FPGA	Field-Programmable Gate Arrays
HD	High Density
HP	High Performance
HPC	High-Performance Computing
HW	Hardware
MIO	Multi-use I/O
MPSoC	MultiProcessor System-on-Chip
OS	Operating System
PCB	Printed Circuit Board
PDN	Power Distribution Network
PL	Programmable Logic
PS	Processing System
PVT	Process, Voltage and Temperature

RO	Ring Oscillator
RSA	Rivest–Shamir–Adleman
SCA	Side-Channel Attack
SoC	System-on-chip
SPA	Simple Power Analysis
TDC	Time-to-digital converter
UART	Universal Asynchronous Receiver/Transmitter
US+	UltraScale+

Chapter 1: Introduction

1.1 Background

Field Programmable Gate Arrays (FPGAs) are able to provide high computing efficiency and flexibility as they combine the benefits of hardware compute engines with the reprogrammability offered by their self-reconfiguration capabilities. They are widely deployed as accelerator engines for numerous applications, from embedded to High-Performance Computing (HPC) and cloud systems, with vendors offering multiple System-on-Chip (SoC) solutions and cloud service providers such as Amazon AWS [1] providing compute instances with one or more FPGA devices.

The deployment of FPGAs in cloud services has significantly broadened their reach and FPGA-accelerated applications have been gaining sizable traction in the research, educational and commercial domains [2],[3],[4]. Nevertheless, this trend began facing an obstacle that needs to be surmounted such that the full potential of this type of technology can be taken advantage of.

Hardware accelerated functions often consume no more but a fraction of the available reconfigurable resources of modern FPGA devices. Cloud providers do not have the flexibility to provide numerous alternate FPGA devices to match each potential use-case and, therefore, for cost efficiency reasons, they tend to offer large devices that are able to accommodate the most demanding implementations. As such, more often than not, FPGA devices are significantly underutilised and shared in a highly inefficient way among different users.

It becomes, therefore, apparent that a solution to this problem is to share the reconfigurable resources between different applications and/or users, thereby, increasing utilisation and cost efficiency. In other words, the goal is to have multiple users cohabiting in the same compute instance, which, consequently, leads to a multi-tenancy scenario for cloud FPGA devices. Unfortunately, security concerns are raised as Side-Channel Attacks (SCAs) are known to be feasible when multiple circuits are implemented on the same fabric [5]. That is, neighbouring, albeit independent designs, can be exposed to a number of different attacks which pose significant risks [6],[7],[8]. The type of attack that our work focuses on, is the power side-channel attack, one of the most popular methods for implementing SCAs.

1.2 Thesis contributions

In that context, various countermeasures have been proposed, such as logical isolation [9], masking techniques [10], [11] and hiding techniques [12]. The work presented in this thesis emulates a 2-tenant FPGA adversary scenario between a malicious user implementing a power sensor against a 128-bit AES hardware module. It evaluates the use of different Active Fence configurations, originally introduced by Krautter et al. [12], as an algorithm-independent hiding countermeasure against remote power SCAs. In addition, a new Active Fence control method that uses a small number of resources is proposed. The design and its configurations, are implemented and compared between two platforms. The AVNET ZedBoard™ equipped with the Xilinx Zynq®-7000 processor and 7-Series programmable logic and the Xilinx Zynq UltraScale+ MPSoC ZCU104 equipped with a quad-core ARM® Cortex™-A5 and Xilinx UltraScale+ programmable logic, the most commonly used programmable logic by cloud providers.

1.3 Thesis structure

The remainder of this work is organized as follows:

- Chapter 2: Related Work
- Chapter 3: Platforms and Tools
- Chapter 4: System's Architecture
- Chapter 5: Experimental Procedure
- Chapter 6: Results and Evaluation
- Chapter 7: Conclusions and Future Work

Chapter 2: Related Work

2.1 Multi-Tenant Attack Model

FPGA-based side-channel attacks have been conducted under three different scenarios:

- 1) *Inter-Chip Attack*: The Inter-Chip Side-channel proves that an untrusted chip inside a PCB can sense voltage variations induced by other chips through the power distribution network (PDN). In this exploit, an adversary FPGA is able to perform a CPA attack against an AES module and a SPA attack against an RSA module running on another FPGA fabric.
- 2) *Heterogeneous Chip Attack*: Xilinx Zynq technology integrates a dual core ARM processor and a FPGA fabric within the same SoC. In [8], malicious ROs were implemented in the FPGA fabric to perform a SPA against an RSA algorithm running on a Linux OS inside the ARM CPU core.
- 3) The main attack model that must be examined in a cloud multi-tenant FPGA scenario is the Intra-FPGA Attack, where the FPGA fabric is shared among two or more users for hardware module implementation [8], [13], [14]. Each user can program an arbitrary design in a partial region of the fabric. Nevertheless, despite the fact that empty slices and lines of empty DSP blocks [15] ensure logical isolation between the different users and designs, a malicious user can implement voltage sensors in his part of the FPGA fabric to remotely monitor voltage fluctuations induced by surrounding computation modules.

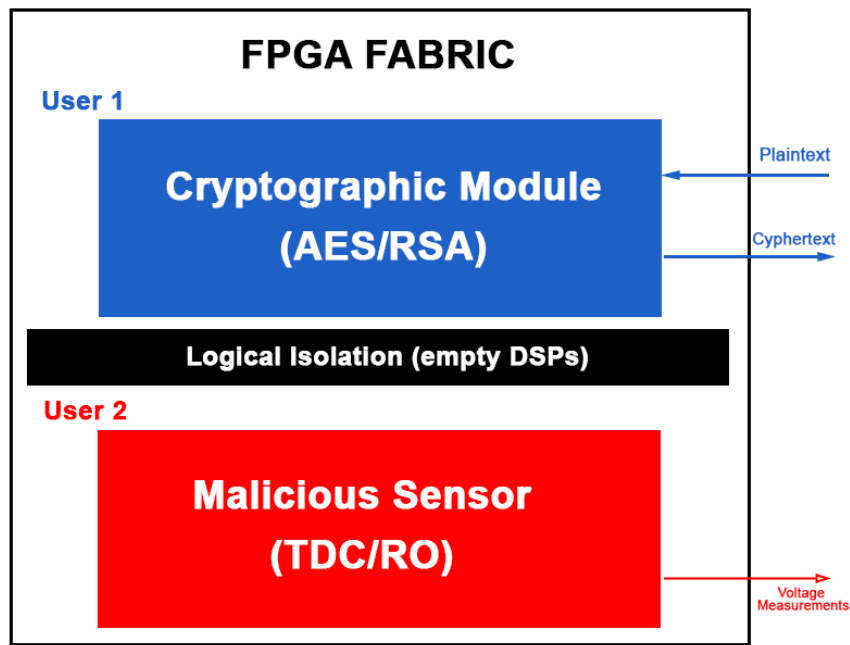


Figure 2.1.i: Adversary Model Block Diagram

2.2 Power Distribution Network

PDNs deliver power from a main power source to the individual transistors inside an electronic chip ([16], [17]) and, therefore, are commonly found within FPGAs and MPSoC

chip packages. They are crucial components as they ensure that voltage is properly distributed among different parts of the chip so that transition delay remains as stable as possible. If the delay surpasses a certain limit, timing violations and errors may occur. As such, the power supply level must be as stable as possible. Moreover, some components of the chip and the PDN act as parasitic transistors and inductors [16]. Consequently, they have an impact on the circuit's switching activity because they generate inductive and resistive voltage drops. It becomes, therefore, possible to extract information from implemented designs simply by monitoring the switching activity on the chip, through the implementation of appropriate voltage sensors. [18]

2.3 Electrical Level Attacks

Side-channel attacks are closely related to the existence of physically observable phenomena caused by the execution of computing tasks in present microelectronic devices. For example, microprocessors consume time and power to perform their assigned tasks. They also radiate an electromagnetic field, dissipate heat, and even make some noise. There are plenty of information sources leaking from actual computers that can consequently be exploited by malicious adversaries.

Power analysis is a branch of side channel attacks where power consumption data is used as the side channel to attack the system. In electronic devices, the instantaneous power consumption is dependent on the data that is being processed in the device as well as the operation performed by that device. By capturing power consumption of a cryptographic algorithm, the attacker can perform a statistical evaluation to test which key hypothesis results in intermediate values that indeed correspond to the observed power consumption. For Correlation Power Analysis (CPA), the correct key candidate will show the highest correlation when the attack is successful. Because the side-channel leakage is usually very small, many traces have to be recorded and evaluated. Thus, side-channel resilience is usually measured in the number of traces required for a successful attack.

Such attacks are widely studied and typically require physical access to the device to capture the power consumption, i.e., to connect the probe of an oscilloscope or to place an EM probe in the near vicinity of the device. Yet, recent work has shown remote attacks on multi-tenant FPGAs that break with this assumption, i.e., without any physical access to the device. Using the on-chip power distribution network as source or carrier of side-channel information, user programmable FPGA primitives can be utilized in order to implement an on-chip voltage sensor.

A circuit's activity will result in voltage fluctuations across the PDN, independent of any logical connection. Given that, the propagation delay of a circuit depends on the supply

voltage, capturing the activity of a circuit becomes the task of measuring the speed of the circuit.

2.4 Remote Power Side-Channel Attack Sensors

Remote power side-channel attacks, namely the instigation of an attack from afar and with no physical access to the device, exploit voltage fluctuations on the power distribution network of the chip by implementing reconfigurable logic sensors on the fabric. These sensors are capable of capturing nanosecond voltage fluctuations [19] and, therefore, make it possible for malicious users to extract sensitive data from algorithms, such as the key of a cryptographic hardware module, even without any physical access to the chip [20], [8], [14].

Commonly, there exist two main types of voltage sensors for mounting remote power side-channel attacks. The Ring Oscillator (RO) based delay sensor and the taped delay-line or Time-to-Digital Converter (TDC) sensor.

2.4.1 Ring Oscillator Sensors

RO sensors consist of a NAND gate, which uses as inputs an external *enable* signal and the output of the gate itself. When enabled, it creates an infinite oscillation and its propagation delay changes along with voltage variations, effectively changing its oscillation frequency. Therefore, measuring the frequency variations of the RO with digital counters, provides information on the power consumption [13].

2.4.2 Time-to-Digital Converter Sensors

On the other hand, TDC-based sensors work by checking how far a signal can propagate through a path by adding latches between the logic elements of that path. The simplest path is a chain of buffers. As the delays of these buffers are sensitive to any joint process, voltage, and temperature (PVT) variations, they will become measurable by reading the latches. To do that, a signal is connected to both the input of the first buffer and the latch enable signals, which means that the signal on the wire will be faster than the same signal delayed through the buffer elements. A readily available and precisely timed signal is the clock signal itself. At half of the clock period, the latches will become disabled and keep the state how far the clock propagated through the buffer chain, which can then be processed further.

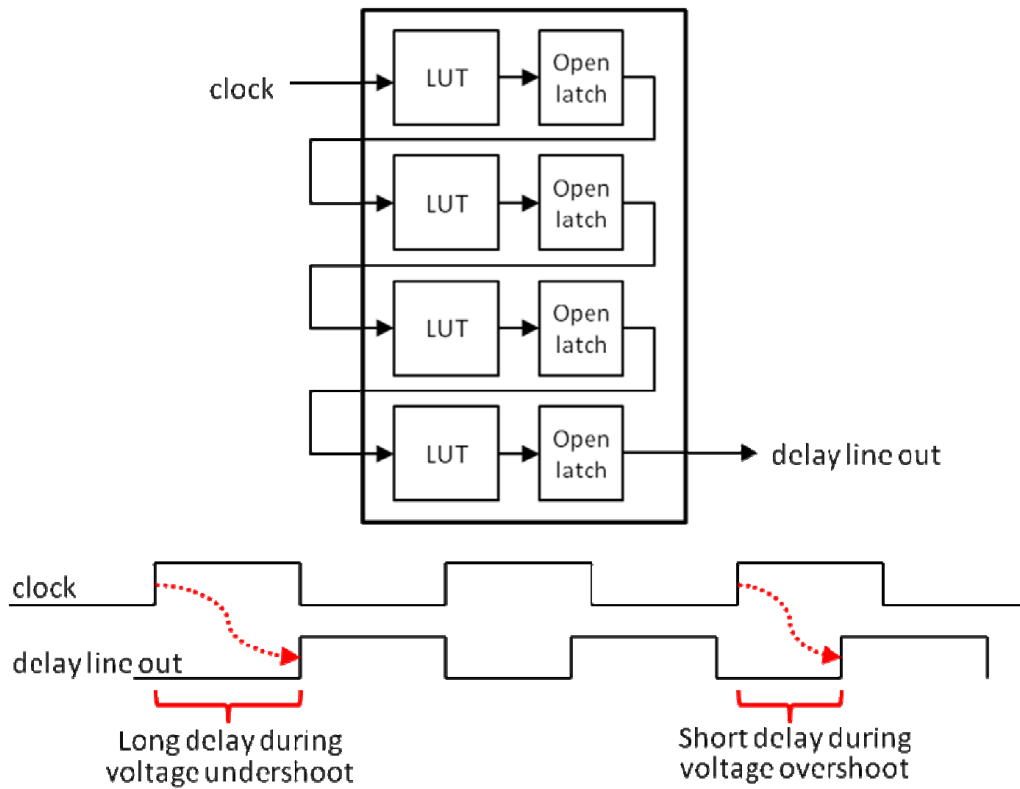


Figure 2.4.i : Initial Delay Line of TDC and Timing Diagram

The initial delay needs to be adjusted for less than a half of the clock period (the time when latches are enabled). Then, latches are connected between the last buffer elements, marked as the *observable delay line*, that fall within the expected delay variation that we want to observe with the sensor. This limited operating range is chosen based on the acceptable area overhead and expected operating range values.

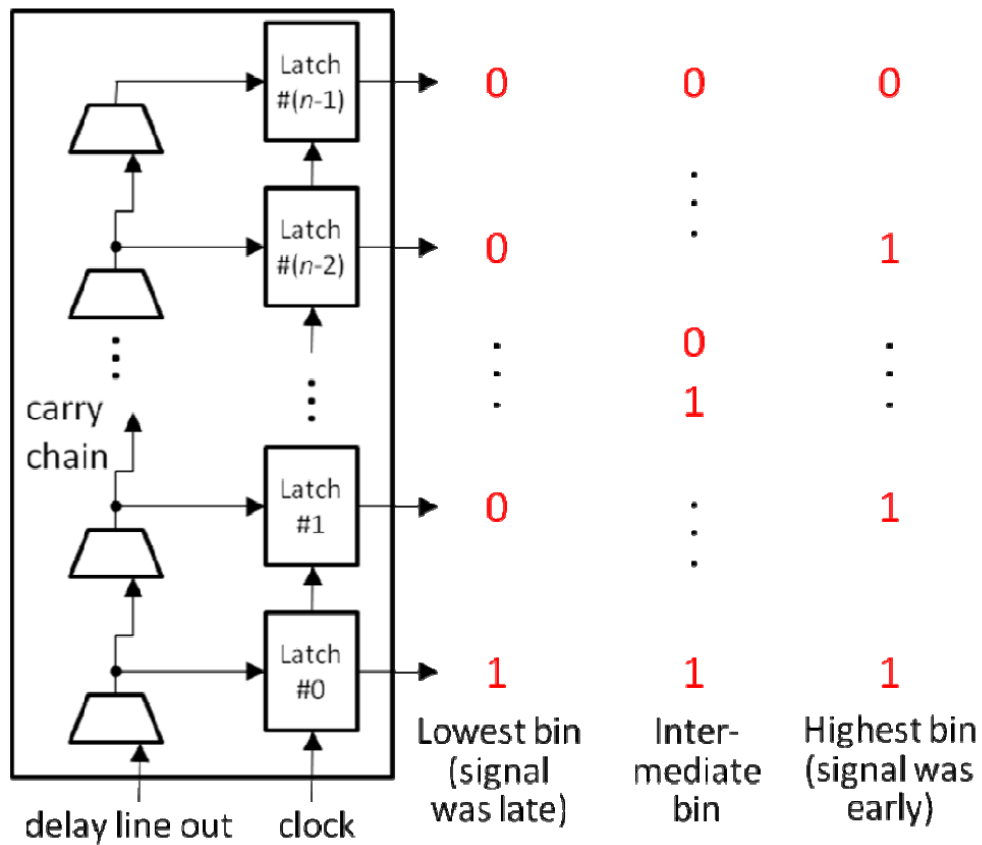


Figure 2.4.ii: Three Scenarios of TDC Output

Comparably, the circuitry required by RO sensors is less complex and they, therefore, require lower FPGA fabric resources to be implemented. Nevertheless, they offer less accuracy, since the use of counters for measuring oscillation frequency does not allow for nanosecond scale sampling. In addition, they are easy to detect and flag as malicious designs, hence, not that much preferable as an attack mechanism. In contrast, TDC sensors enable nanosecond scale measurement of the FPGA's internal voltage fluctuations and can be used as thermal sensors. Also, it is difficult for design tools to block their implementation. Thus, TDCs constitute a far more attractive implementation for power SCAs.

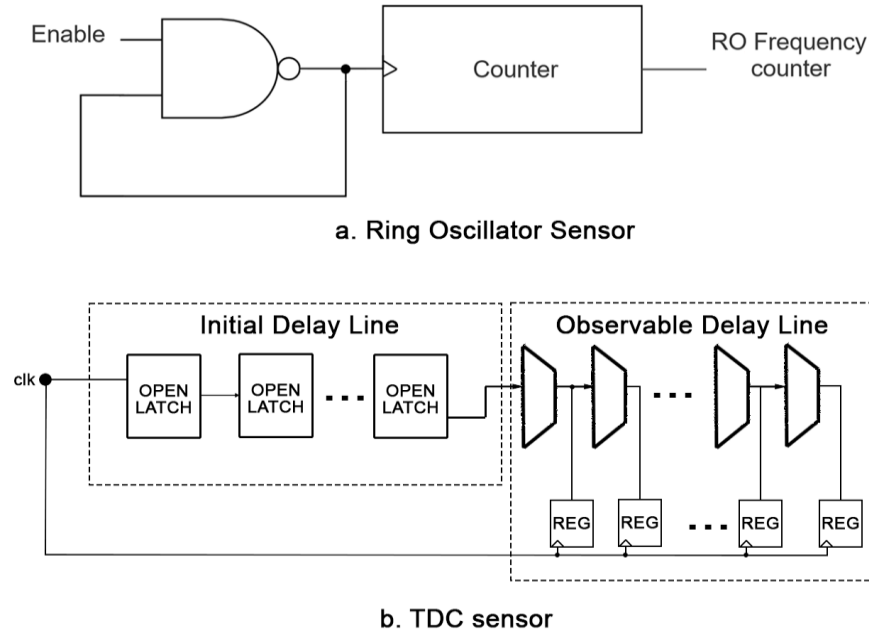


Figure 2.4.iii: RemoteRemote Power Side-Channel Attack Sensors

2.5 Advanced Encryption Standard Algorithm (AES)

The Advanced Encryption Standard is a symmetric block cipher that process data blocks using cipher keys with lengths of 128, 192 and 256 bits [16]. Each data block consists of 4×4 array of bytes called the state. The AES is a round-based encryption algorithm. The number of rounds, N_r , is 10, 12, or 14, when the key length is 128, 192 or 256 bits, respectively. In the encryption of the AES algorithm, each round, except the final round, performs four transformations: AddRoundKey, SubBytes, ShiftRows and MixColumns, while the final round does not have the MixColumns transformation. The key used in each round, called the round key, is generated from the initial key by a separate key scheduling module.

The SubBytes transformation is a non-linear byte substitution, operating on bytes independently. The SubBytes is invertible and is constructed by the composition of the following two transformations:

- Inversion in the $GF(2^8)$ field, modulo an irreducible polynomial $m(x)$ given by:

$$m(x) = x^8 + x^4 + x^3 + x + 1 \quad (1)$$

- Affine transformation defined as follows: $Y = AX^{-1} + b$, where A is a 8×8 fixed matrix and b is a 8×1 vector-matrix.

The ShiftRows transformation is a circular shifting operation on the rows of the state with different numbers of bytes. As seen in below, the first row of the state is kept as it is, while the second, third and fourth rows cyclically shifted by one byte, two bytes and three bytes to the left, respectively.

$$\begin{bmatrix} s_0 & s_4 & s_8 & s_{12} \\ s_1 & s_5 & s_9 & s_{13} \\ s_2 & s_6 & s_{10} & s_{14} \\ s_3 & s_7 & s_{11} & s_{15} \end{bmatrix} \xrightarrow{\text{ShiftRows}} \begin{bmatrix} s_0 & s_4 & s_8 & s_{12} \\ s_5 & s_9 & s_{13} & s_1 \\ s_{10} & s_{14} & s_2 & s_6 \\ s_{15} & s_3 & s_7 & s_{11} \end{bmatrix}$$

The MixColumns transformation operates on the state column by column, treating each column as a four-term polynomial. The columns are considered as polynomials over $GF(2^8)$ and multiplied $x^4 + 1$ with a fixed polynomial $a(x)$ given by:

$$A(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\}$$

In matrix form, the MixColumns transformation can be expressed as:

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix}$$

$$0 \leq c \leq 3$$

The AddRoundKey is a XOR operation that adds a round key to the state in each iteration, where the round keys are generated during the Key Expansion phase.

The AES algorithm takes the cipher key and performs a Key Expansion routine to generate a key schedule. The Key Expansion generates a total $Nb(Nr + 1)$ words, where $Nb = 4$.

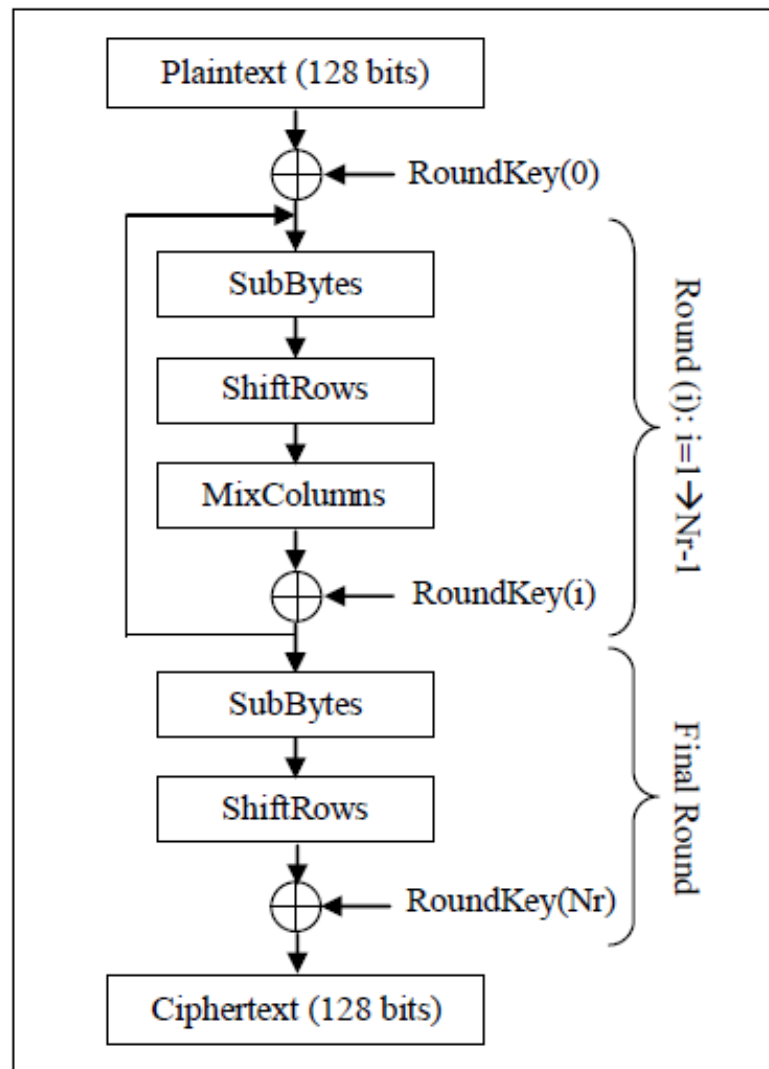


Figure 2.5.i: AES Block Diagram

2.6 Corelation Power Analysis

The hypothesis of the correlation power analysis (CPA) is that the measured power traces of the target device are correlated to the operands and operations being processed at that time. This type of power analysis technique requires a power model to attack a cryptographic device. The adversary needs to build a hypothetical model of the cryptographic device under attack. He can recover secret data by analysing the power consumption usage during cryptographic operations. An efficient way to calculate the correlation coefficient, between theoretical predictions of the power consumption and real power measurements is to use the Pearson correlation function.

The Pearson correlation function is the most widely used way to compute the linear relationship between data. Whence, it is an excellent choice for statistical analysis tool when it comes to perform CPA attacks.

Given N plaintexts/ciphertexts, P the predicted power calculated by a power model and W the equivalent real power traces measured when processing the cryptographic operation. The correlation coefficient is defined as:

$$\rho(W, P) = \frac{\text{Cov}(W, P)}{\sqrt{\text{Var}(W)} \sqrt{\text{Var}(P)}}$$

where W and P , are N -dimension vectors, Cov denotes the covariance operation, and Var denotes the variance operation.

The Pearson correlation coefficient can take values from -1 to +1. A value of +1 shows that the W and P are perfectly linear related by an increasing relationship, a value of -1 shows that W and P are perfectly linear related by a decreasing relationship, and a value of 0 shows that W and P are not linear related by each other.

2.6.1 Attack Methodology

The AES transforms 128-bit plaintext with the 128-bit key to 128-bit ciphertext. Each round has a round key: k_1 to k_{10} , computed from the original key k_0 . The attack is performed on the last round encryption because the latter has been isolated from the other rounds and has relatively clear power signals [14]. Since the AES Key Expansion is invertible, it is then possible to compute the initial secret key, k_0 , going backwards.

Then the power consumption of the last round encryption is predicted. Let C_{10} the output ciphertext of the last round and D_{10} the input data to this round. The ciphertext C_{10} is picked up to compute D_{10} by Inverse-ShiftRows and Inverse-SubBytes using the guessed keys K_{10} (256 possible values).

$$D_{10} = \text{SubBytes}^{-1}(\text{ShiftRows}^{-1}(K_{10(\text{guess})} \text{ XOR } C_{10}))$$

After that, the prediction of power consumption of the last round is calculated by the Switching Distance (SD) between C_{10} and D_{10} as presented below.

$$P_{prediacted} = SD(D_{10}, C_{10})$$

The Switching Distance model (SD) is based on the fact that $0 \rightarrow 1$ and $1 \rightarrow 0$ transitions consume different power in CMOS device. The Switching Distance of the transition $0 \rightarrow 1$ is assigned 1, the Switching Distance of the transition $1 \rightarrow 0$ is assigned Φ which is named Switching Distance factor.

The correlation coefficient between the measured power consumption, denoted $P_{measured}$, and the predicted power consumption $P_{predicted}$ is calculated as follows:

$$\rho(P_{measured}, P_{predicted}) = \frac{Cov(P_{measured}, P_{predicted})}{\sqrt{Var(P_{measured})}\sqrt{Var(P_{predicted})}}$$

where $Cov(P_{measured}, P_{predicted})$ is the covariance between the measured power consumption and the predicted power consumption, the $Var(P_{measured})$ and $Var(P_{predicted})$ are the variance of the $P_{measured}$ and the $P_{predicted}$ respectively.

The correlation coefficient measures the linear relationship between $P_{measured}$ and $P_{predicted}$. Its value will always be between -1 to 1, when the correct key guess appears, the correlation coefficient is supposed to be highest.

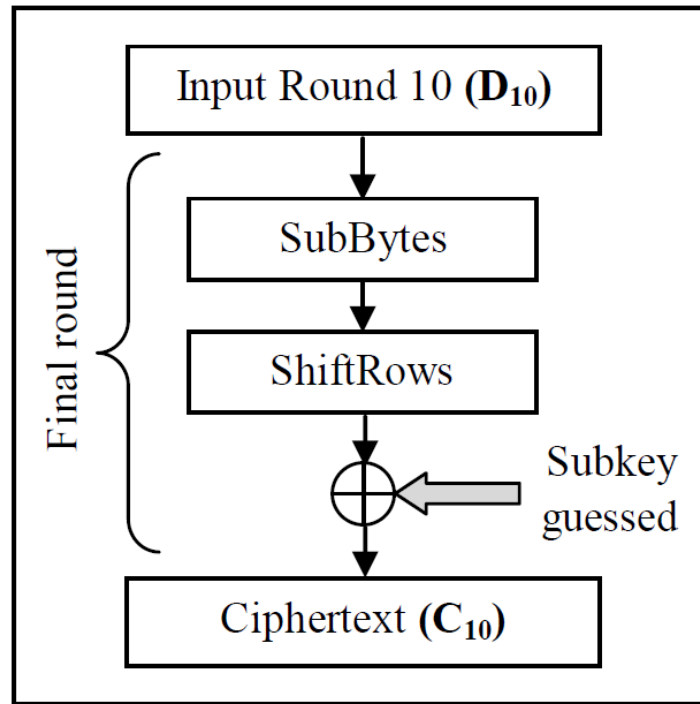


Figure 2.6.i: AES Last Round

2.7 Defence Mechanisms Against SCAs

A number of countermeasures against power SCAs exist and can be considered in the context of multi-tenant cloud FPGAs. One such countermeasure is to perform checks on the user bitstreams, prior to programming the FPGA resources [21]. Checks can be performed for known malicious designs, such as voltage sensors, and upon detection the reconfiguration of the device can be blocked. One such instance, involves blocking designs that contain ROs, however, there are implementations of sensors that can bypass those checks and therefore the detection mechanisms must be updated frequently and the platform provider must be aware that it may not be possible to capture every possible malicious design.

Moving at the device level, a common countermeasure is the logical isolation between the different design instances. Empty slices surrounding each design, create an isolation barrier between them, known as a *Passive Fence*, aiming at preventing neighbouring designs from sharing device components as a result of their proximity at the physical level. Although this has been a documented design practice for improving security in FPGAs, these Passive Fences have proven to be entirely ineffective against attacks at the electrical level [9], [15], [22], [13]. Connections remain through the power distribution network and, therefore, side-channel attacks may be performed no matter the size or properties of the Passive Fence(s). Consequently, alternative countermeasures against power side-channel attacks need to be considered, making them one of the main topics of ongoing research on FPGA security. Presently there are two types of such countermeasures, i.e., **Masking** and **Hiding**.

2.7.1.1 Masking

It applies algorithmic-level changes that focus on separating intermediate values and computations into different design module instances [10]. This way, the sensitive data can only be extracted by combining the values computed by each individual module. This method amplifies the noise exponentially, making attacks harder to mount, [23] and [24]. However, it comes with significant disadvantages. Each algorithm must be examined individually and a different implementation has to be devised. Furthermore, in numerous cases, e.g., non-linear functions, it may be impossible to divide up the computation effectively into individual parts. Lastly, this method creates an excessive area overhead and increases the complexity as well as the cost of the design[25].

2.7.1.2 Hiding

The main goal of hiding is to reduce the Signal-to-Noise Ratio (SNR) at the electrical level and it follows two approaches. The first is based on the introduction of noise sources, aiming at artificially increasing the level of noise. Noise generators have been implemented in the past using, for instance, shift registers, BRAM write collision, and short-term short circuits[11]. The second approach involves the generation of correlated noise and clock randomisation to spread side-channel information. This second direction aims at equalising the instantaneous power consumption by balancing the overall consumption. However, as perfectly balanced computation paths are hard to achieve, a proposed design practice to accomplish them is to duplicate and invert the computations in the victim design, as proposed in [26] for a cryptographic module. The drawback of this practice is that it results in a large area overhead rendering it impossible to be used with architectures that need many resources. Hence, what has been considered as a solution to the cost of significant fabric resources when aiming for power equalisation, is the use of ROs.

2.7.1.3 Active Fence Countermeasure

As an attack mechanism, RO are used to detect voltage fluctuations on the PDN of a chip, nevertheless, they can also be used for defence purposes by increasing the overall power consumption. Thus far, the only work on the use of ROs as a defence mechanism against remote power side-channel attacks is the RO Active Fence presented in [12]. Its presence can affect the voltage fluctuations and induce additional noise to what is being observed by a sensor implemented by a malicious co-tenant. Finally, by introducing an Active Fence between the different users, a two to three orders of magnitude leakage reduction is observed.

The activation and deactivation of the ROs is controlled and when they are enabled, they have a high switching activity, which leads to additional power consumption. Thus, at the expense of a relatively small area overhead, it is possible to efficiently inject a high voltage drop into the power grid, which equalises the PDN's voltage fluctuations, i.e., the indirect information leakage from the victim or target module. In the original scheme of [12], the fence size, in terms of hardware resources, is the same as those of the target module and its activation is controlled using either a TDC or a random generator. Nevertheless, these two approaches raise a number of issues.

First, an activation mechanism based on a TDC may be superfluous, which means that a simpler but equally effective design can be used for the same job. In addition, an activation

mechanism based on a random generator may be wasteful in terms of resources and power consumption as well as the fact that, by the authors' own admission, it is a worse solution when compared to the TDC activation method. Finally, the extent to which the fence size affects the outcome of the attack, has to be researched further. It may be possible for fences of less area overhead, compared to the one of the modules under attack, to remain effective and discourage potential attacks.

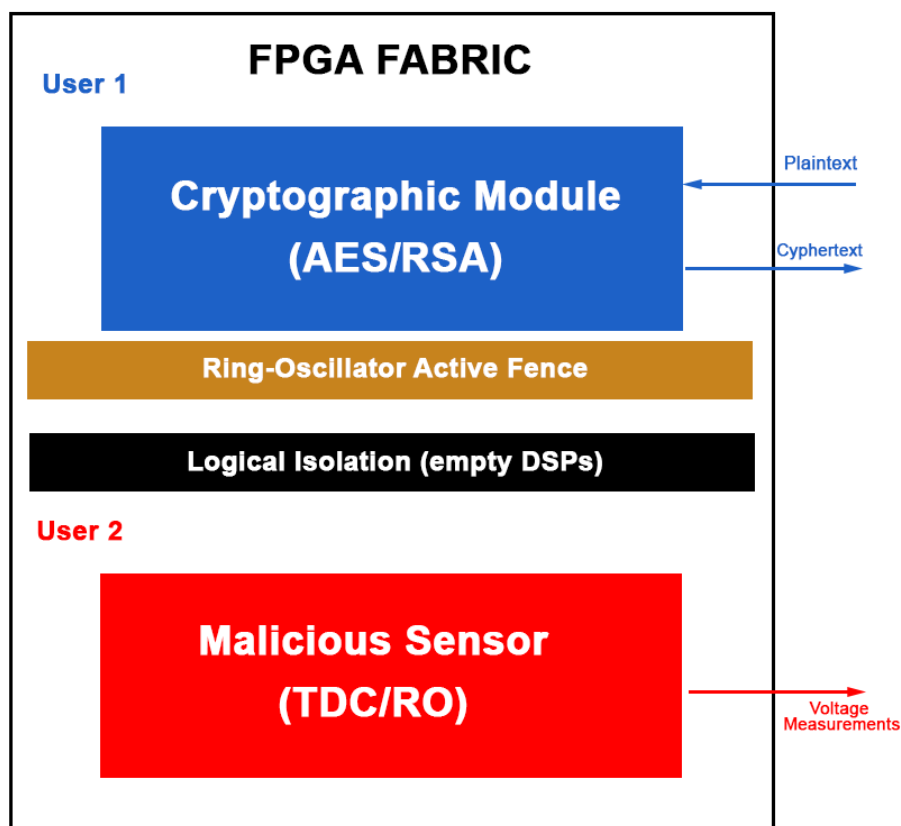


Figure 2.7.i: Adversary Model With Active Fence Block Diagram

Chapter 3: Platforms and Tools

3.1 Platforms

The two main platforms that were used to emulate and implement the adversary scenario are the following:

3.1.1 ZedBoard™

The ZedBoard™ [27], is a low-cost development board for the Xilinx Zynq®-7000 All Programmable SoC. The Zynq®-7000 family is based on the Xilinx SoC architecture. ZedBoard integrates a feature-rich dual-core ARM® Cortex™-A9 based processing system (PS) and 28 nm Xilinx programmable logic (PL) in a single device. The ARM Cortex-A9 CPU is the heart of the PS and also includes on-chip memory, external memory interfaces, and a rich set of peripheral connectivity interfaces. Zedboard is supported by the zedboard.org community website where users can collaborate with other engineers also working on Zynq designs. The ZedBoard is supported by Xilinx's Vivado Design Suite, including the free WebPACK version. Some of its core features include :

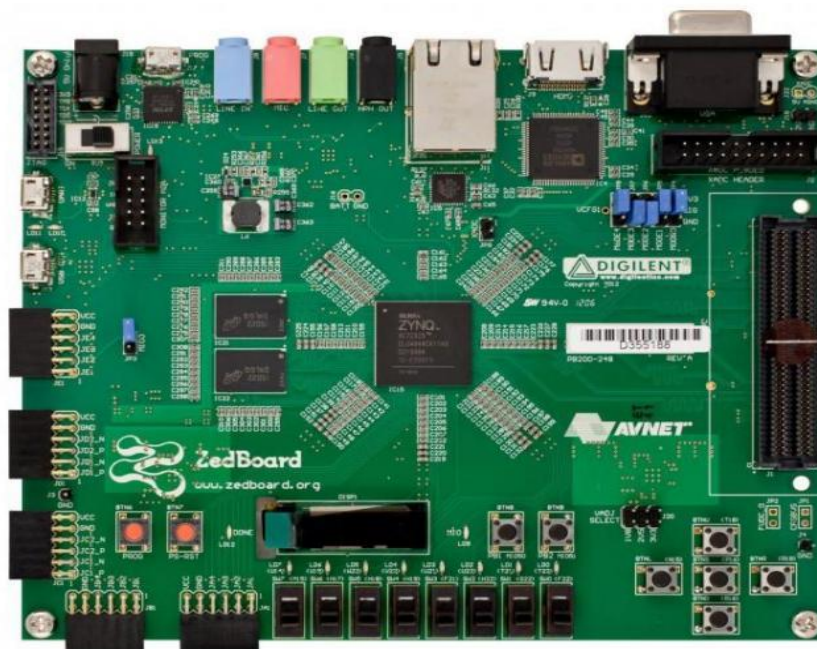


Figure 3.1.i: ZedBoard Platform

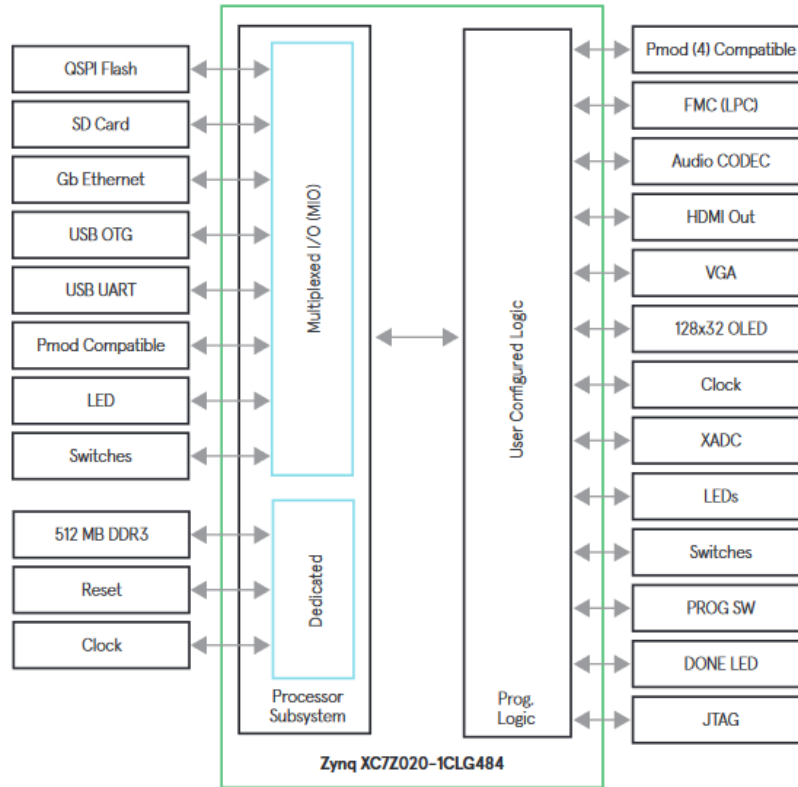


Figure 3.1.ii: ZedBoard Block Diagram

3.1.2 ZYNQ UltraScale+ MPSoC ZCU104

The ZCU104 [28], provides a rapid prototyping platform using the XCZU7EV-2FFVC1156 device. The ZU7EV contains PS hard block peripherals exposed through the multi-use I/O (MIO) interface and several FPGA programmable logic (PL), high-density (HD), and high-performance (HP) banks. The ZU7EV device integrates a quad core Arm® Cortex™-A53 processing system (PS) and a dual-core Arm Cortex-R5 real-time processor, which provides application developers an unprecedented level of heterogeneous multiprocessing. The ZCU104 evaluation board provides a flexible prototyping platform with high-speed DDR4 memory interfaces, multi-gigabit per second serial transceivers, a variety of peripheral interfaces, and FPGA fabric for customized designs.

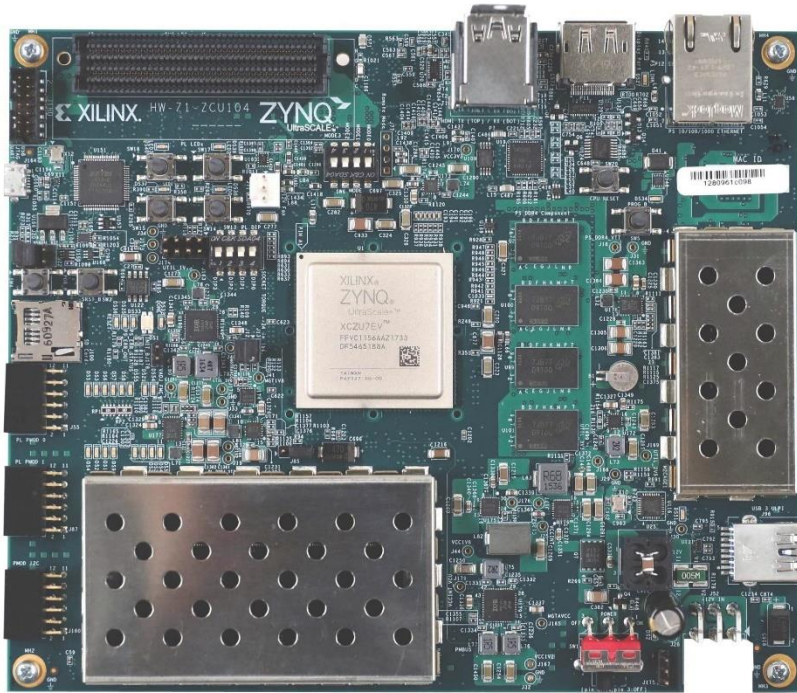


Figure 3.1.iii: ZCU104 Platform

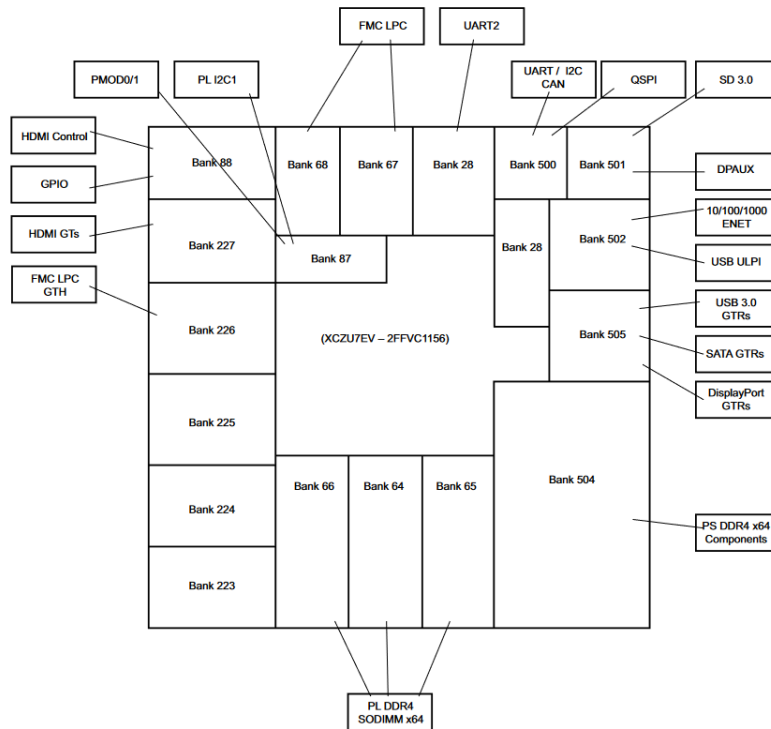


Figure 3.1.iv: ZYNQ UltraScale+ MPSoC ZCU104 Block Diagram

3.2 Basic Tools

The tools that were used for the implementation of the design, the bitstream generation, the Bare Metal application are the following.

3.2.1 Xilinx Vivado Design Suite 2020.2

The Xilinx® Vivado® Design Suite enables implementation of the following Xilinx device architectures: Versal™ adaptive compute acceleration platform (ACAP), UltraScale™, UltraScale+™, and Xilinx 7 series FPGA. A variety of design sources are supported, including:

- RTL designs
- Netlist designs
- IP-centric design flows

shows the Vivado tools flow.

Vivado implementation includes all steps necessary to place and route the netlist onto device resources, within the logical, physical, and timing constraints of the design.

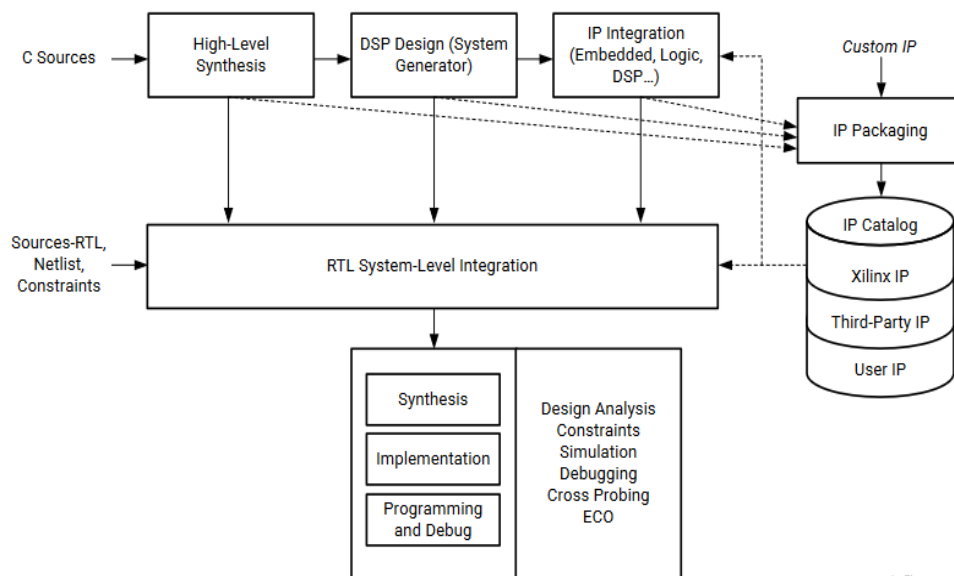


Figure 3.2.i: Vivado Design Suite High-Level Design Flow

3.2.2 Xilinx Vitis Unified Software Platform Documentation

For FPGA-based acceleration, the Vitis™ core development kit lets you build a software application using an API, to run hardware (HW) kernels on accelerator cards. The Vitis core development kit also supports running the software application on an embedded processor platform running Linux, such as on Zynq UltraScale+ MPSoC devices. For the embedded processor platform, the Vitis core development kit execution model also uses the

OpenCL API and the Linux-based Xilinx Runtime (XRT) to schedule the HW kernels and control data movement.

The Vitis software platform allows you to migrate data center applications to embedded platforms. The Vitis core development kit includes the v++ compiler for the hardware kernel on all platforms, the g++ compiler for compiling the application to run on an x86 host, and an Arm® compiler for cross-compiling the application to run on the embedded processor of a Xilinx device.

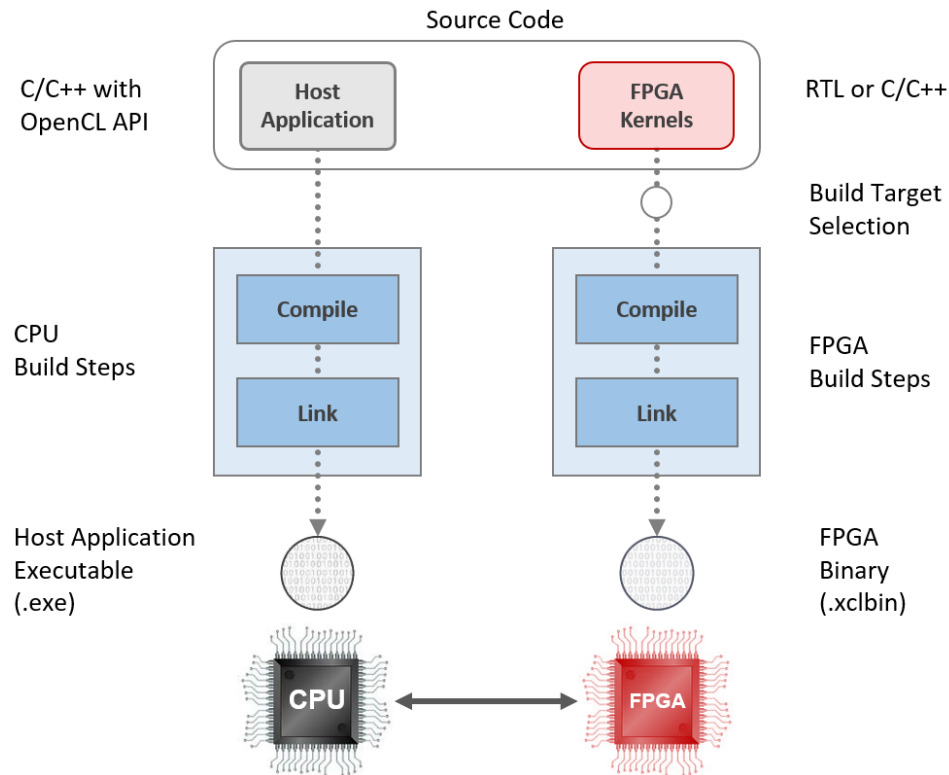


Figure 3.2.ii: Xilinx Vitis Software/Hardware Build Process

3.3 SCABox UI: Vitis Bare-Metal App and Python Corelation Analysis App

To control the design and emulate the adversary scenario we modified and used the open-source code of the SCABoxApp. It consists of 2 main parts, the C code for the Bare-Metal application that runs on the platforms' processing systems and a Python application that communicates with the design through serial connection and conducts the Corelation Power Analysis attack.

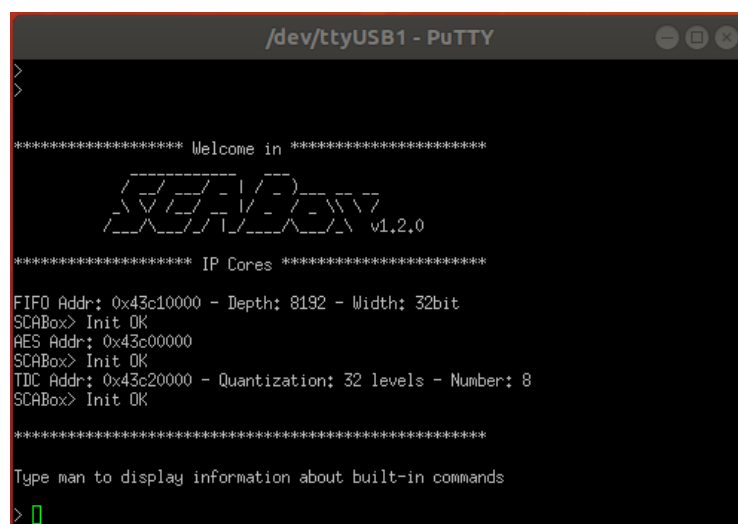
3.3.1 Vitis Bare-Metal Application

The Vitis Bare-Metal App runs on the processing systems of the boards. Its main purpose is to enable and control the UART communication between the user and the design. Through serial connection the user can send instructions to the PS and receive the outputs.

The hardware part that enables the communication of the PS with the rest of the design is the AXI protocol and the relevant implemented modules in the design.

The user can run a single AES encryption by entering a 128-bit text and a 128-bit key in hexadecimal format, or run multiple AES encryptions with random texts and a certain key and receive the ciphertext(s). While the AES encryption are occurring, the memory is enabled and stores sensor measurements. The number of measurements in each iteration can also be configured in the instructions.

The user can also enter instructions to get sensor values at any time, or read the FIFO data that are being displayed in a binary format.



```

/dev/ttyUSB1 - PuTTY

>
>

***** Welcome in *****

          /- - - \
         /  \  /  \
        /    \/    \
       /      \      \
      /        \        \
     /          \          \
    /            \            \
   /              \              \
  /                \                \
 /                  \                  \
/                    \                    \
\                    /                    /
 \                  /                  /
  \                /                /
   \              /              /
    \            /            /
     \          /          /
      \        /        /
       \      /      /
        \    /    /
         \  /  /
          \-/  /

***** IP Cores *****

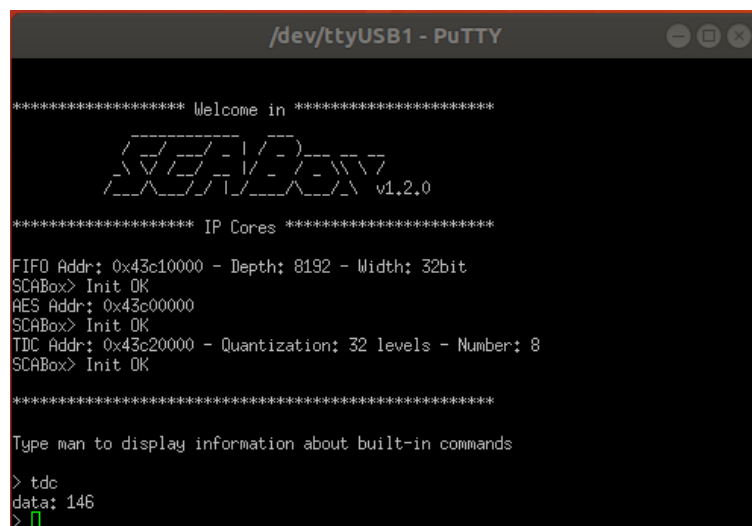
FIFO Addr: 0x43c10000 - Depth: 8192 - Width: 32bit
SCABox> Init OK
AES Addr: 0x43c00000
SCABox> Init OK
TDC Addr: 0x43c20000 - Quantization: 32 levels - Number: 8
SCABox> Init OK

*****

Type man to display information about built-in commands
>

```

Figure 3.3.i: SCABox Starting Screen



```

/dev/ttyUSB1 - PuTTY

***** Welcome in *****

          /- - - \
         /  \  /  \
        /    \/    \
       /      \      \
      /        \        \
     /          \          \
    /            \            \
   /              \              \
  /                \                \
 /                  \                  \
/                    \                    \
\                    /                    /
 \                  /                  /
  \                /                /
   \              /              /
    \            /            /
     \          /          /
      \        /        /
       \      /      /
        \    /    /
         \  /  /
          \-/  /

***** IP Cores *****

FIFO Addr: 0x43c10000 - Depth: 8192 - Width: 32bit
SCABox> Init OK
AES Addr: 0x43c00000
SCABox> Init OK
TDC Addr: 0x43c20000 - Quantization: 32 levels - Number: 8
SCABox> Init OK

*****

Type man to display information about built-in commands
> tdc
data: 146
>

```

Figure 3.3.ii: SCABox TDC Sensor Value Instruction


```

>
>
>
> sca -m hw -t 5 -s 0 -e 50
sensors: 8;;
target: 0;;
mode: hw;;
direction: enc;;
keys: 4df8cb69 70ba327f 2ebfb7d2 19a21a6b;;
;;
plains: 3eb907b7 68297977 280e3391 690bcaf1;;
ciphers: 69775ff8 32423e8d eed886a2 a474de6c;;
samples: 50;;
code: ]_ZaXYXcbe_ZRQUaYYV_[ ]WYZdcj0AIhsucPFM^jf^YXTZC!N;;
;;
plains: 170afbe4 50891219 33129d08 645905d4;;
ciphers: bb1ff969 774f4c74 424c72f1 c37faa5f;;
samples: 50;;
code: YVPuQZTRX_SRJPSWNTS[0TPTPVOSOWSTSWrf\4.HgtgN7<0!L;;
;;
plains: 6f5cf011 38e8aabb 3e17067f 5fa640b8;;
ciphers: 7995e6c4 dff75fcd 3d37b29f 47f4c27d;;
samples: 50;;
code: bd`dd`ebdc`S\be^YYe^XY^T_beN;:_ts_QBHRffdSTQXB!K;;
;;
plains: 47aee43f 2148435d 491b6ff6 5af37b9b;;
ciphers: b8244fd6 3b7c5821 e7e0bbcd d9031026;;
samples: 50;;
code: cjae f`ebjdbM^`a)a^c^Za[aZ_[^]b]bioM68Z)gQAJB!O;;
;;
plains: 2000d86c 09a7dbff 541fd96c 5640b67f;;
ciphers: 69f63a56 addcd123 edbe2bf6 5c941f57;;
samples: 50;;
code: fggidhchgidf]_ejcd_hfhbdi`ccjjm[JJ`pupfTYXjilF!H;;
;;
>
>
>
>

```

Figure 3.3.iv: SCABox Multiple AES Encryptions Instruction

3.3.2 Python Corelation Analysis Application

After the implementation of the design, the bitstream generation and the programming of the boards with the bare-metal application, the remaining part to complete a side-channel attack is to collect the required data so that the Correlation Power Analysis attack can be performed.

The Python application of the SCABoxApp runs on any computer connected to the board's UART connector via USB. It uses the instructions mentioned in [Section 3.3.1](#) to run multiple AES iterations in chunks and gather sensor measurements. After each chunk has concluded, it performs a Correlation Power Analysis attack on the last cycle of each AES iteration as shown in [Section 2.5](#) using the sensor's data and the outputted ciphertext. The key used to perform the attack is known, so we can observe when we have gathered sufficient amount of data for the attack to be successful.

The application has a graphic interface that displays plots of the Correlation Analysis. To conduct an attack, the user defines the number of AES iterations (n), the number of chunks(c) and the target of the attack. $n*c$ gives us the total number of iterations that will

be run in a single attack. The target of the attack can either be the USB port of the board that runs the design or a directory with stored binary files of previous attacks. Filtering can be also applied on the acquired data, but is not necessary for the attack to be successful.

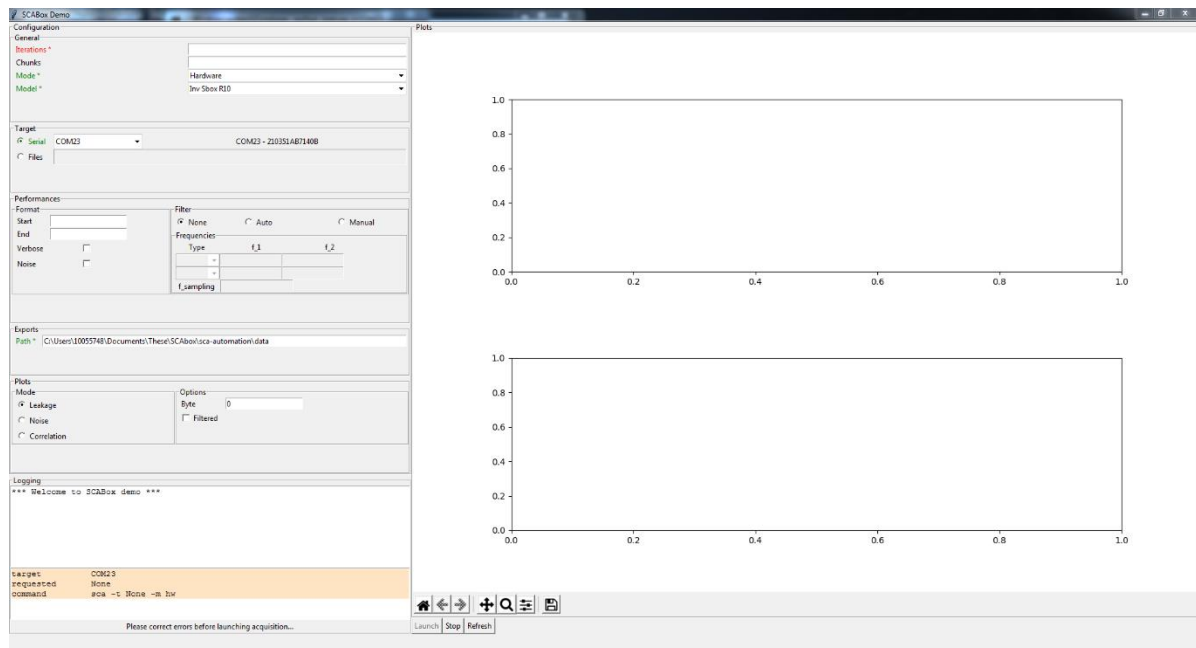


Figure 3.3.v: SCABox Python Application UI

3.4 The AXI protocol

The AXI is a point-to-point interconnect that designed for high-performance, high-speed microcontroller systems. The AXI protocol is based on a point-to-point interconnect to avoid bus sharing and therefore allow higher bandwidth and lower latency. AXI is arguably the most popular of all AMBA interface interconnect.

The essence of the AXI protocol is that it provides a framework for how different blocks inside each chip communicate with each other. It offers a procedure before anything is transmitted, so that the communication is clear and uninterrupted. That way, different components can talk to each other without stepping on each other. The procedure for the AXI protocol is as follows:

- Master & slave must “handshake” to confirm valid signals
- Transmission of control signal must be in separate phases
- Separate channels for transmission of signals
- Continuous transfer may be accomplished through burst-type communication

By working with the master and slave devices, the AXI protocol works across five addresses that include read and write address, read and write data, and write response.

Since each channel has its own unique signal, it can send the handshake response uninterrupted so that it can be received and put into order. That way, the channel that has priority will be responded to first and so forth. The source must provide a valid signal and one that gets a proper response from the receiver.

By having the transmission performed in separate phases, it allows the transfer of information to be performed in an orderly manner. This means that a handshake or agreement is reached first, then the information is moved from the source to the recipient. And that's how the AXI protocol works to move information between different sources without interference.

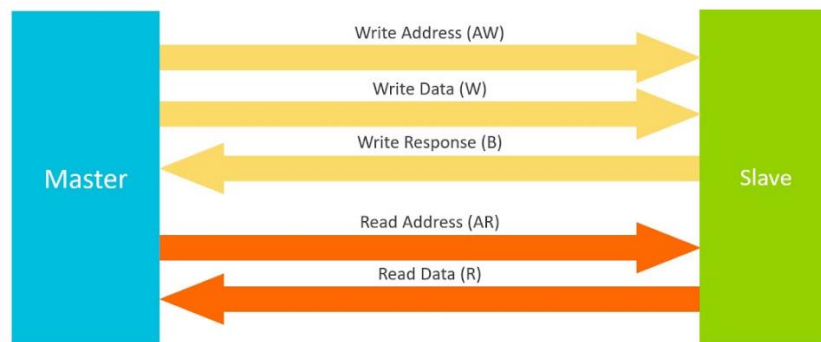


Figure 3.4.i: AXI Interconnect Block Diagram

3.5 UART serial connection

UART stands for Universal Asynchronous Receiver/Transmitter. In UART communication, two UARTs communicate directly with each other. The transmitting UART converts parallel data from a controlling device like a CPU into serial form, transmits it in serial to the receiving UART, which then converts the serial data back into parallel data for the receiving device. Only two wires are needed to transmit data between two UARTs. Data flows from the Tx pin of the transmitting UART to the Rx pin of the receiving UART.

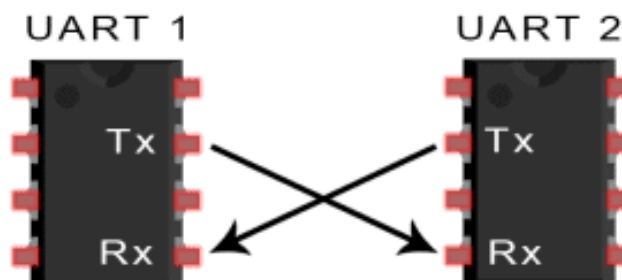


Figure 3.5.i: UART Connection Block Diagram

UARTs transmit data *asynchronously*, which means there is no clock signal to synchronize the output of bits from the transmitting UART to the sampling of bits by the receiving UART. Instead of a clock signal, the transmitting UART adds start and stop bits to the data packet being transferred. These bits define the beginning and end of the data packet so the receiving UART knows when to start reading the bits.

When the receiving UART detects a start bit, it starts to read the incoming bits at a specific frequency known as the *baud rate*. Baud rate is a measure of the speed of data transfer, expressed in bits per second (bps). Both UARTs must operate at about the same baud rate. The baud rate between the transmitting and receiving UARTs can only differ by about 10% before the timing of bits gets too far off. Both UARTs must also must be configured to transmit and receive the same data packet structure.

Chapter 4: System Architecture

In this chapter, there is an extensive description of the hardware designs that were implemented in this thesis, including the parameters and the thought process of the full design.

We emulate a multi-tenant scenario, where the fabric is shared among two users. A malicious user implements voltage fluctuation sensors to perform a Correlation Power Analysis (CPA) attack against a victim user's AES hardware module. The victim sends plaintext and receives the cyphertext, while the malicious user receives sensor measurements.

The attacker and victim modules are separated by columns of unused slices and DSP blocks. This ensures passive isolation between the users. Subsequently, we implement our Active Fence countermeasure between the two users and nearer the AES module. In both of the platforms, most of the implemented modules' code is identical. The hardware modules that are different between the platforms, are the implementations of the TDC sensors and the PS relevant modules. That is because the platforms use different architecture libraries. Finally, the resource utilisation of each platform is also different.

4.1 Emulation Assumptions

To consider the attack successful and achieve the key retrieval, we make some assumptions and compromises that do not contradict a possible real scenario. More specifically:

- The malicious user has access to the ciphertext. This is possible if a public channel is used by the victim for data transfers.
- The 128-bit key of the AES does not change throughout the attack.
- The AES module works with lower operating frequency than the sensor, so that there is more information for each AES cycle.
- The sensor readings are aligned exactly to each encryption. When a command is given to the AES for encryption, a start signal enables the memory to store sensor values, until a stop signal is sent at the end. This could be avoided using trace alignment techniques on the sensor traces.

4.2 Victim and Attacker Emulation

4.2.1 AES algorithm

On the victim's side, both platforms used the open-source, 128-bit AES core implemented in *SCABox* [29] by Gravelier et al ([13]). The module ciphers and deciphers 128-

bit words using a 128-bit key and produces a valid output every 11 clock cycles, one for loading the data and ten for each round of the AES. The 128-bit key is defined as a constant in the VHDL code, but can also be changed through the bare-metal application commands. An AXI wrapper is used for the communication of the module with the PS. Using serial communication, we send 128-bit plaintexts and receive the 128-bit ciphertexts.

4.2.2 TDC – sensor Bank

The implemented TDC sensor is different for each platform, because of the different architecture libraries that they use. The ZedBoard™ Uses the 7 series architecture library, while the ZCU104 uses the UltraScale+ architecture library.

4.2.2.1 ZedBoard™ sensor Implementation

The TDC sensor implemented on the ZedBoard platform consists of an initial delay line of 16 open latches and 16 LUTS acting as buffers, placed alternately, followed by 16 more LUTS/buffers. The initial delay line is driven by the clock signal and drives the input of the observable/sampled delay line.

The observable delay line consists of a carry chain line of CARRY4 primitives included in the 7 series architecture library. Furthermore, registers at each byte output of the carry chain sample the delay line. CARRY4 primitives are used for carry chain counters, adders and subtractors. They consist of 4 multiplexers in line as shown in *Fig.4.2.i*. They are useful to implement a long chain of buffers with small area overhead and are also more sensitive in voltage fluctuations than LUTS and Latches, thus they offer a good choice for TDC sensors. 8 CARRY4 primitives and 32 registers create a 32-bit wide observable delay line of the TDC sensor, enough to capture the magnitude of the voltage transitions occurring. The registers are driven by the clock signal that also drives the initial delay line.

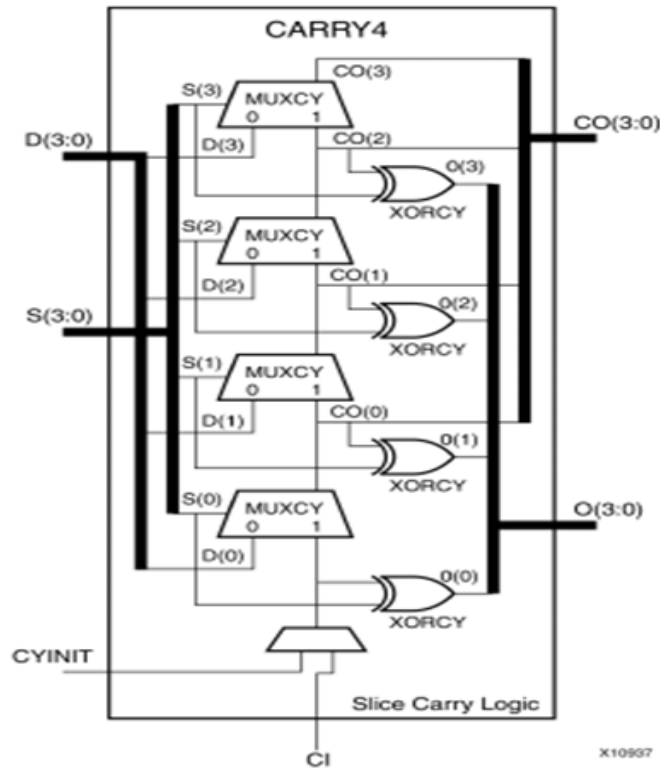


Figure 4.2.i CARRY4 primitive

4.2.2.2 ZCU104 sensor Implementation

The TDC sensor implemented on the ZCU104 board consists of an initial delay line of 64 open latches and 64 LUTs acting as buffers, placed alternately, followed by 64 more LUTs/buffers. The initial delay line is driven by the clock signal and drives the input of the observable/sampled delay line.

The observable delay line consists of a carry chain line of CARRY8 primitives included in the UltraScale+ architecture library. Furthermore, registers at each byte output of the carry chain sample the delay line. They consist of 8 multiplexers in line as shown in *Fig.4.2.ii*. They have the same use as the CARRY4 primitive used in the ZedBoard design. They can also be configured as 2 CARRY4 modules. 8 CARRY4 primitives and 32 registers create a 32-bit wide observable delay line of the TDC sensor, enough to capture the magnitude of the voltage transitions occurring. The registers are driven by the clock signal, shifted 45°.

In both platforms, the sensor bank consists of 8 TDC sensors followed by an adder. Each sensor has a range of 32 values. The adder encodes the output binary value of each adder to the corresponding decimal value and adds them. The final output of the bank drives the data input port of the FIFO memory.

The TDC sensors require different implementation in each platform, because of the different fabric architectures. The delays of each board's primitives differ; thus, the sensors'

initial delay lines require different length to produce similar result. In the same manner, the registers of the observable delay line use different clock signals.

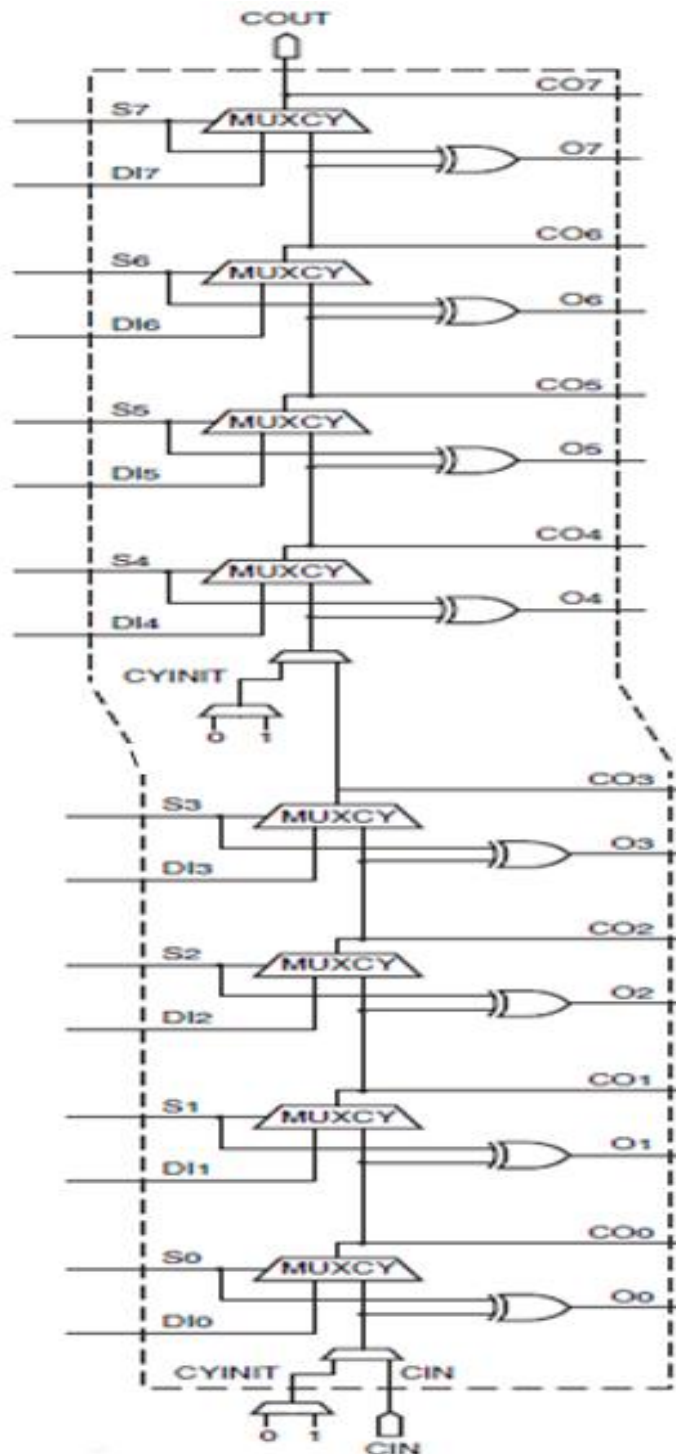


Figure 4.2.ii CARRY8 primitive

4.2.3 Memory

A hardware FIFO memory, generated with the *Vivado Memory Generator* tool, is used to store the TDC sensor measurements. The memory has a size of 270 Kbits. One AES

iteration corresponds to 500, 32-bit TDC sensor measurements, thus it can store up to 17 AES iterations before being full.

The memory is being controlled by simple FSM machine, as shown below.

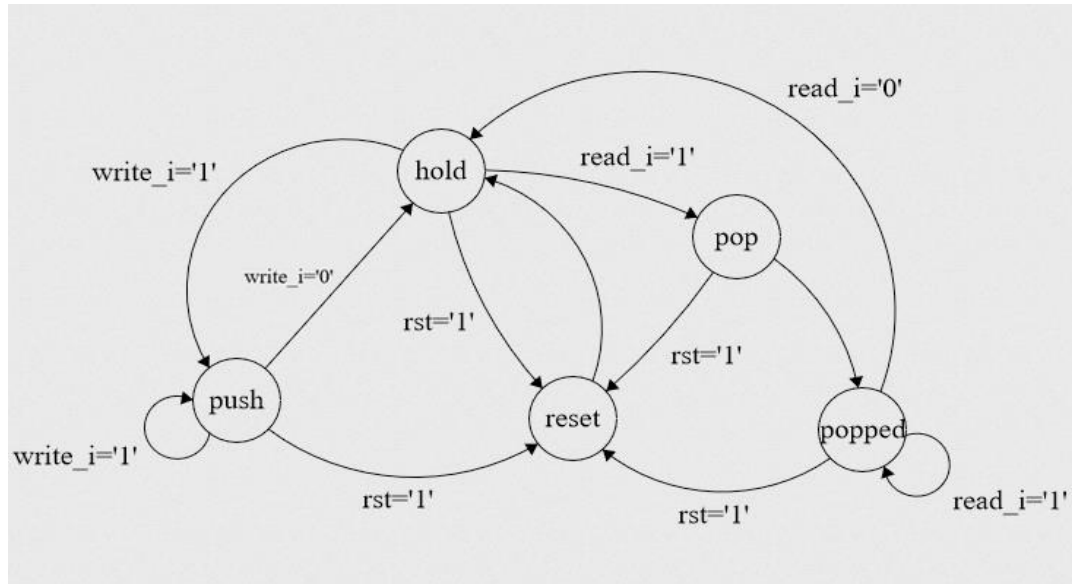


Figure 4.2.3: Memory Controller FSM

4.2.4 Active Fence Architecture

Our Active Fence is implemented using banks of ROs. LUT2 primitives are used as 2-input NAND gates to implement the RO with an *enable* signal. Moreover, we map RO banks between the attacker and the victim, placing them in a densely packed uniform array, as this represents the most effective way[12].

As previously mentioned, the goal of the Active Fence proposed in [12] was to neutralise the effect of the victim's core on the instantaneous power consumption. This, nonetheless, can potentially result in relatively big area overheads, depending on the algorithm that needs protection since the Active Fence needs to occupy as many resources as the victim's algorithm, in order to have an equally strong influence on the PDN. Our focus is now different since we are trying to establish the level of effectiveness for various fence sizes, all with smaller number of resources occupied than the module under protection, for increasing noise levels. In our case the AES-core occupies 4271 CLBs as LUTs, CARRYs and MUXs and 2026 registers. Our work tests the fence using six different configurations, with 1024, 2048, 3072, 4096, 5120 and 6144 LUTs as ROs divided in 16 banks, using 64, 128, 192, 256, 320 and 384 ROs per bank respectively.

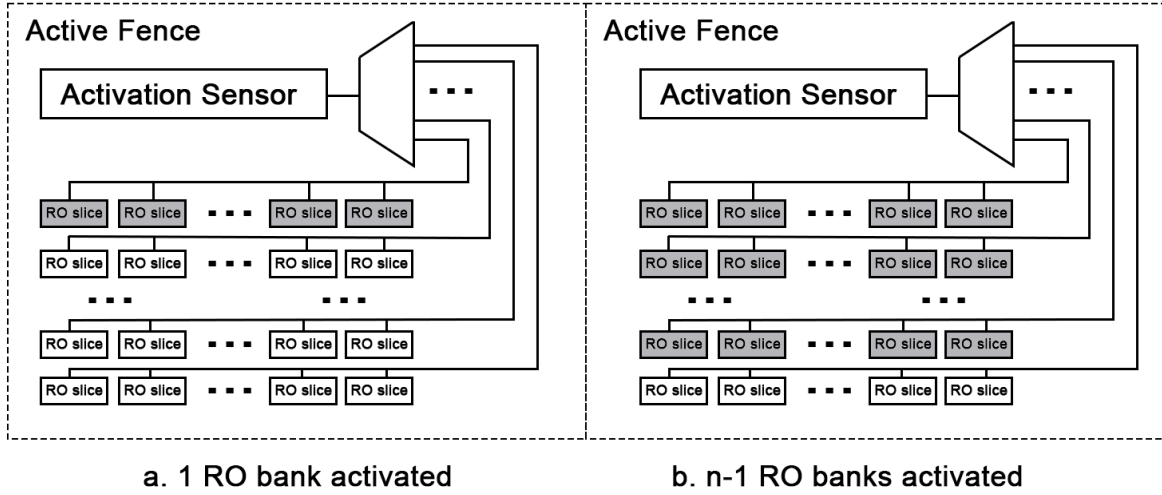


Figure 4.2.iii: Active Fence Block Diagram

A single RO is used to control our proposed Active Fence countermeasure. Although ROs have a lower resolution than TDCs as power sensors, they use significantly less resources [2]. Therefore, in this instance, where we are interested in introducing noise injection and not in hiding through the power consumption matching between the active fence and the protected module, we prefer an efficient countermeasure with a low cost of resources. That is the reason why we do not need a high-resolution sensor for controlling the fence. Despite the lower resolution, a single RO remains sensitive to adequate transitions in power consumption and can, therefore, effectively control the fence activation. Additionally, the quantisation error of a single RO can work in favour of this goal, meaning that the variable frequency can lead to more unpredictable activation patterns and, thus, in random noise injection.

We measure the RO's frequency using an 8-bit Libaw–Craig ring counter. The sensor uses only one LUT for the RO, eight flip-flops for the counter and an 8-bit register. The output register drives an encoder with sixteen possible values while the ROs of the fence have been divided into sixteen banks. It is worth noting that despite the fact that the sensor has a range of sixteen values, it is not necessary for all of them to be reached. That is a function of the power consumption transitions and their variance, meaning that there may be instances where this variance does not become high-enough to lead to the activation of all sixteen sensor values. As a result, some of the RO banks may never be activated or deactivated, however, this does not constitute a limitation to our results.

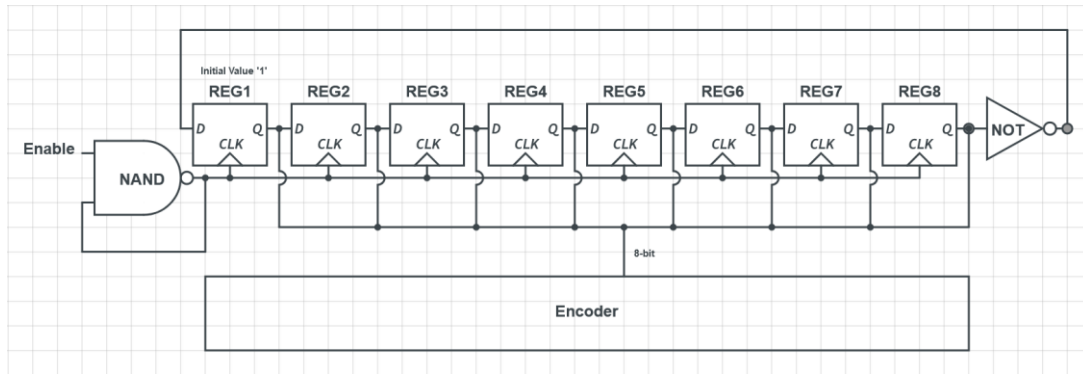


Figure 4.2.iv: Ring Oscillator Sensor with Libaw-Craig Counter

4.2.5 Design Clocking

The design implemented on the ZedBoard platform is driven by three clock domains that are outputted by the PS, while the design implemented on the ZCU104 is driven by four clock domains, that are outputs of a clock manager that is driven by the 100MHz PS main clock output. A 50 MHz clock drives the AXI interconnect and a 10 MHz clock is used for the AES core, for both platforms. Also, a 200 MHz clock drives the sensor delay line. In the ZCU104 platform, a 45° shifted 200 MHz clock drives the sensor's output register and the memory. Finally, the UART baud rate for the serial communication is set to 115200 for the ZedBoard, and 460800 for the ZCU104.

4.2.6 Remaining Modules

The remaining modules include the PS IP Core and the AXI protocol modules included in the Vivado IP library for each platform. The AXI modules enable the communication between the hardware modules and the PS, while the PS core enables the UART communication of the design, outputs the main clock(s) and runs the bare metal application of Vitis.

All the hardware modules were tested with the Vivado simulation tool separately to validate their correct functionality before being combined in a single design.

The remaining part of this chapter displays the Block Diagram, the FPGA floorplan and the resource utilization of the design for each platform.

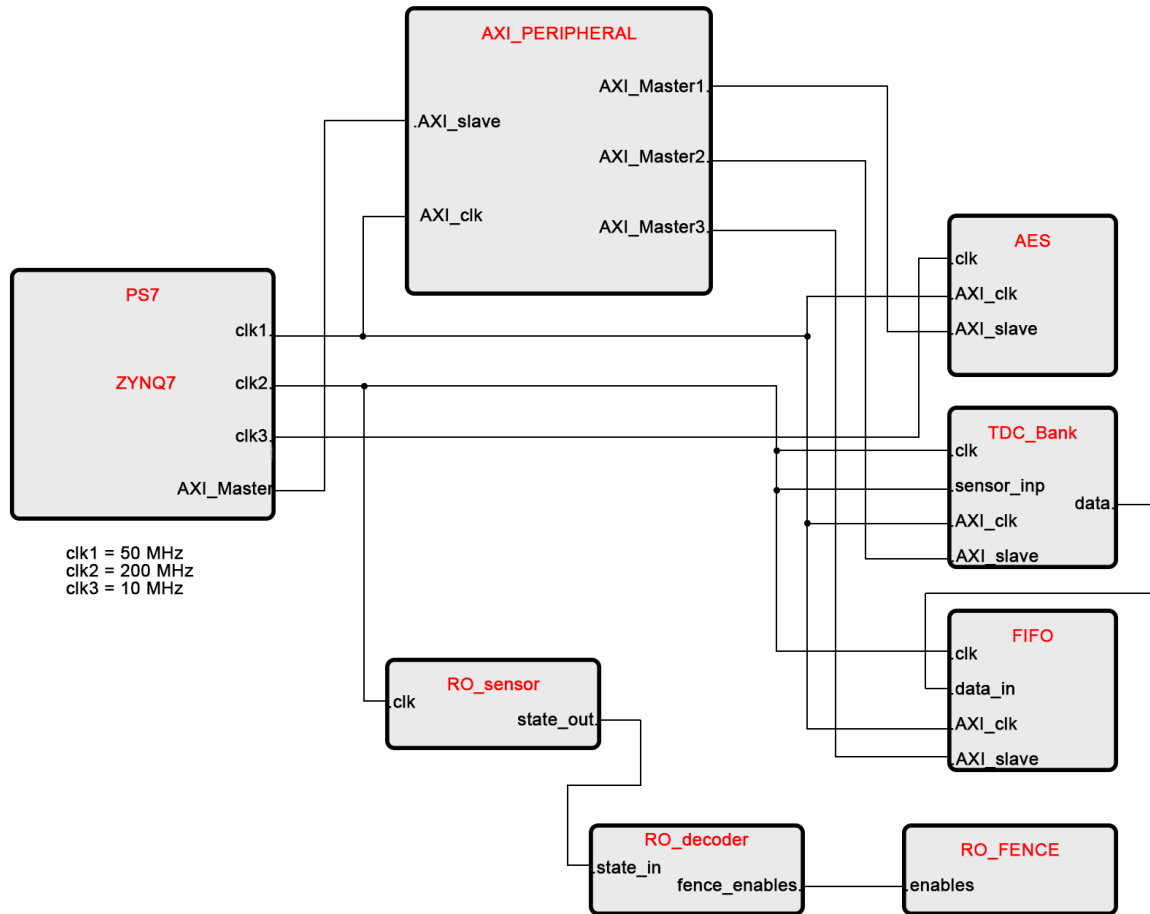


Figure 4.2.v: ZedBoard Design Block Diagram

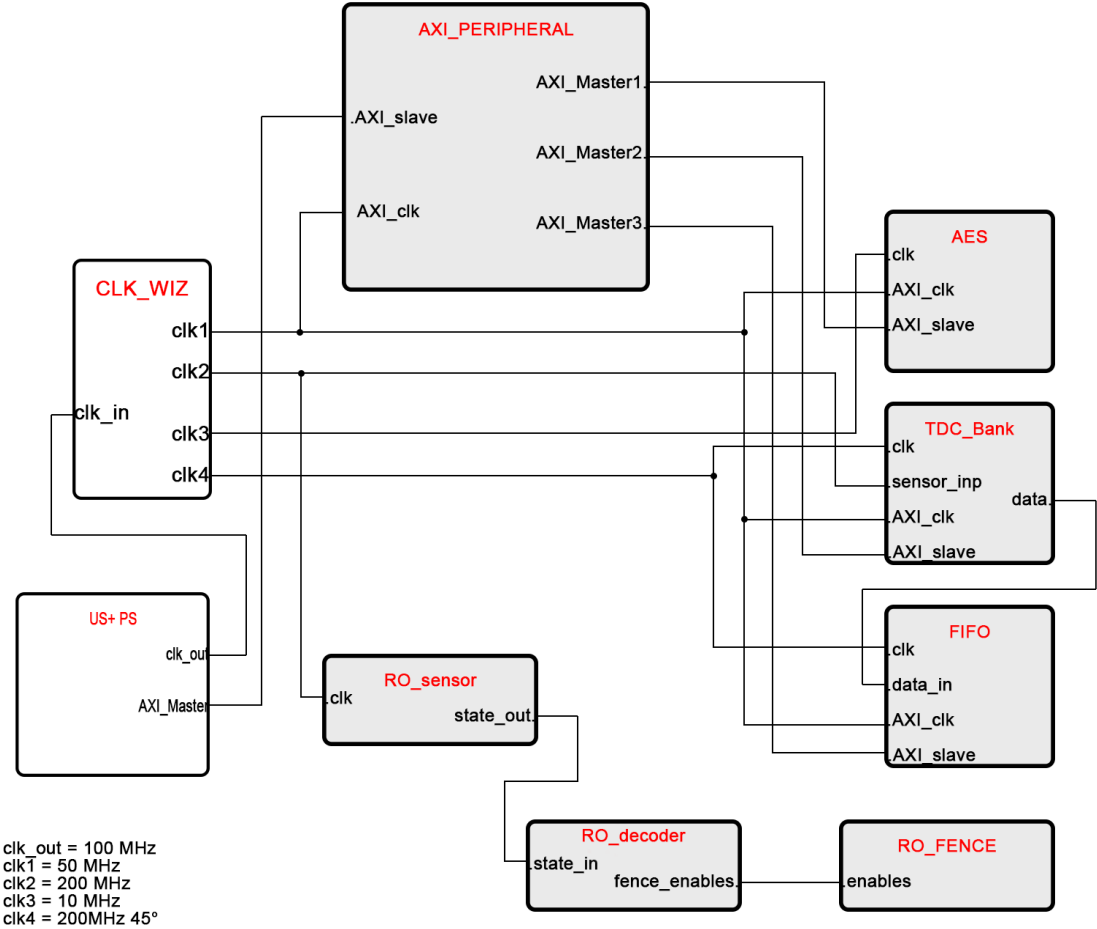


Figure 4.2.vi: ZCU104 Design Block Diagram

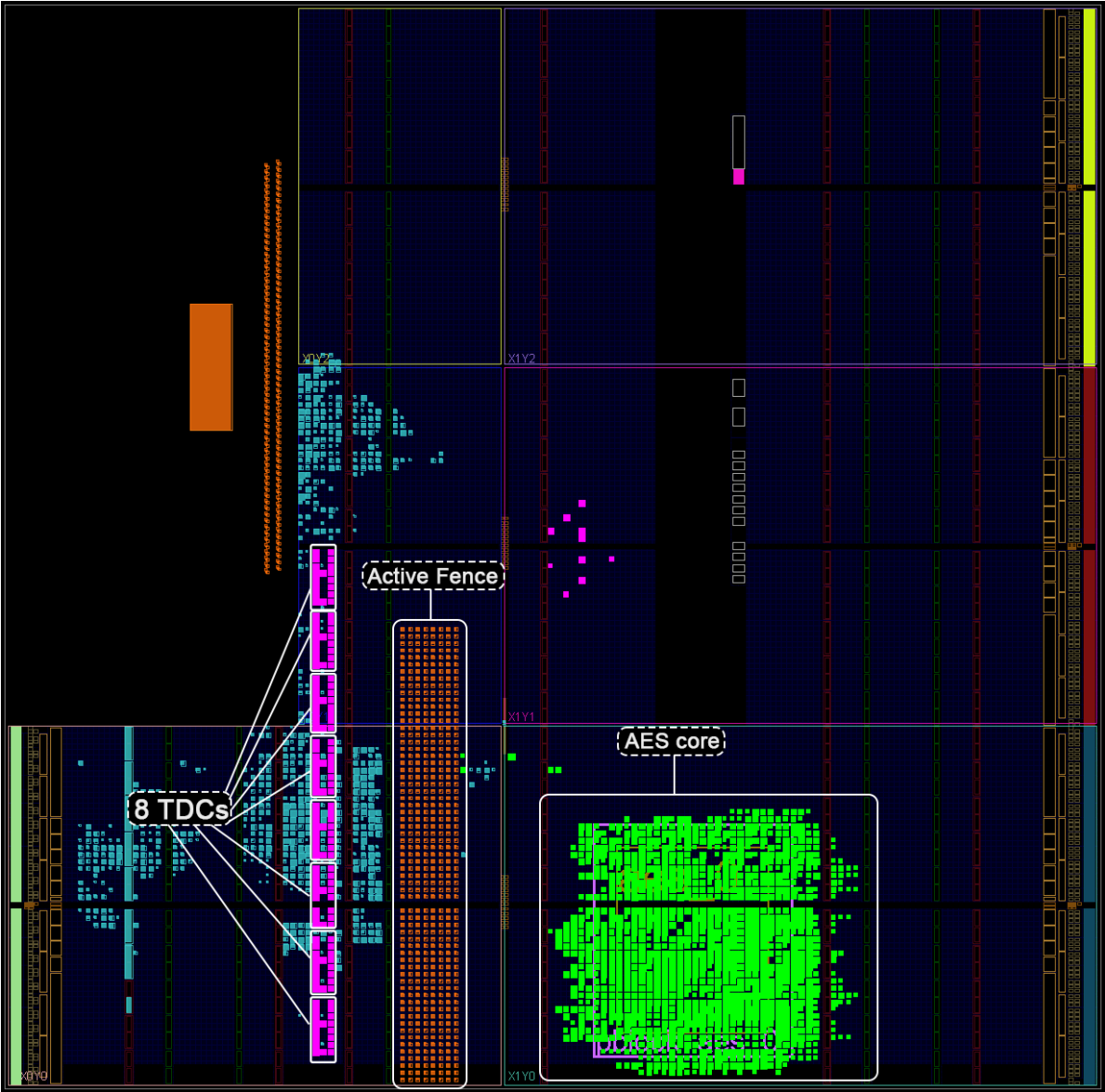


Figure 4.2.vii: ZedBoard Floorplaning

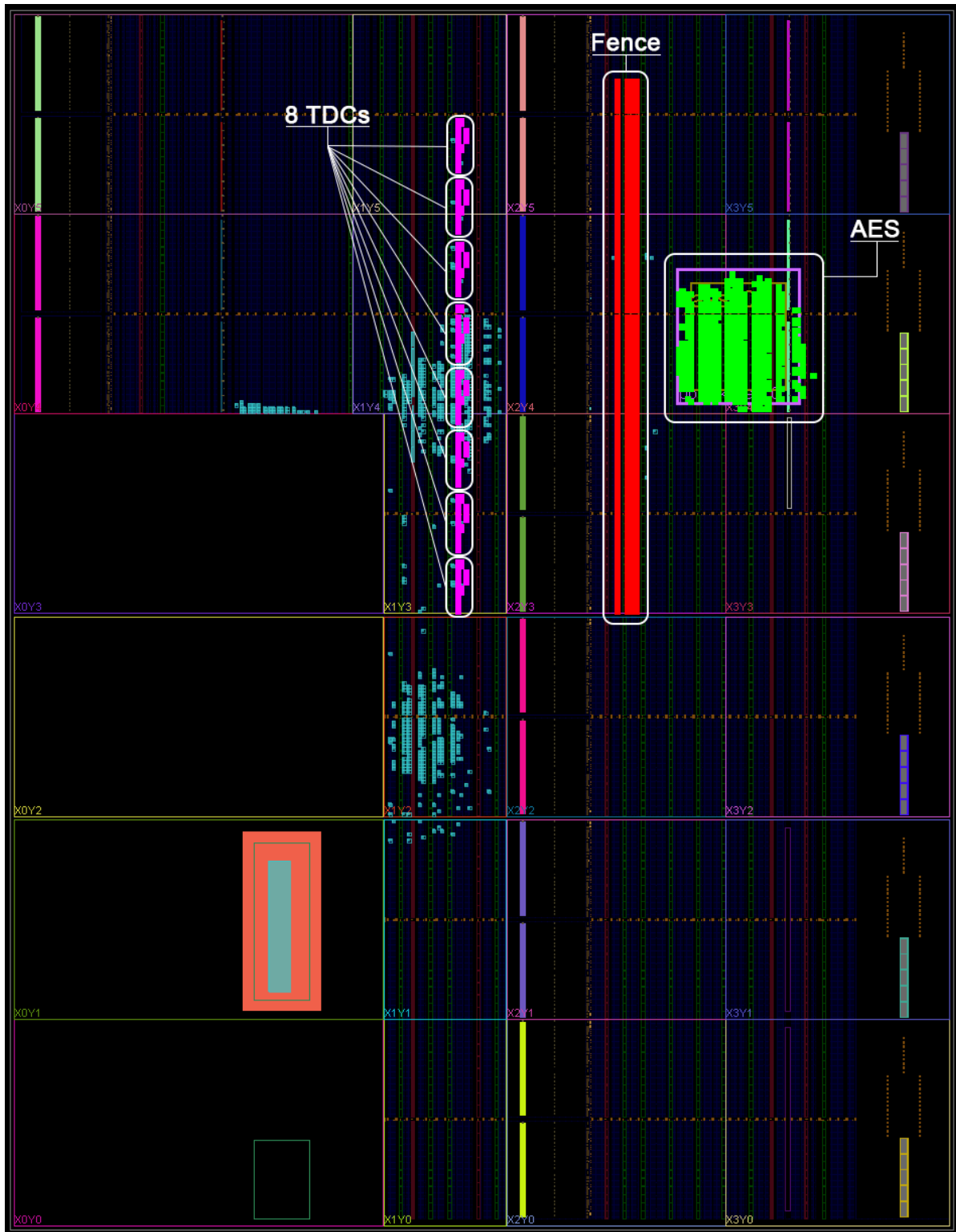


Figure 4.2.viii: ZCU104 Floorplaning

0	Module	FLOP_LATChes	CARRYs	LUTs	MUXFs	CLOCK	BMEM	DMEM	IO	CLK	HARD_IP
1	AES	2025	11	3324	942	0	0	0	0	0	0
2	MEM	238	26	183	0	0	7.5	0	0	0	0
3	MEM ctrl	3	0	6	0	0	0	0	0	0	0
4	TDCs bank	1633	79	1325	208	0	0	0	0	0	0
5	RO sensor	0	0	2	0	0	0	0	0	0	0
6	Active Fence	0	0	1024/ 2048/ 3072/ 4096/ 5120/ 6144	0	0	0	0	0	0	0
7	PS7 AXI Periph	610	12	589	2	0	0	65	0	0	0
8	RST PS7	33	0	20	0	0	0	1	0	0	0
9	Processing system7	0	0	0	0	0	0	0	130	3	1

Table 4.2-1: ZedBoard Design Total Resource Utilization

#	Module	CLBs as					REGISTERS as		CLOCK as	
		RAM36	CARRYs	LUTs	MUXFs	SRLs	SDRs	LATChes	BUFFER	PLL
1	AES	0	6	3323	942	0	2026	0	1	0
2	MEM	7.5	13	185	0	0	238	0	0	0
3	MEM ctrl	0	0	6	0	0	3	0	0	0
4	TDCs bank	0	38	2506	640	0	1505	512	0	0
5	RO sensor	0	0	2	0	0	16	0	0	0
6	Active Fence	0	0	1024/ 2048/ 3072/ 4096/ 5120/ 6144	0	0	0	0	0	0
7	PS8 AXI Periph	0	8	1461	2	45	1393	0	0	0
8	RST PS8	0	0	19	0	1	34	0	1	0
9	CLK WIZ	0	0	0	0	0	0	0	4	1
10	ZYNQ PS	Occupies 1 ADVANCED PROCESSOR							1	0

Table 4.2-2: ZCU104 Design Total Resource Utilization

Chapter 5: Experimental Procedure

In this chapter, there is a step-by-step description of the experimental process.

As a first step we use the VHDL and Verilog code of the modules, the Vivado IP library and the *Vivado connection automation tool* to create a Block Design in the Vivado 2020.2. After the simulation, the synthesis and the implementation of the design, we generate the bitstream and export the hardware specification (.xsa) file.

Using the Xilinx Vitis 2020.2 we import the .xsa file exported by Vivado to generate the correct device platform and create an application project that runs on this platform. We import the .c code files for the Bare-Metal application, define the needed libraries and directories for the compiler and build the Vitis project. If there are no errors, we establish connection with the board connected to the computer and download the application and the design to the board.

At this point, we have created and downloaded a working setup to emulate a two-tenant power Side-Channel attack scenario. By connecting to the board's serial port, we test the basic instructions to see if the design is fully functional.

Lastly, we run the Python application. We define the number of AES iterations, the number of chunks and the serial port of the board as the target to conduct a Correlation Power Analysis. We then observe and evaluate the results. We repeat the whole process for each active fence configuration in both boards.

We conduct each attack multiple times, as the results may differ, especially when the active fence countermeasure is present. We take into account the worst-case scenario, meaning the attack that is successful with the least acquired traces.

Chapter 6: Results and Evaluation

In this chapter, we present and evaluate the results of each platform. Then we compare the results between the platforms and between similar works. The plots analysed in this chapter are the SCABox python application plots and plots made by the results of the conducted attacks. The SCABox python application displays three plots that are useful for our conclusions:

- *The Quantification Vs. Time Samples plot*, which displays the temporal average of the TDC measurements, i.e., the time samples, of all the AES iterations
- *The Pearson's Correlation Vs. Traces Acquired plot*, which displays the Pearson's Correlation of the 256 possible values of a byte, in relation to the number of the AES iterations, i.e., the traces that have been collected
- *The Pearson's Correlation Vs. Time Samples plot*, which displays the Pearson's Correlation of the 256 possible values of a byte, in relation to the temporal average of the TDC measurements, i.e., the time samples of the AES iterations.

The emulation experiments implemented on the two platforms operate on the premise that a malicious entity uses the intercepted ciphertext to create a power consumption hypothesis for each key byte value per AES encryption. Then, by gathering TDC sensor data, it becomes possible to calculate the correlation between each hypothesis and the actual measurements, selecting the value with the highest correlation as the most probable to be true. The configuration of an Active Fence of ROs between two adversary users adds noise to the sensors' readings, which increases the size of data that needs to be acquired and processed and, subsequently, the time for the most probable value to reach the true key byte.

Based on the above, we implemented six different Active Fence configurations, using a different number of ROs each time, i.e., 1024, 2048, 3072, 4096, 5120 and 6144. These numbers correspond to approximately 17%, 33%, 50%, 67%, 83% and 100% of the total AES module resources, respectively. The attacks for each configuration were ran multiple times, measuring the minimum number of the AES iterations, i.e., the least data needed to be gathered for the most probable byte value hypothesis to reach the true key byte value.

6.1 ZedBoard Platform Results

We first need to verify that our design is fully functional, meaning that we can conduct a successful, remote, power side-channel attack against the AES hardware core without any countermeasures.

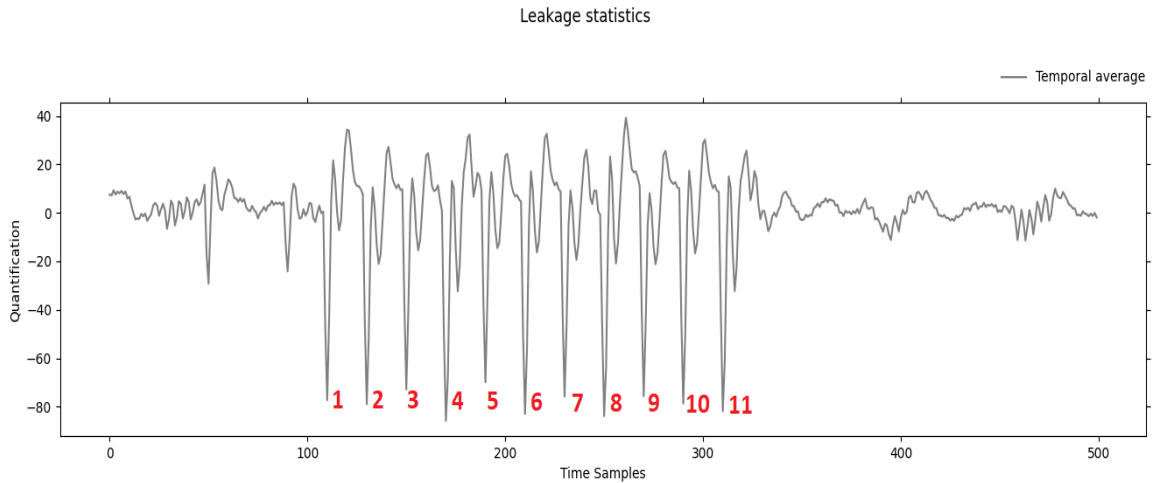


Figure 6.1.i: Quantification vs. Time Samples. Average of 500 AES Iterations without countermeasure (ZedBoard)

By observing the plot of Figure 6.1.i we see that there are 11 distinct spikes, one for every AES cycle. The measurements between those spikes show a similar, repeated pattern. This is due to the same calculations that happen at each AES cycle and the divergence between them is because of the different data that are being processed. This means that the AES functionality is capturable by the TDC sensors and that each time the AES 128-bit output register is overwritten at the end of each cycle, a high voltage drop occurs and is captured by the sensor.

Although the cryptographic module's behaviour is observable in the plot of the TDC sensor measurements, this does not mean that the statistical evaluation of the data can successfully lead to the extraction of the key. Multiple AES iterations need to be captured for the Correlation Power Analysis to have a good result for any of the key's bytes.

The first byte of the key that is successfully extracted, is the second byte (byte 1 of 15). The value of the second byte, 0x32, has the highest correlation amongst the other 255 possible values after almost 2000 captured AES iterations, but it becomes clear that this is the most possible true value after 3500 captured AES iterations, as shown Figure 6.1.ii: Pearson's Correlation Plots for the Second Key Byte. We see that after the 3500 AES iterations/traces point, its correlation has diverged enough from the correlation of the other values.

The first byte that is extracted will be the same only if the key and the ciphered text are the same for every run. For our experiments, we use the same key, but randomly generated texts as input of the AES, thus the sequence of the extracted bytes may differ.

After the first, the rest of the bytes are also extracted as more traces are acquired. We focus only on the number of traces of the first extracted byte because it is the least useful amount of data that the malicious user needs to collect for the attack to have effect.

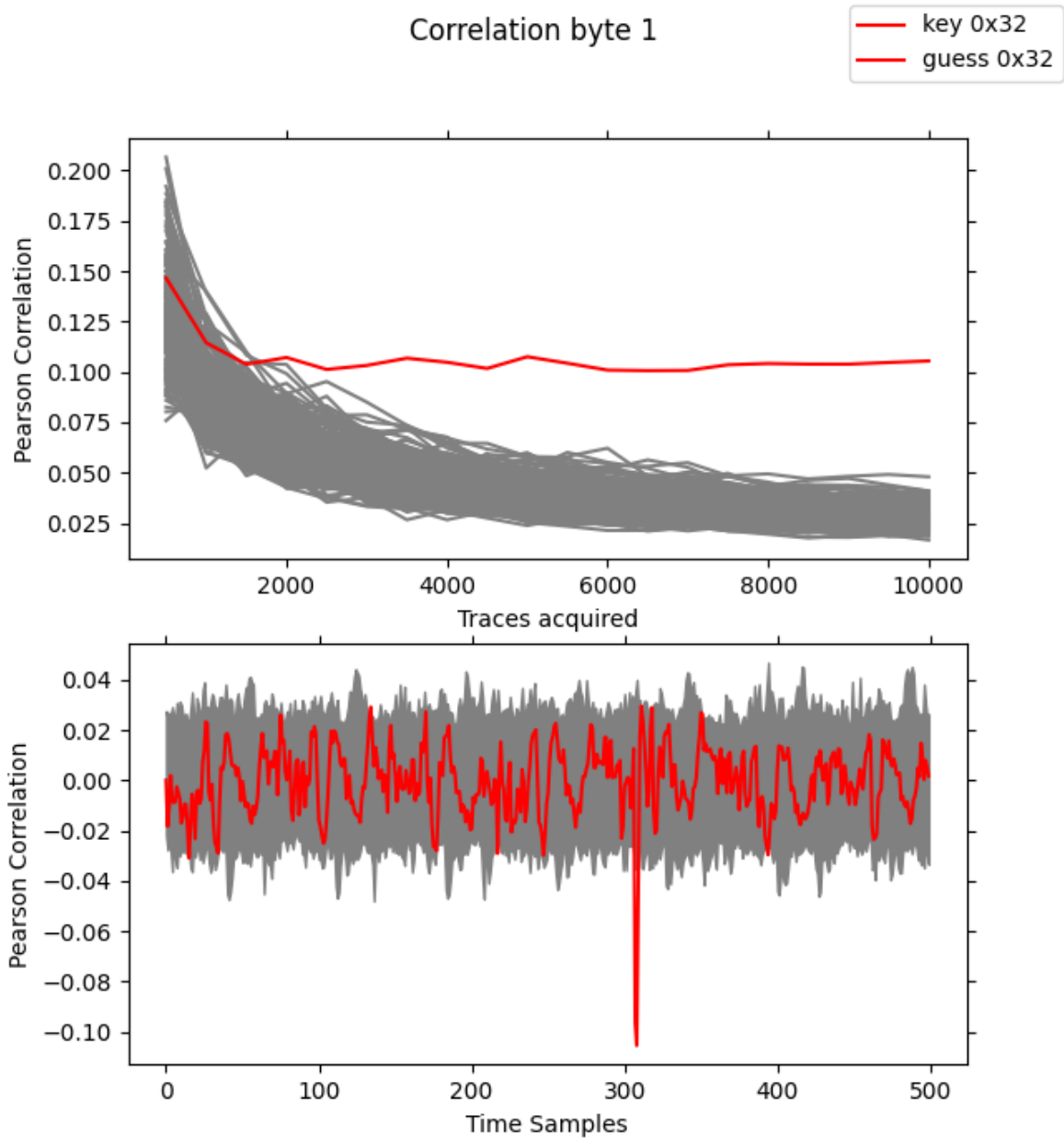


Figure 6.1.ii: Pearson's Correlation Plots for the Second Key Byte with no Active Fence

For the implementation without any countermeasures, the first byte was extracted at 3500 acquired AES iterations, i.e., 1750000 TDC measurements, which corresponds from 5.72 to 6.68MB of sensor data, 46.875KB of plaintexts and 46.875KB of ciphertexts.

We add the Active Fence countermeasure and observe its effect on the design by conducting the attack multiple times. We increase the total number of ROs by 1024 for each configuration. We first evaluate the active fence with 1024 ROs

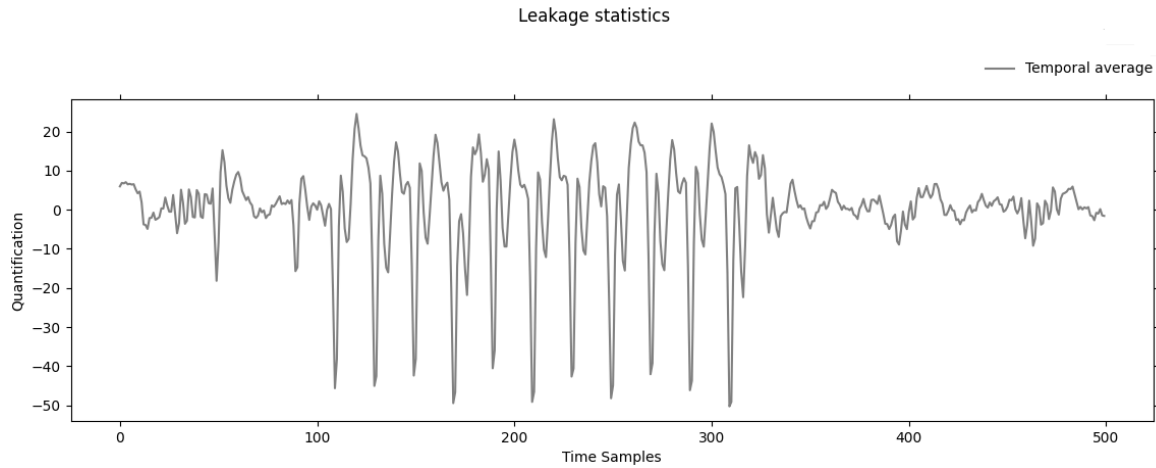


Figure 6.1.iii Quantification vs. Time Samples. Average of 500 AES Iterations with Active Fence (ZedBoard)

If we compare the Quantification vs. Time Samples plot of the Active Fence (Figure 6.1.iii) with the one with no countermeasure present two observations emerge. The first is that the magnitude of the plot for the design with the Active Fence is lower. This means that the countermeasure is successful at hiding the functionality of the AES core on some level, not quantifiable by this plot though. By enabling and disabling the ROs, the Active Fence influences the PDN, and makes the power consumption transitions of the AES less detectable.

The second observation, is the high frequency behaviour that occurs between the high magnitude spikes, that is not present in the design without the countermeasure. This behaviour translates to noise in the sensor's readings by the Active Fence. The high frequency oscillation of the ROs injects noise to the system that interferes with the sensor's measurements. We conduct the power side-channel attack with the 1024 RO Active Fence. The first byte that is extracted with the least acquired AES iterations is the fourth byte, as shown in Figure 6.1.iv. The correct byte value is extracted after 12500 AES iterations have been acquired.

By comparing the results with the Pearson's Correlation plots of the design with no countermeasure, we again can make three significant observations. The first and most obvious, is that the needed iterations for the most probable value to reach correct key byte value have risen to 12500, making the attack at least 3.57x harder to mount.

The second thing we can observe by comparing the plots is that the correlation of the correct byte value is lower when the Active Fence is present in the design. Also, the divergence between the values' correlation is smaller, making the most probable value to be less certain to be the true value.

Both the above observations emerge from the Pearson's Correlation Vs. Traces Acquired plots. The third observation is shown in the Pearson's Correlation Vs. Time Samples Plot. The Time Sample with the highest Correlation has a lower correlation value than the one in the design without the countermeasure. This again means that the time sample with the highest correlation is less probable to be the correct one.

The 12500 iterations correspond to 23,84MB of the TDC measurements, 195.31KB of plaintext and 195.31KB of ciphertext.

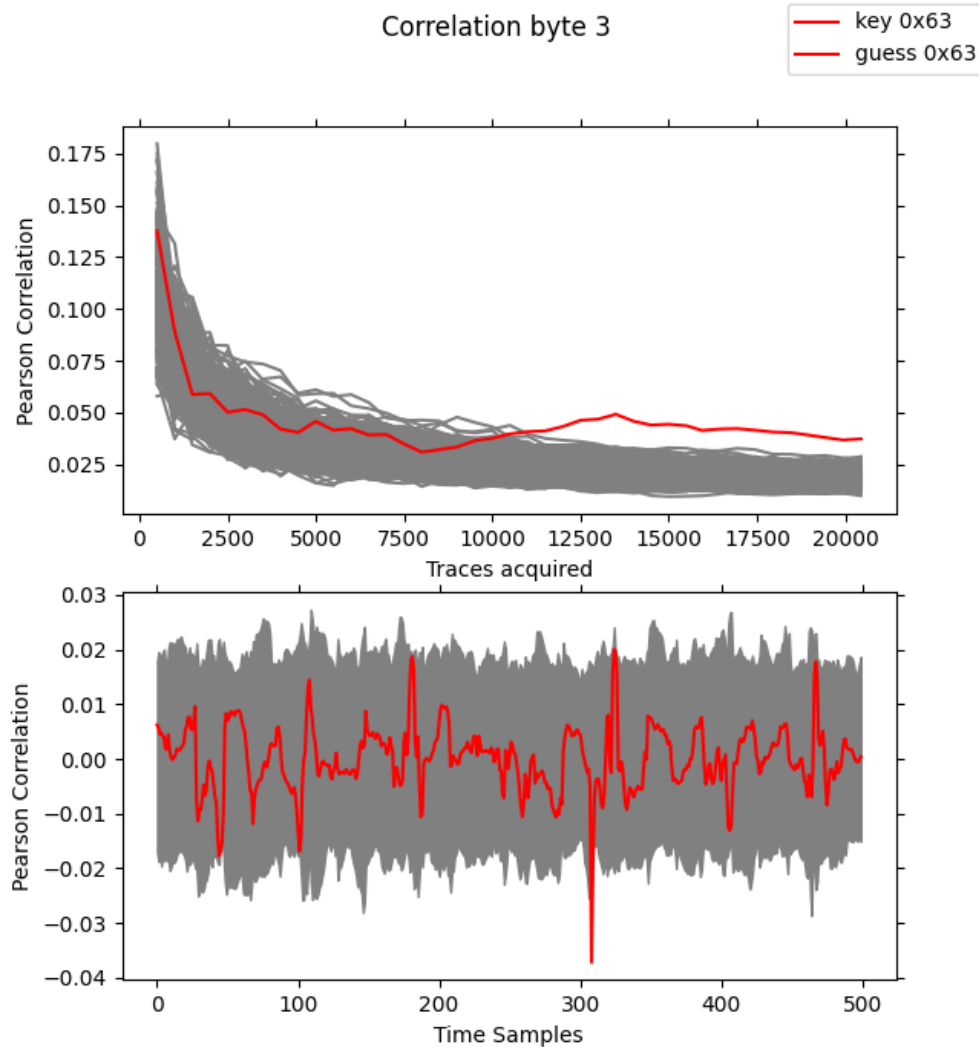


Figure 6.1.iv: Pearson's Correlation Plots for the Fourth Key Byte with 1024 RO Active Fence

We present the plots of one more Active Fence configuration, as the observations that are to be made are concluded by these three implementations. With 4096 RO Active Fence, the correct key byte value of the first extracted byte, reaches the highest correlation at 94000 collected AES iterations.

In this implementation the byte value and time sample with the highest correlation have a really small gap from the other byte values and time samples. This means that while we know that the byte value with the highest correlation is the correct one, an attacker in a real scenario cannot be certain that he has extracted the correct key byte value.

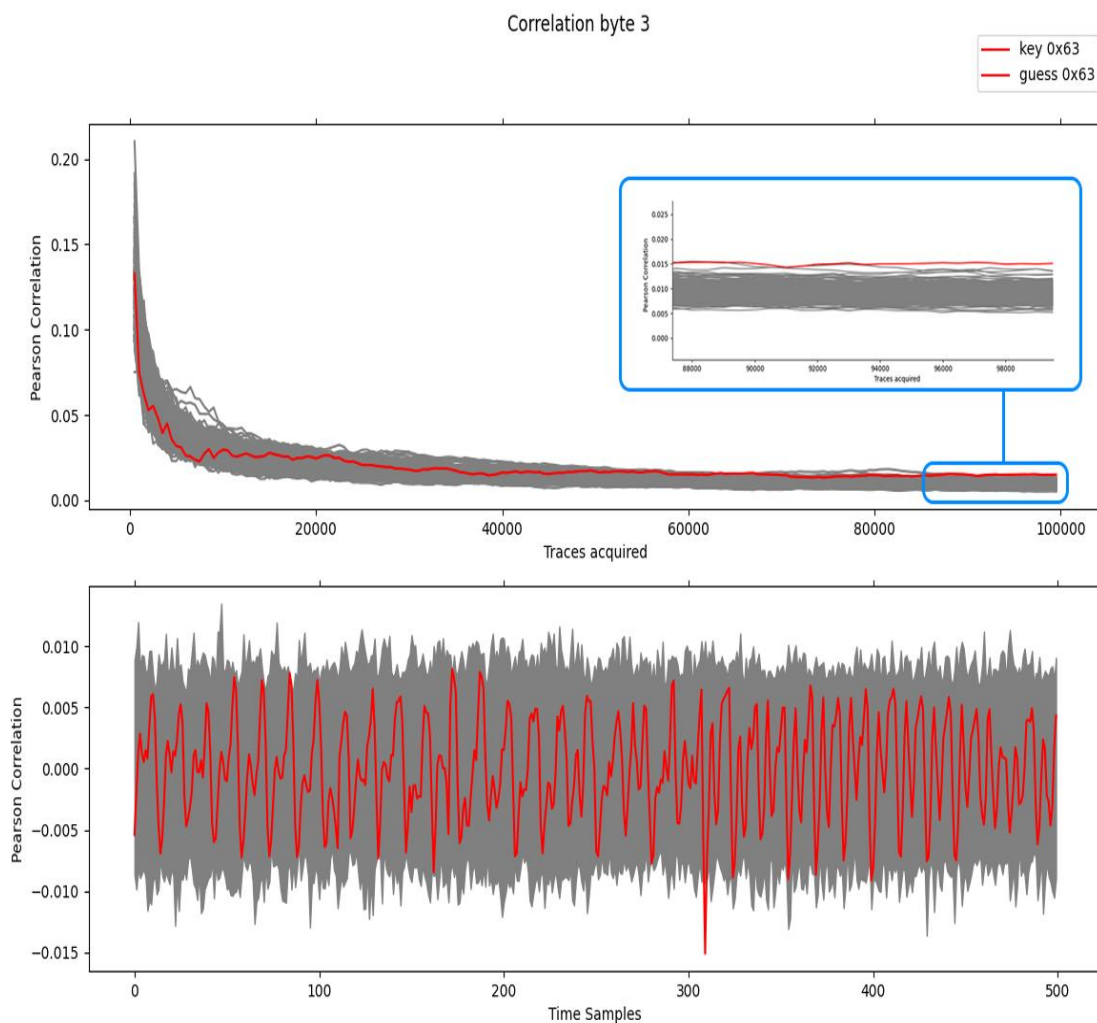


Figure 6.1.v: Pearson's Correlation Plots for the Fourth Key Byte with 4096 RO Active Fence

We avoid to map the last two Active Fence configurations on the ZedBoard Platform. The parallel activity of these many ROs can have a negative effect on the overall functionality of the platform. The high frequency oscillation of the ROs increases the temperature and has

a straining effect on the fabric's logic components, degrading the life expectancy of the board. Furthermore, the size of the ZedBoard's FPGA fabric discourages the implementation of such a big grid of ROs. As we have extracted enough information to support the conclusions of this thesis, there is no reason to test the last two configurations.

By analyzing the CPA results plot (Figure 6.1.vi) , we notice that as the ROs increase, the iterations required to extract the first byte not only increase, but they increase at a higher rate. A major contributor to this is that as the number of ROs per bank is increased, each activation level injects more noise to the system. In addition to that, the fence itself increases the total power consumption, affecting the oscillator sensor. This results in a greater range of the sensor's values, thus, in more levels of activation and a higher quality of noise injection in total.

The complete results of the ZedBoard platform are shown on Table 1.

Countermeasure	AES Iterations	Increase Over the Implementation Without Active Fence	TDC Data	Plaintext/Ciphertext Data
None	3500		6.68 MB	54.69 KB
1024 ROs Active Fence	12500	3.57x	23.84 MB	195.31 KB
2048 ROs Active Fence	31000	8.86x	59.18 MB	484.37 KB
3072 ROs Active Fence	54000	13.43x	102.99 MB	843.75 KB
4096 ROs Active Fence	94000	26.86x	179.29 MB	1.43 MB
5120 ROs Active Fence	-	-	-	-
6144 ROs Active Fence	-	-	-	-

Table 6.1-1: ZedBoard CPA Results

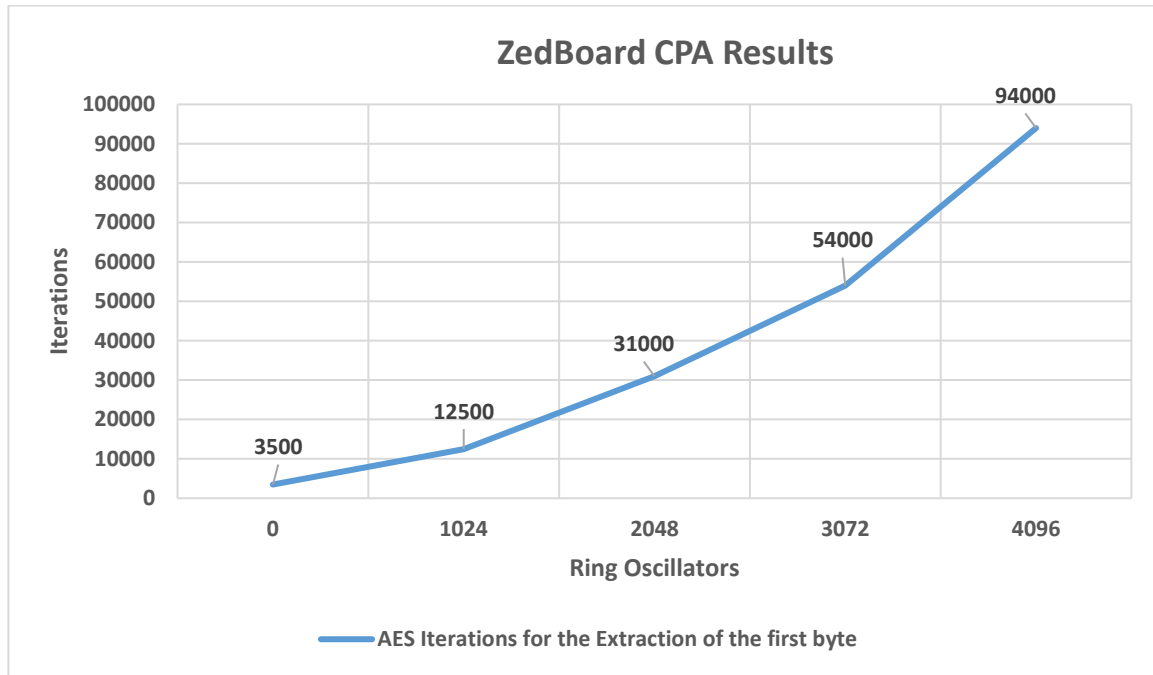


Figure 6.1.vi : ZedBoard CPA Results Plot

6.2 ZCU104 Platform Results

We follow the same process for the ZCU104 platform. We first confirm that our setup is fully functional by conducting an attack without any countermeasures present.

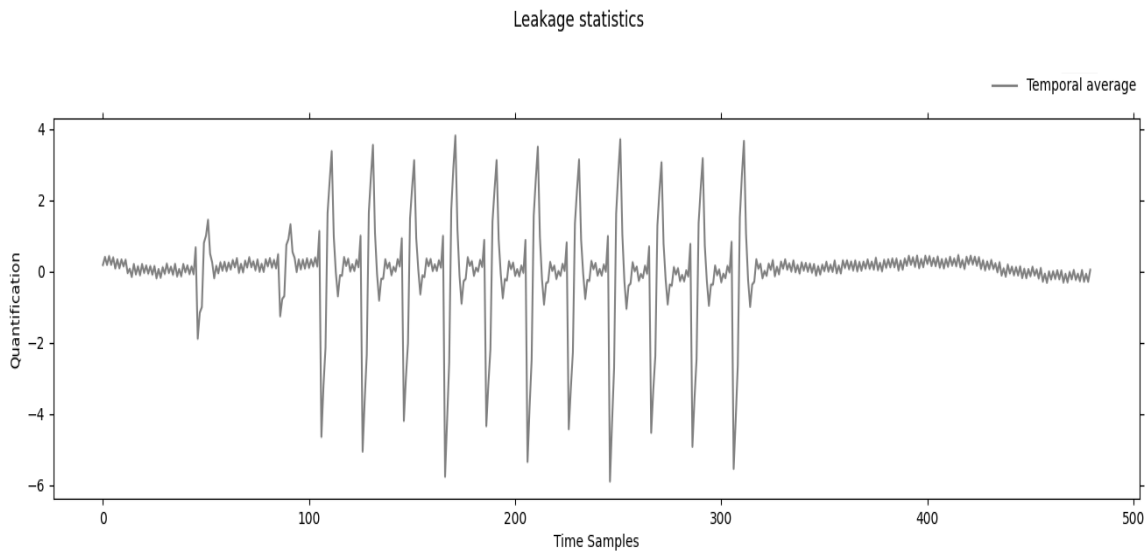


Figure 6.2.i: Quantification vs. Time Samples. Average of 500 AES Iterations without countermeasure (ZCU104)

As in Section 6.1, the Quantification Vs. Time Samples plot of the design on the ZCU104 (Figure 6.2.i) shows that the behaviour of the AES hardware module is capturable by the sensor as there are 11 distinct spikes with similar behaviour between them.

By comparing the plots we see that there are differences between the two platforms. First, the magnitude of the ZCU104 platform plot, is one order of magnitude lower than that of the ZedBoard, which means that the functionality of the AES hardware module has a smaller effect on the PDN of the ZCU104 platform. This has been anticipated as the ZCU104's FPGA fabric is of newer architecture and larger scale than the one on the ZedBoard. The presence of the PS of the ZCU104 also has a hiding effect on the overall system. It also introduces the high frequency noise that is apparent in the plot.

Finally, we also see that the differences of the 11 AES cycles is less distinctable. This also proves the above conclusion that the partial functionality of the AES has a smaller effect on the PDN while running on the US+ fabric of the ZCU104.

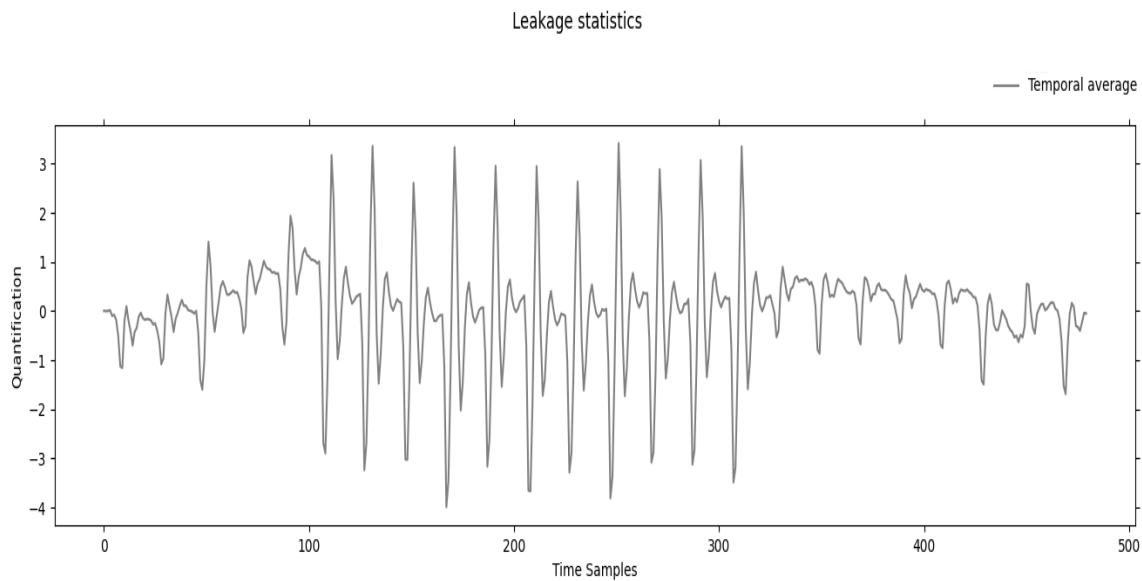


Figure 6.2.ii: Quantification vs. Time Samples. Average of 500 AES Iterations with Active Fence (ZCU104)

We test the different Active Fence configurations on the ZCU104 platform as in [Section 6.1](#). The observations and conclusions that are derived by the Pearson's Correlation plots of the attack on the ZCU104 plots are the same as the ones on the ZedBoard plots.

By comparing the Quantification vs. Time Samples plots between the design with and without the Active Fence countermeasure (Figure 6.2.i vs. Figure 6.2.ii) we see that the values of the second plot, are of lower magnitude. We can also observe sign wave behaviour on the plot of the design with the Active Fence present. This shows that the countermeasure, as in the ZedBoard platform, injects noise to the design and achieves hiding of the AES functionality, altering the measurements.

After the mapping of each Active Fence configurations we reach the following observations and conclusions. As the number of the ROs increases for each configuration,

we observe a significant increase in the AES iterations that need to be collected to extract the first true byte value through the Power Analysis.

Furthermore, the correct byte values have a lower correlation, and their correlation has a smaller divergence between the other values' correlation, as the number of ROs increases, making them less probable to be true. This can be observed by comparing the plots between Figure 6.2.iii and Figure 6.2.iv.

The power analysis of the ZCU104 implementation without any countermeasures (Figure 6.2.iii) achieves extraction of the second byte (1 of 15) after the collection of 60000 AES iterations. This corresponds to 114.44MB of collected TDC measurements, 937.5KB of plaintexts and 937.5KB of ciphertexts.

The power analysis of the ZCU104 implementation with a 5120 Active Fence (Figure 6.2.iv) achieves extraction of the second byte (1 of 15) after the collection of 760000 AES iterations. This corresponds to 1.42GB of collected TDC measurements, 11.6MB of plaintexts and 11.6MB of ciphertexts.

At the configuration of the 6144 RO Active Fence we reach a cut off point. We were not able to achieve the extraction of any key byte, even with 10^6 collected AES iterations, which correspond to 1.86GB of collected TDC measurements, 15.25MB of plaintexts and 15.25MB of ciphertexts. As explained in [Section 6.1](#), using an Active Fence with this amount of ROs for a long period of time can be straining for the board and may cause even the corruption of the fabric's logic components. Thus, we didn't push the tests of the 6144 to more than 10^6 AES iterations.

By analyzing the CPA results plot (Figure 6.2.v) , we notice that we have the same phenomenon in both platfroms. As the ROs increase, the iterations required to extract the first byte not only increase, but they increase at a higher rate, meaning that the increase of the ROs in the Active Fence results in a greater range of the sensor's values, thus, in more levels of activation and a higher quality of noise injection in total.

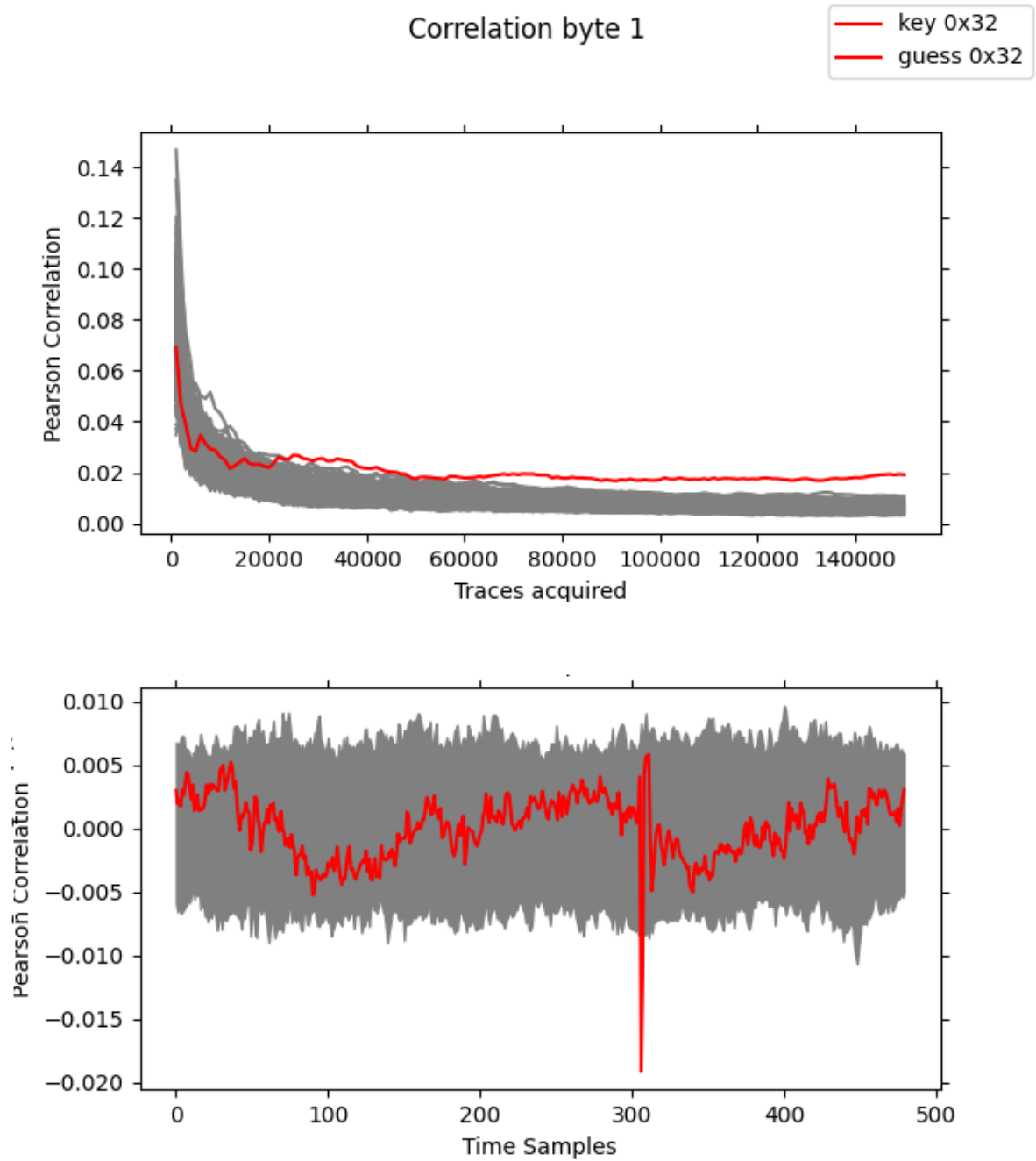


Figure 6.2.iii: Pearson's Correlation Plots for the Second Key Byte without Active Fence (ZCU104)

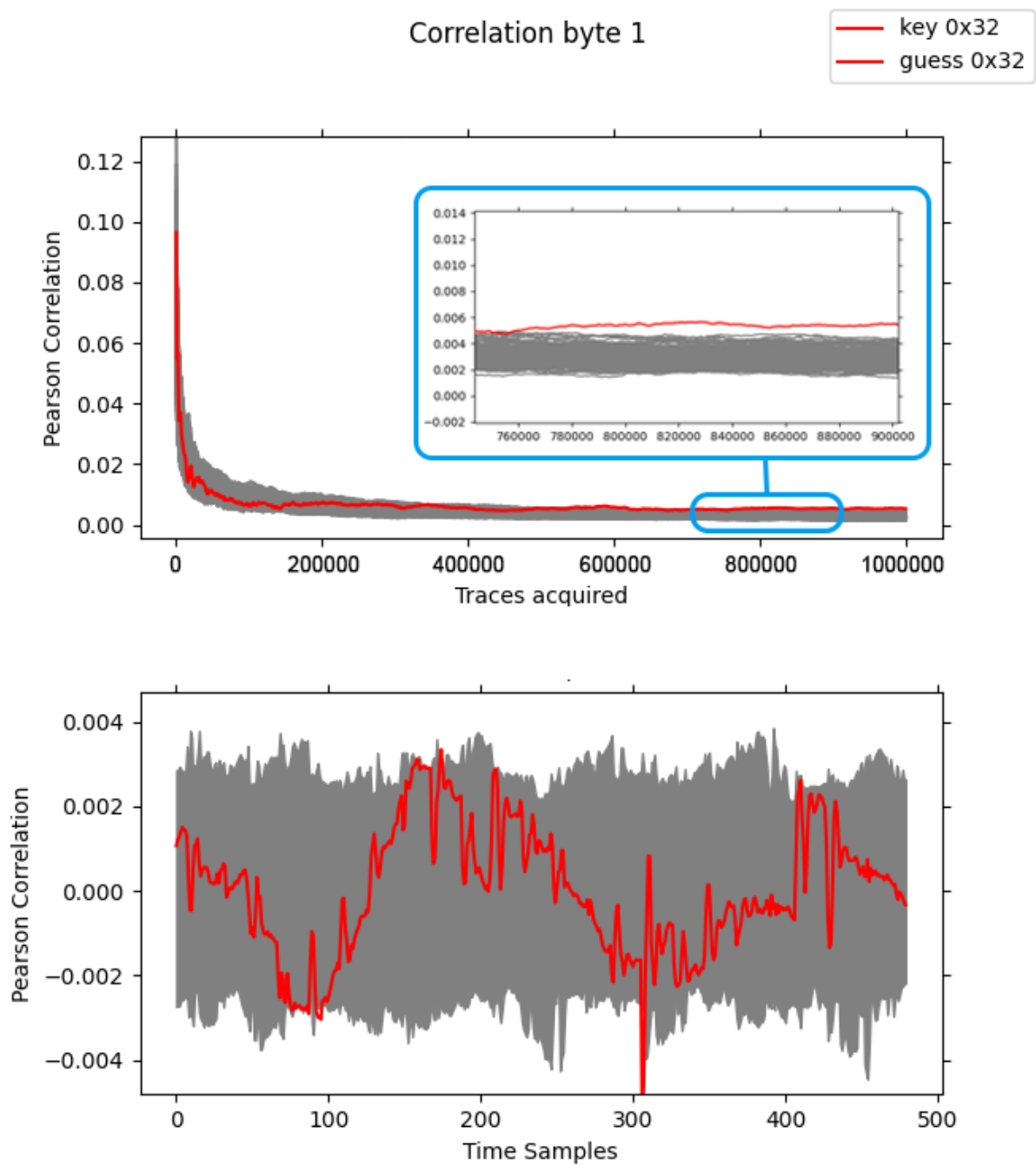


Figure 6.2.iv: Pearson's Correlation Plots for the Fourth Key Byte with 5120 RO Active Fence (ZCU104)

Countermeasure	AES Iterations	Increase Over the Implementation	TDC data	Plaintext/Ciphertext Data
None	65000	Without Active Fence	123.98 MB	0.99 MB
1024 ROs Active Fence	100000		23.84 MB	1.53 MB
2048 ROs Active Fence	170000	1.54x	23.84 MB	1.53 MB
3072 ROs Active Fence	220000	2.62x	190.73 MB	2.59 MB
4096 ROs Active Fence	480000	3.38x	324.25.99 MB	3.36 MB
5120 ROs Active Fence	770000	7.38x	707.45 MB	7.32 MB
6144 ROs Active Fence	> 10 ⁶	11.85x	1.1 GB	11.75 MB
6144 ROs Active Fence	> 10 ⁶	> 15.38x	> 1.44 GB	15.26 MB

Table 6.2-1 : ZCU104 CPA Results

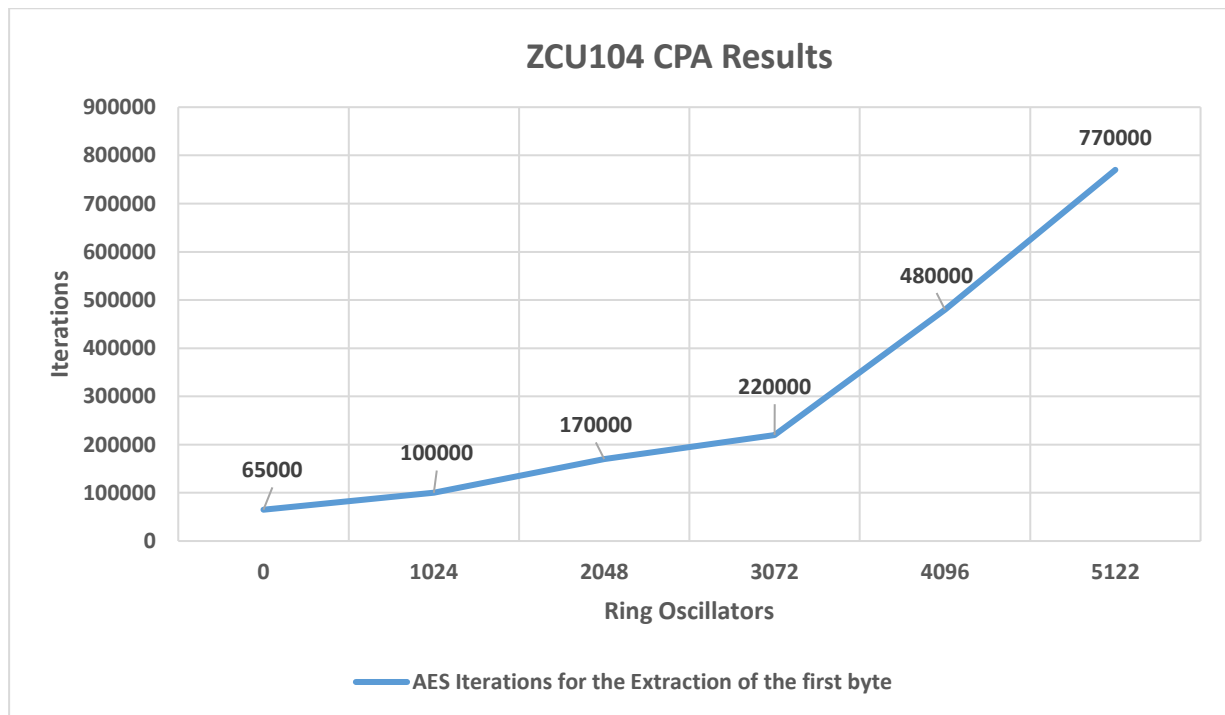


Figure 6.2.v: ZCU104 CPA Results Plot

Comparison of the two platforms

Design	AES Iterations	Design	AES Iterations	Percentage Over ZedBoard Implementations
ZedBoard no countermeasure	3500	ZCU104 no countermeasure	65000	1757.14% (18.57x)
ZedBoard 1024 ROs Active Fence	12500	ZCU104 1024 ROs Active Fence	100000	700% (8x)
ZedBoard 2048 ROs Active Fence	31000	ZCU104 2048 ROs Active Fence	170000	448.39% (5.48x)
ZedBoard 3072 ROs Active Fence	54000	ZCU104 3072 ROs Active Fence	220000	307.41% (4.07x)
ZedBoard 4096 ROs Active Fence	94000	ZCU104 4096 ROs Active Fence	480000	304.26% (4.04x)
ZedBoard 5120 ROs Active Fence	-	ZCU104 5120 ROs Active Fence	770000	-
ZedBoard 6144 ROs Active Fence	-	ZCU104 6144 ROs Active Fence	$> 10^6$	-

Table 6.2-2 ZedBoard Vs ZCU104 implementations

In both platforms, the Active Fence configurations help to achieve hiding by noise injection, without mitigating the attack completely. In comparison of the results, the countermeasure seems to have a better result on the ZedBoard platform as it makes the attack 26.86x more difficult to mount with a 4096 RO Active Fence, while the on the ZCU104 the 5120 RO Active Fence achieves a 11.85x. Furthermore, increasing the ROs of the Active Fence on the ZedBoard, seems to increase the AES iterations that need to be collected for the extraction of at least one byte, at a higher rate than the Active Fence implemented on the ZCU104.

Although the Active Fence seems to have a higher quality effect on the ZedBoard, there is a big difference in the absolute number of AES iterations for each case, between the two platforms. As shown in Table 6.3-1, to achieve the extraction of one of the key bytes without the presence of any countermeasures on the ZedBoard, we need to capture only 3500 AES iterations. The same design mapped in the ZCU104 platform achieves the extraction of the first byte at 65000 AES iterations, meaning that the attack on the later platform is at least 18.57x harder to mount from the start. This gap closes as the ROs increase in every Active Fence configuration, but the absolute numbers still have a big divergence, meaning that the Active Fence countermeasure may achieve better hiding and higher quality noise injection in the ZedBoard platform but it is much harder to mount a successful attack on the ZCU104 platform with the Active Fence present.

This happens because as the AES hardware module has a smaller effect on the ZCU104 platform's PDN, so does the Active Fence countermeasure. It does not affect the power consumption of the overall design and the sensor's readings with the same rate as it does on the ZedBoard designs. Thus, the same Active Fence configuration achieves different level of hiding and noise injection on each platform.

6.3 Comparison with Similar Works

We compare this work with two other works. First, we compare it to the work of Krautter et. al. [12] as it is the first and only work so far to implement Active Fences against remote power SCAs. This work was conducted on the open-source FPGA development board Radiona ULX3S [30] in a version that integrates a Lattice ECP5 12F FPGA with 12K LUT elements with various other components such as an Espressif ESP32 IoT microcontroller module.

The second work that we compare it with is by Glamocanin et. al. [14]. It is the most recent work that evaluates a platform used by cloud providers, similar to the ZCU104, against remote power side-channel attacks, without the presence of any active countermeasure. This work was conducted on an Amazon EC2 F1 instance [1].

Design	AES Iterations	Active Fence Resource Utilization Percentage of AES	Design	AES Iterations	Percentage Over ZedBoard Implementations
ZedBoard no countermeasure	3500		Glamocanin et. al.	200000	5614.29% (57.14x)
ZedBoard 1024 ROs Active Fence	12500	17%			1500% (16x)
ZedBoard 2048 ROs Active Fence	31000	33%			545.16% (6.45x)
ZedBoard 3072 ROs Active Fence	54000	50%			270.37% (3.7x)
ZedBoard 4096 ROs Active Fence	94000	67%			112.77% (2.13x)
ZedBoard 5120 ROs Active Fence	-	-			-
ZedBoard 6144 ROs Active Fence	-	-			-

Table 6.3-1: ZedBoard Implementations Vs. Glamocanin et. al. Work

Design	AES Iterations	Active Fence Resource Utilization Percentage of AES	Design	AES Iterations	Percentage Over ZCU104 Implementations
ZCU104 no countermeasure	65000		Glamocanin et. al.	200000	207.69% (3.08x)
ZCU104 1024 ROs Active Fence	100000	17%			100% (2x)
ZCU104 2048 ROs Active Fence	170000	33%			17.65% (1.18x)
ZCU104 3072 ROs Active Fence	220000	50%			-9.09% (0.91x)
ZCU104 4096 ROs Active Fence	480000	67%			-58.33% (0.42x)
ZCU104 5120 ROs Active Fence	770000	83%			-74.03% (0.26x)
ZCU104 6144 ROs Active Fence	$> 10^6$	100%			> -80% (0.2x)

Table 6.3-2: ZCU104 Implementations Vs. Glamocanin et. al. Work

Design	AES Iterations	Active Fence Resource Utilization Percentage of AES	Design	AES Iterations	Active Fence Resource Utilization Percentage of AES	Percentage Over ZedBoard Implementations
ZedBoard no countermeasure	3500		Krautter et. al. no countermeasure	1800		-48.57% (0.51x)
ZedBoard 1024 ROs Active Fence	12500	17%	Krautter et. al. TDC Active Fence	300000	100%	2300% (24x)
ZedBoard 2048 ROs Active Fence	31000	33%				867.74% (9.67x)
ZedBoard 3072 ROs Active Fence	54000	50%				455.56% (5.56x)
ZedBoard 4096 ROs Active Fence	94000	67%				219.15 % (3.19x)
ZedBoard 5120 ROs Active Fence	-	-				-
ZedBoard 6144 ROs Active Fence	-	-				-

Table 6.3-3: ZedBoard Implementations Vs. Krautter et. al. Work

Design	AES Iterations	Active Fence Resource Utilization Percentage of AES	Design	AES Iterations	Active Fence Resource Utilization Percentage of AES	Percentage Over ZCU104 Implementations
ZCU104 no counter-measure	65000		Krautter et. al. no counter-measure	1800		-97.23% (0.03x)
ZCU104 1024 ROs Active Fence	100000	17%	Krautter et. al. TDC Active Fence	300000	100%	200% (3x)
ZCU104 2048 ROs Active Fence	170000	33%				76.47% (1.76x)
ZCU104 3072 ROs Active Fence	220000	50%				36.36% (1.36x)
ZCU104 4096 ROs Active Fence	480000	67%				-37.5% (0.62x)
ZCU104 5120 ROs Active Fence	770000	83%				-61.04% (0.39x)
ZCU104 6144 ROs Active Fence	$> 10^6$	100%				> -70% (0.3x)

Table 6.3-4: ZCU104 Implementations Vs. Krautter et. al. Work

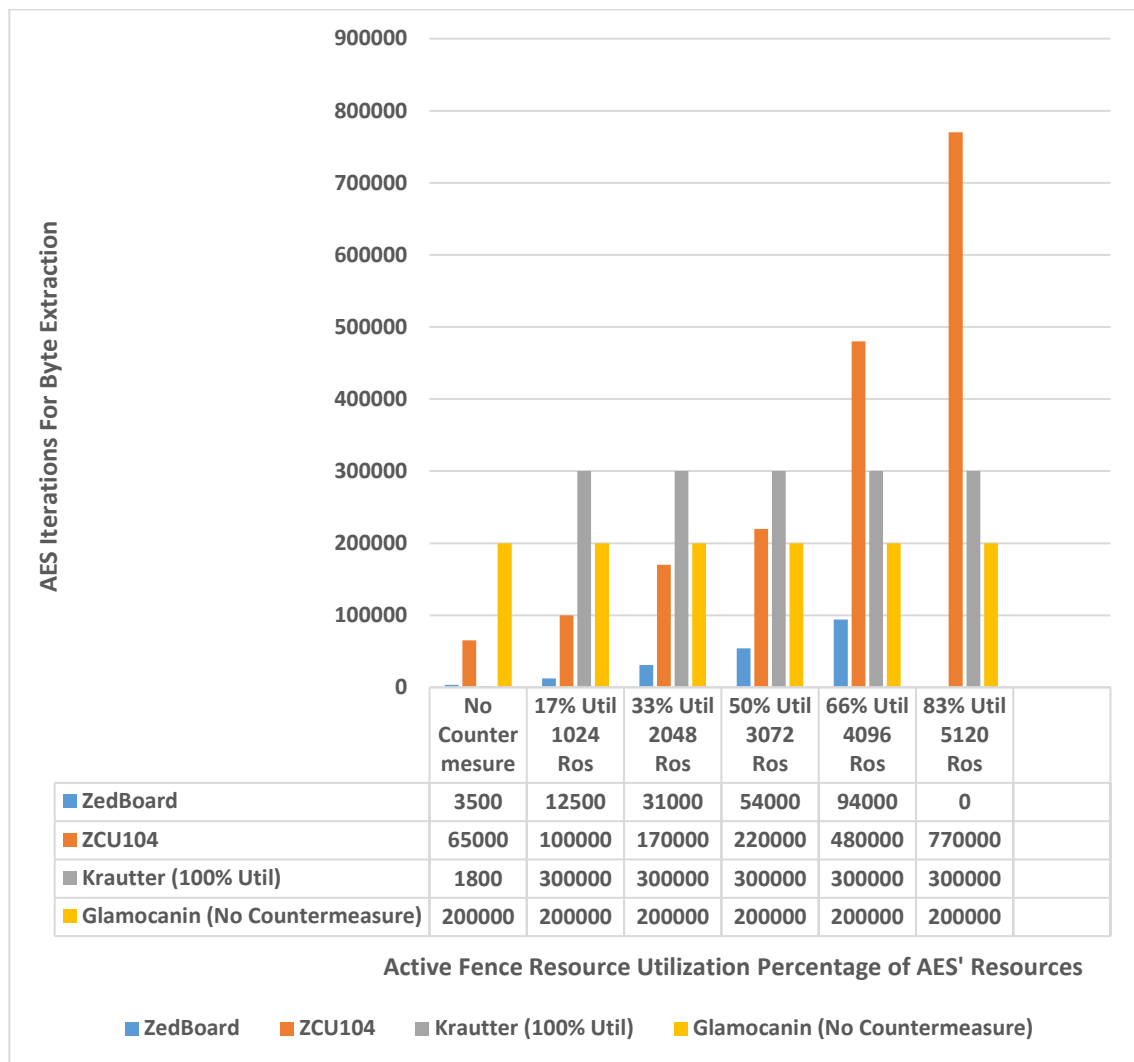


Figure 6.3.i Comparisson of Each Platform

Glamocanin et. al. [14], does not report the exact number of captured AES iterations for extracting the first correct value of the key, but report that the third byte of the key is extracted after around 200000 AES traces have been collected. Thus, we compare it with our experimental results based on this information as it is the only work so far to mount a remote SCA on an US+ FPGA fabric.

6.4 Results' Evaluation Summarization

The results and evaluation of this work can be summarized as follows:

- The required iterations to extract any byte of the key, increase along with the number of ROs.
- The correct value of each byte has a lower correlation compared to the other 255 as the number of ROs increases.
- The Active Fence countermeasure achieves hiding and injects noise to the system, without utilizing as many resources as the module under attack.
- The Active Fence countermeasure has a different effect, depending on the platform. On the ZedBoard platform an Active Fence utilizing only 67% of the AES' resources, makes the attack up to 26.86x more difficult to mount. On the ZCU104 platform, the same Active Fence makes it 7.38x.
- An RO sensor can effectively control the Active Fence. It is less effective than using a TDC power sensor but uses significantly less resources as can be observed in Table 4.2-1 and Table 4.2-2. The TDC sensor enabled Active Fence of Krautter et. al. [12] utilizes 100% of the AES core resources and achieves more than 166x leakage reduction while the RO sensor enabled Active Fence on the ZedBoard, utilizing 67% of the AES' resources, makes the attack up to 26.86x more difficult to mount.
- Larger platforms, of newer technology and architecture, are less vulnerable to SCAs. By adding the Active Fence and other countermeasures, an SCA can become almost unmountable on such platforms. To mount successfully extract even one key byte on the ZCU104 and EC2 F1 platforms without countermeasures, at least 65000 AES iterations need to be collected compared to the 1800 and 3500 AES iterations that are needed on the Radiona ULX3S and ZedBoard, respectively. The Active Fence on a ZCU104 achieves only 15.38x leakage reduction at maximum resource utilization but the absolute number of the AES iterations is more than 10^6 , while the TDC sensor enabled Active Fence of Krautter et. al. achieves 166x leakage reduction with 300000 AES iterations, 70% less iterations than the ZCU104 implementation.

Chapter 7: Conclusions and Future Work

FPGA devices have a multifaceted range of applications and are subsequently moving to cloud instances as more and more applications are deployed there. In these environments, cost efficiencies are paramount and the highest degrees of utilisation are sought by providers. In that context, sharing the FPGA resources between different users is the next logical step, thus creating multi-tenant use-case scenarios. Security issues though that plague those sharing efforts prevent a wide deployment of the concept and measures to prevent side-channel attacks are sought.

This is the focus of this work. More specifically, we evaluate different Active Fence configurations on two different platforms, with one of them using same transistor technology (US+) as the one commonly used in the cloud service providers' platforms. This defence mechanism has the potential of ensuring low exposure to power side-channel attacks on unsuspecting users trying out their designs on the cloud FPGAs. For demonstration purposes, we use a specific type of setup comprised of a TDC as the attacker's sensor and an AES module as the victim's design. In that, context, different Active Fences of different sizes have been tried out and insightful observations have been made.

First, we quantify a relationship between Active Fence size and size of data traces required for a successful attack. Our work shows that with the correct choice of Active Fence configuration and platform, the amount of data traces required for key extraction can be many times greater compared to no countermeasures present. This leads to prohibitively large amount of data and time for the attack, while using an Active Fence that requires less resources than the module under attack. Additionally, even though the two most efficient Active Fence configurations occupy almost as many resources as the algorithm, not all of the fence's ROs are used, and that is due to the activation method. This shows that we can implement Active Fences with smaller area overhead, that can increase the noise to a sufficient level and make the attacker almost unable to succeed in extracting the key of the cryptographic module.

Future work may assess and point out the most efficient combination of Active Fence configuration, activation technique and platform, in terms of performance. Furthermore, the present emulation has accounted for a two-tenant, Intra-FPGA, scenario with some compromises. We do not take into account possible masking techniques that may be used for each algorithm or the possibility of more users on the platform. Our work may be

extended, using algorithmic level countermeasures and/or with more than two tenants, malicious or not.

Finally, although different platforms have been evaluated against remote power side-channel attacks, power level countermeasures that are algorithm independent have not been tested widely. A wide evaluation of different platforms and countermeasures, could help the service providers chose the best combination of the two, providing the most secure solution.

Chapter 8: References

- [1] “Amazon EC2 F1 Instances.” <https://aws.amazon.com/ec2/instance-types/f1/> (accessed Dec. 09, 2021).
- [2] K. Eguro and R. Venkatesan, “FPGAs for trusted cloud computing,” in *22nd International Conference on Field Programmable Logic and Applications (FPL)*, Aug. 2012, pp. 63–70. doi: 10.1109/FPL.2012.6339242.
- [3] S. Byma, J. G. Steffan, H. Bannazadeh, A. Leon-Garcia, and P. Chow, “FPGAs in the Cloud: Booting Virtualized Hardware Accelerators with OpenStack,” in *2014 IEEE 22nd Annual International Symposium on Field-Programmable Custom Computing Machines*, May 2014, pp. 109–116. doi: 10.1109/FCCM.2014.42.
- [4] S. A. Fahmy, K. Vipin, and S. Shreejith, “Virtualized FPGA Accelerators for Efficient Cloud Computing,” in *2015 IEEE 7th International Conference on Cloud Computing Technology and Science (CloudCom)*, Nov. 2015, pp. 430–435. doi: 10.1109/CloudCom.2015.60.
- [5] S. Trimberger and S. McNeil, “Security of FPGAs in data centers,” in *2017 IEEE 2nd International Verification and Security Workshop (IVSW)*, Jul. 2017, pp. 117–122. doi: 10.1109/IVSW.2017.8031556.
- [6] J. Krautter, D. R. E. Gnad, and M. B. Tahoori, “FPGAhammer: Remote Voltage Fault Attacks on Shared FPGAs, suitable for DFA on AES,” *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, pp. 44–68, Aug. 2018, doi: 10.13154/tches.v2018.i3.44-68.
- [7] F. Schellenberg, D. Gnad, A. Moradi, and M. Tahoori, “Remote inter-chip power analysis side-channel attacks at board-level,” Nov. 2018, pp. 1–7. doi: 10.1145/3240765.3240841.
- [8] M. Zhao and G. E. Suh, “FPGA-Based Remote Power Side-Channel Attacks,” in *2018 IEEE Symposium on Security and Privacy (SP)*, May 2018, pp. 229–244. doi: 10.1109/SP.2018.00049.
- [9] D. Corbett, M. Automation, M. A. Stake, M. Automation, and M. A. Stake, “The Xilinx Isolation Design Flow for Fault-Tolerant Systems.” 2013.
- [10] J. Dj. Golic, “Techniques for Random Masking in Hardware,” *IEEE Trans. Circuits Syst. Regul. Pap.*, vol. 54, no. 2, pp. 291–300, Feb. 2007, doi: 10.1109/TCSI.2006.885974.
- [11] T. Güneysu and A. Moradi, “Generic Side-Channel Countermeasures for Reconfigurable Devices,” in *Cryptographic Hardware and Embedded Systems – CHES 2011*, Berlin, Heidelberg, 2011, pp. 33–48. doi: 10.1007/978-3-642-23951-9_3.
- [12] J. Krautter, D. R. E. Gnad, F. Schellenberg, A. Moradi, and M. B. Tahoori, “Active Fences against Voltage-based Side Channels in Multi-Tenant FPGAs,” in *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, Nov. 2019, pp. 1–8. doi: 10.1109/ICCAD45719.2019.8942094.
- [13] J. Gravelier, J.-M. Dutertre, Y. Teglia, and P. Loubet-Moundi, “High-Speed Ring Oscillator based Sensors for Remote Side-Channel Attacks on FPGAs,” in *2019 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, Dec. 2019, pp. 1–8. doi: 10.1109/ReConFig48160.2019.8994789.
- [14] O. Glamocanin, L. Coulon, F. Regazzoni, and M. Stojilović, “Are Cloud FPGAs Really Vulnerable to Power Analysis Attacks?,” in *2020 Design, Automation Test in Europe Conference Exhibition (DATE)*, Mar. 2020, pp. 1007–1010. doi: 10.23919/DATE48585.2020.9116481.
- [15] T. Huffmire et al., “Moats and Drawbridges: An Isolation Primitive for Reconfigurable Hardware Based Systems,” *2007 IEEE Symp. Secur. Priv. SP 07*, 2007, doi: 10.1109/SP.2007.28.

- [16] M. S. Gupta, J. L. Oatley, R. Joseph, G.-Y. Wei, and D. M. Brooks, "Understanding Voltage Variations in Chip Multiprocessors using a Distributed Power-Delivery Network," in *Automation Test in Europe Conference Exhibition 2007 Design*, Apr. 2007, pp. 1–6. doi: 10.1109/DATE.2007.364663.
- [17] R. Bertran *et al.*, "Voltage Noise in Multi-Core Processors: Empirical Characterization and Optimization Opportunities," in *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*, Dec. 2014, pp. 368–380. doi: 10.1109/MICRO.2014.12.
- [18] D. R. E. Gnad, F. Oboril, S. Kiamehr, and M. B. Tahoori, "Analysis of transient voltage fluctuations in FPGAs," in *2016 International Conference on Field-Programmable Technology (FPT)*, Dec. 2016, pp. 12–19. doi: 10.1109/FPT.2016.7929182.
- [19] K. M. Zick, M. Srivastav, W. Zhang, and M. French, "Sensing nanosecond-scale voltage attacks and natural transients in FPGAs," in *Proceedings of the ACM/SIGDA international symposium on Field programmable gate arrays*, New York, NY, USA, Feb. 2013, pp. 101–104. doi: 10.1145/2435264.2435283.
- [20] F. Schellenberg, D. R. E. Gnad, A. Moradi, and M. B. Tahoori, "An inside job: Remote power analysis attacks on FPGAs," in *2018 Design, Automation Test in Europe Conference Exhibition (DATE)*, Mar. 2018, pp. 1111–1116. doi: 10.23919/DATE.2018.8342177.
- [21] D. R. E. Gnad, S. Rapp, J. Krautter, and M. B. Tahoori, "Checking for Electrical Level Security Threats in Bitstreams for Multi-tenant FPGAs," in *2018 International Conference on Field-Programmable Technology (FPT)*, Dec. 2018, pp. 286–289. doi: 10.1109/FPT.2018.00055.
- [22] M. McLean and J. Moore, "FPGA-BASED SINGLE CHIP CRYPTOGRAPHIC SOLUTION (U)," 2007. [https://www.semanticscholar.org/paper/FPGA-BASED-SINGLE-CHIP-CRYPTOGRAPHIC-SOLUTION-\(-U-\)-McLean-Moore/39372338344c0210576b9869fcc66c2a3935f044](https://www.semanticscholar.org/paper/FPGA-BASED-SINGLE-CHIP-CRYPTOGRAPHIC-SOLUTION-(-U-)-McLean-Moore/39372338344c0210576b9869fcc66c2a3935f044) (accessed Dec. 09, 2021).
- [23] Z. Yuan, Y. Wang, J. Li, R. Li, and W. Zhao, "FPGA based optimization for masked AES implementation," in *2011 IEEE 54th International Midwest Symposium on Circuits and Systems (MWSCAS)*, Aug. 2011, pp. 1–4. doi: 10.1109/MWSCAS.2011.6026388.
- [24] J. Waddle and D. Wagner, "Towards Efficient Second-Order Power Analysis," 2004. doi: 10.1007/978-3-540-28632-5_1.
- [25] L. Ordu and B. Ors, "Power Analysis Resistant Hardware Implementations of AES," in *2007 14th IEEE International Conference on Electronics, Circuits and Systems*, Dec. 2007, pp. 1408–1411. doi: 10.1109/ICECS.2007.4511263.
- [26] A. Wild, A. Moradi, and T. Güneysu, "GliFreD: Glitch-Free Duplication Towards Power-Equalized Circuits on FPGAs," *IEEE Trans. Comput.*, vol. 67, no. 3, pp. 375–387, Mar. 2018, doi: 10.1109/TC.2017.2651829.
- [27] "ZedBoard | Avnet Boards." <https://www.avnet.com/wps/portal/us/products/avnet-boards/avnet-board-families/zedboard/>
- [28] "Zynq UltraScale+ MPSoC ZCU104 Evaluation Kit." <https://www.xilinx.com/products/boards-and-kits/zcu104.html>
- [29] "SCAbox 1.1.0 documentation." <https://emse-sas-lab.github.io/SCAbox/tuto/home.html#tutorials>
- [30] "Radiona ULX3S." <https://radiona.org/ulx3s/>