5-2022

Enhancing System Security Using Dynamic Hardware

Sydney L. Davis
University of South Alabama, sld1424@jagmail.southalabama.edu

ENHANCING SYSTEM SECURITY USING DYNAMIC HARDWARE

A Thesis

Submitted to the Graduate Faculty of the
University of South Alabama
in partial fulfillment of the
requirements for the degree of

Master of Science

in

Computer and Information Sciences

by
Sydney L. Davis
B.S., University of South Alabama, 2020
May 2022

ACKNOWLEDGEMENTS

I express my deepest gratitude to my research mentor and supervisor, Dr. Todd Andel, for his close guidance, valuable critiques, and continued support throughout the research process. I thank Dr. Todd McDonald and Dr. Sam Russ for sharing their invaluable knowledge and giving me so much of their time in the development of this research. To the National Science Foundation, I give my deepest appreciation for funding my research under grant D0GE-1564518. And finally, I give my special thanks to my family for supporting me from beginning to end. Thank you.

TABLE OF CONTENTS

Page

LIST OF TABLES

Table Page

LIST OF FIGURES

Figure Page

LIST OF ABBREVIATIONS

AES	Advanced Encryption Standard
ARM	Advanced RISC Machines
ASIC	Application Specific Integrated Circuit
AXI	Advanced Extensible Interface
CBC	Cipher Block Chaining
CFB	Cipher Feedback
CLB	Configurable Logic Block
CMOS	Complementary Metal Oxide Semiconductor
DES	Data Encryption Standard
DPA	Differential Power Analysis
ECB	Electronic Codebook
EMA	Electromagnetic Analysis
FF	Flip Flop
FPGA	Field Programmable Gate Array
f_s	Sample Frequency
GPIO	General Purpose Input/Output
GPP	General Purpose Processor
I/O	Input/Output
IP	Initial Permutation
LUT	Look Up Table
MTD	Moving Target Defense
OFB	Output Feedback
PCC	Power Consumption Curves
PDR	Partial Dynamic Reconfiguration

PET	Program Encryption Toolkit
PL	Programmable Logic
PR	Partial Reconfiguration
SCA	Side Channel Analysis
SDK	Software Development Kit
SoC	System on a Chip
SUT	System Under Test
t_e	Execution Time
UART	Universal Asynchronous Receiver-Transmitter
VHDL	VHSIC Hardware Description Language
XOR	Exclusive OR

ABSTRACT

Davis, Sydney, L., M. S., University of South Alabama, May 2022. Enhancing System Security Using Dynamic Hardware. Chair of Committee: Todd, Andel, Ph.D.

Within the ever-advancing field of computing, there is significant research into the many facets of cyber security. However, there is very little research to support the concept of using a Field Programmable Gate Array (FPGA) to increase the security of a system. While its most common use is to provide efficiency and speedup of processes, this research considers the use of an FPGA to mitigate vulnerabilities in both software and hardware. This paper proposes circuit variance within an FPGA as a method of Moving Target Defense (MTD) and investigates its effect on side-channels. We hypothesize that although the functionality of native and variant circuits is the same, their subsequent side-channel characterizations will differ thus creating unique electromagnetic signatures. The investigation and observations of the study include circuit variant construction, side channel attacks and analyses, and subsequent comparisons of electromagnetic signatures. We found that in the analysis of variant DES implementations, there are small but present differences in side channel depictions from native to variant.

CHAPTER I

INTRODUCTION

In an increasingly technologically advancing world, ensuring the security of systems and their integrity against malicious attacks has become a major research focus in the computer security community. Although computer security and its many facets are advancing at an exponential rate, hackers are unfortunately progressing at a similar rate. Just recently, an Equifax data breach orchestrated by a community of hackers cost 143 million Americans the sensitivity of their confidential personal information [1]. Misfortunes such as these generate an even bigger outcry for more secure systems. A very widely used method of securing these systems is moving target defense otherwise known as MTD. This technique creates a more difficult environment for an attacker to implement a successful intrusion thus defending against vulnerabilities within the system [2], [3], [4], [5]. Although there are various types of MTDs, a specific application is the use of circuit variance, which at its most basic level is the act of generating equivalent circuits to produce variants that are constantly changing over a previously specified period. Using these variants creates the illusion that the entire system is constantly changing which assists in making it more challenging for an attacker to reverse engineer the system.

In this research, we investigate the utilization of programmable hardware to identify and mitigate vulnerabilities in software and/or hardware. We theorize that implementing circuit variance as a method of MTD will provide added security to a given system because side channel characterizations of native and variant will be different. In exploration of this theory, we chose an FPGA as the programmable hardware and decided on a simple implementation of DES encryption as the native circuit. The goal of the research is to determine if side channel characterizations, whether power consumption or electromagnetic signatures, differ between native and variant although they are functionally equivalent.

1.1 Research Question and Goals

Often defense mechanisms for system intrusions by attackers are fixed. The defenses stay the same and do not vary therefore allowing attackers ample time to determine the inner workings of a circuit with the purpose of reverse engineering it. This paper proposes circuit variance as a method of moving target defense (MTD) to counter attacks and increase the overall level of security for the system. We theorize that while the functionality of native and variant circuits is the same, their subsequent side-channel characterizations will differ thus creating unique electromagnetic signatures.

The goal of this research is to determine if employing circuit variance as a method of moving target defense (MTD) will provide varying levels of security. The fundamental question underlying the research is "*How does circuit variant implementation relate to side channel characterization?"* Therefore, in this approach, the side-channel characteristics of a native DES circuit implementation and its variants will be evaluated

and compared with the purpose of determining if these characteristics can distinguish native from variant.

1.2 Thesis Outline

The remainder of this document is outlined as follows: Chapter II presents an overview of background concepts and related works. Chapter III describes in detail the methodology and process for the experimental setup and analyzation of data. Chapter IV details our experimental results and discussion. Chapter V summarizes our conclusions and gives a brief proposal for potential future work.

CHAPTER II

BACKGROUND AND RELATED WORKS

This chapter provides background on frequently mentioned concepts and methods in Chapter III as well as an overview of relevant literature in this field. Section 2.1 offers the contextual information necessary to grasp the concepts presented in the remainder of this document including an overview of such topics as moving target defense, circuit variance, properties of FPGAs, and side channel analysis Section 2.2, presents and summarizes relevant related works in this research area including but not limited to dynamic code mutation, and software partitioning.

2.1 Background

In this section, several key words and phrases will be defined in detail. These subsections and their contents will help to give a well-rounded view of the research area and in turn create a level knowledge base for those unfamiliar with the specifics of this field.

2.1.1 Moving Target Defense (MTD)

Moving Target Defense (MTD) is a fairly new concept of security that involves increasing entropy in any given system in order to reduce the attack surface available to hackers or any entity that would do harm to the system [3], [2], [4]. In systems, both

hardware and software, that do not utilize an MTD method an attacker's biggest advantage is often the static nature of the system [2], [5]. Because manufacturers do not always design a system with an attacker's plans in mind, most systems are vulnerable to intrusion based solely on the fact that there is no variance within the system itself. An attacker can take all the time he needs to attempt to reverse engineer the circuit or the program because he knows that he can always return to this same circuit or program. Conversely, employing MTD confuses attackers by creating more entropy or randomness thus allowing a considerably smaller window for successful invasions. Investigating this method using obfuscation and/or circuit variance is the overarching goal of this research.

2.1.2 Obfuscation

Obfuscation is one of several defense mechanisms used to mitigate attacks by hackers that would do harm to a system. It is simply the use of obscurity to thwart intrusion. Designers who implement obfuscation into their code or circuits desire to make the code or circuit as challenging to comprehend as possible while still retaining the identical functionality [6]**,** [7]. The use of obfuscation reduces the chances of an attacker reverse engineering a system in order to gain access to it [2], [5], [8]. By making the system harder to interpret, the attacker must spend more time attempting to figure it out. Using this tactic as a form of moving target defense (MTD) along with the entropy it creates helps to ensure the security of a system. This research investigates circuit obfuscation by introducing redundancies into the variant DES implementation to be analyzed therefore expanding the circuit.

2.1.3 Circuit Variance

Circuit variance is the method of creating obfuscated logic or generating functionally equivalent variants of a circuit with the intention of creating a more difficult environment for an attacker to thrive [7], [6], [8]. These variants are a "family of equivalent circuits that maintain the functional program properties…, but look entirely different from a structural point of view" [9]. Circuit variance is a type of dynamic variation meaning that the variants generated are constantly changing over a period of time. This variation in the structure of the logic would appear to an attacker as several completely different circuits as opposed to variants of the same logic. Because one of the first things an adversary must determine when attempting to reverse engineer a system are the different components of a circuit and their function within that circuit, using this obfuscated logic within the system aids in deterring the adversary from going any further in their plans to tamper with the system.

2.1.4 Field Programmable Gate Arrays and Application-Specific Circuits

Field Programmable Gate Arrays otherwise known as FPGAs are powerful, reconfigurable integrated circuits that have logic structures that can execute the logic of a given circuit or program [2], [10]. They consist of an array of interconnected logic blocks that can be programmed at both the function of the logic blocks, the connections between the blocks, and the inputs and outputs [11]. In this regard, the FPGA is considered reconfigurable hardware. FPGAs combine the advantages of general-purpose processors and specialized circuits because they can be reconfigured during run-time to better meet the needs of a system. These chips are quickly replacing the widely accepted Application-Specific Integrated Circuit (ASIC). An ASIC is a smaller and faster integrated circuit that

unlike an FPGA, has its logical structure hardwired into its design. Although smaller and faster than an FPGA, an ASIC is not reconfigurable like an FPGA. This along with cost and scope of use are causing the ASIC to be replaced with the FPGA more and more [10], [9].

Since its invention, the applications of the FGPA have included speeding up security processes, ensuring the integrity of a program during run-time, and increasing overall program speed [9], [12]. Although several of its applications have to do with enhancing the security of a system by providing speedup of security processing, FPGAs themselves have not been the specifically used to mitigate software or hardware vulnerabilities. This research, however, considers the use of FPGAs to identify and potentially mitigate vulnerabilities within various circuits and their side channel characterizations.

2.1.5 Partial Reconfiguration and Partial Dynamic Reconfiguration (PR & PDR)

"Partial reconfiguration and partial dynamic reconfiguration is the ability of an FPGA to change its functionality without having to be powered down" [2]. This means the FPGA can be reconfigured during execution time. There are two different types of PR: difference-based and module-based. Difference based partial reconfiguration is a process in which there are small changes made to an FPGA configuration by relating to the FPGA what the difference is between the old and new configurations. Module based partial reconfiguration makes uses of several modular design concepts to reconfigure large blocks of logic within the FPGA. These large blocks of logic are then able to replace old configurations without disrupting the rest of the hardware [2], [13]. An advantage of utilizing PR and PDR is increased performance within a system. By using

these techniques, a system can perform without a loss of functionality because of the hardware's ability to efficiently function without powering down every time something needs to be reconfigured. Another advantage of the use of PR and PDR is reduced power consumption and overall size/amount of the hardware used [14]. This functionally is essential in allowing for the concepts proposed in this research into the possibilities of run-time circuit variance and obfuscation for the purpose of system security.

2.1.6 Dynamic Hardware/Software Partitioning

Hardware/Software partitioning can be defined as dividing a program into software on a microprocessor and hardware co-processors. This approach has recently incorporated the use of an FPGA and microprocessor on a single chip, as the hardware, which has made the process more efficient. "By treating the FPGA as an extension of the microprocessor, a designer can move critical software regions from the microprocessor onto FPGA hardware, resulting in improved performance and usually reduced energy consumption…"[15]. Although this is often used as an energy consumption reduction technique, this paper proposes that it be used to better ensure the integrity of a program and its overall security.

2.1.7 Side Channel Analysis (SCA)

Side channel leakage can be easily defined as specific information leakage [16]. Often side channel leakage is in the form of energy or power consumption. SCA attacks are attempts by attackers to use system leakage such as power consumption to aid in reverse engineering that system or to leak cryptographic keying information. "These attacks are conducted by collecting power consumption data of the hardware, referred to as power traces…and statistically correlating this data to the cryptographic key." [9].

Once the attackers obtain the cryptographic key, the system is essentially hacked; and all of this has been done by just analyzing the subsequent side channels of the system. This research proposes that circuit variance, obfuscation, and dynamic reconfiguration will increase the difficulty of side channel analysis, adding to the overall security of a given system.

2.1.8 Differential Power Analysis (DPA)

Differential Power Analysis is an SCA technique that utilizes power side channels to obtain cryptographic keys. Power attacks seek to ascertain information related to operations and manipulated data by examining a device's power consumption [17]. These attacks are carried out "by collecting power consumption data of the hardware, referred to as power traces…and statistically correlating this data to the cryptographic key" [18]. A trace usually shows at least one iteration of encryption, but it is possible to generate a more detailed view of the operation depending on the equipment used to take the measurements and complete the analysis. To take these traces, a current probe must be placed in series with the power source to accurately capture the current. A trigger must also be configured to prompt the probe to capture traces.

Once the probe and trigger are placed and configured, the attacker must decide which part of the algorithm she would like to analyze. For example, if analyzing the Data Encryption Standard, the analysis can either be performed on the first round of DES with the plaintext or on the sixteenth round with the ciphertext. If the analysis is chosen to be performed on the first round, the subkey along with the inputs to each S-box are calculated in turn and their outputs analyzed. Conversely, if the analysis is on the last round, the outputs to each box are calculated and their inputs analyzed.

Once traces are gathered, power consumption curves (PCCs) are collected. These are differential curves derived from box output bits. Because there are sixty-four partial subkey combinations, sixty-four of these traces must be performed. In the analysis, a very noticeable spike should be present in the PCC that was plotted with the correct subkey bits which gives confirmation of key block guesses. Repeating this process yields all 48 subkey bits for the round (where the last 8 bits can be determined by analyzing another round) [17], [19]. Because our goal is not to break the DES cipher and acquire the key, the PCCs will not be gathered and analyzed for subkey bits. Although DPA will not be used directly in this research, power side channels will be collected and analyzed to make comparisons between variant circuits.

2.1.9 Electromagnetic Analysis (EMA)

Electromagnetic analysis is an attack that captures and analyzes electromagnetic emanations from a target device. “Since each active component of the device produces and induces various types of emanations, these emanations provide multiple views of events unfolding within the device at each clock cycle” [20]. Electromagnetic emanations or emissions are the results of current flows through different parts of the device. These emanations can either be intentional or unintentional. The intentional emissions are created from intentional current flows in the device and often correspond to short bursts of current. The unintentional emissions on the other hand are created as a result of proximity. The proximity of the hardware components on a device result in electrical and electromagnetic coupling, and this in turn creates the emanations. The unintentional emanations often provide especially useful and often compromising information about a device and the programs running on it [20]. Because the foundation

of EMA is emanations, it is important to note that in CMOS devices current only flows when logic states are changed; and the changing of states is controlled by a square-wave clock. This in turn means that the resulting emanations carry valuable information about what a program may be doing during each clock cycle.

Analysis of the electromagnetic side channel of a DES cipher begins like DPA with the placement of current probes in addition to near field probes. The near field probe must be placed on the smartcard as close to the microprocessor as possible. The current probe on the other hand is placed in series with the power supply to capture the changing currents through the device. A powerful scope must also be used to capture enough samples of the executing cipher for a meaningful trace. Just like DPA, a trigger is needed to prompt the software to begin taking traces of the cipher. Before traces are collected, an XY scan must be completed on the device to determine the precise location the cipher or process is executing. With the probes and capturing equipment programmed to this hotspot, the signal can be collected. Once captured, the data is analyzed and evaluated similar to that of a DPA attack.

2.1.10 Data Encryption Standard (DES)

The Data Encryption Standard was chosen for evaluation in this research for a variety of factors. First, the cipher is very well known, very easily implemented, and very easily broken. Although these would be cons when determining whether to implement this cipher in real-world applications, these reasons are what makes it a desirable cipher for research purposes. Because our purpose in implementing the cipher is not to break it, it is even more suitable for this research. Second, the DES cipher is very well supported in the Program Encryption Toolkit (PET) that this research will be utilizing to create both

native and variant circuits for evaluation. This toolkit is our circuit generator of choice, so it follows that we choose a cipher that is compatible. Lastly, side channels of DES ciphers are widely evaluated and are able to be straightforwardly analyzed because of the simplicity of the algorithm.

The Data Encryption Standard, often referred to as DES, is a cipher method formulated by researchers at IBM in the early 1970s and adopted into the United States government's National Bureau of Standards by the National Institute of Standards and Technology in 1977 [21]. It was the first encryption method to be endorsed by the United States government and used for classification at a federal level.

The algorithm for this method can be described in its simplest form as a substitution cipher where "an input block of 64 bits, regarded as a 'letter' in this alphabet, is replaced with a new letter, the output block" [22]. A graphical representation of this simple algorithm can be found in Figure 2.1 [23]. Essentially, the DES algorithm takes an input of 64 bits and a key of 64 bits and generates the ciphertext output of 64 bits. In the 64 bit key however, we can see that only 56 bits are used as the effective key and the remaining 8 are used as parity bits [24].

Within this algorithm, the encryption of the plaintext or input block is handled in 16 rounds meaning there are 16 identical iterations of the same encryption operation. Before the first round, the input block is fed through an initial permutation (IP) and divided into 32-bit halves. Within each round, the left and right halves of the input block are each operated on and fed into the next round with the only difference in the rounds being that the expanded right 32 bits are XOR-ed with a subkey. Each round uses a different subkey derived from the main 64-bit key. After the final round is completed,

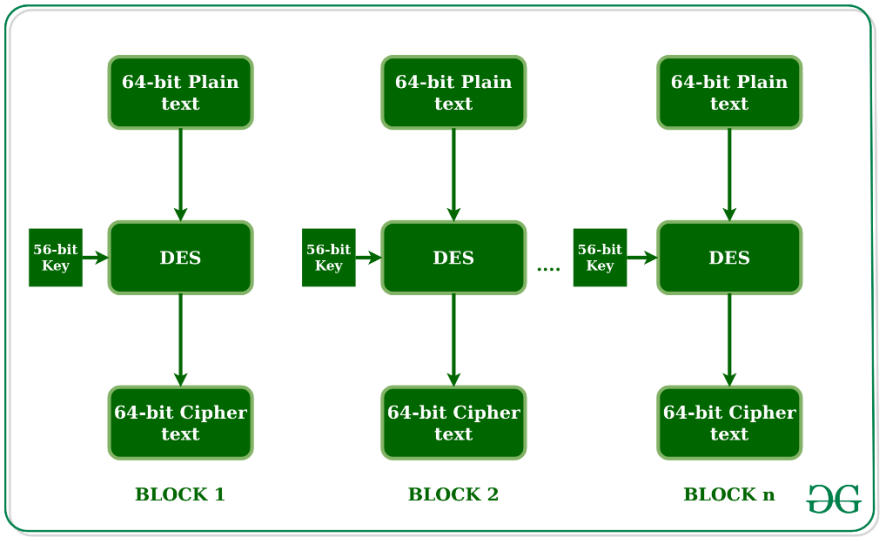

Figure 2.1 Basic DES Cipher [23].

“the right and left bits are concatenated and finally pass through a final permutation…which is the inverse of the initial permutation (IP)” and the subsequent output is the 64 bit ciphertext block [21], [25], [22], [24]. An image outlining this portion of the algorithm can be found in Figure 2.2 [26]. Because this encryption is a private key algorithm, the same key is used for both encryption and decryption meaning that the decryption process is identical to the encryption process save that the subkeys are used in reverse order [24].

For ease of access and the need for a variety of applications of DES, four different modes of operation were developed for the algorithm: Electronic Codebook (ECB), Cipher Block Chaining (CBC), Cipher Feedback (CFB), and Output Feedback (OFB). Each of these modes successfully implements DES encryption, but all in a different way and for use in different applications. OFB is often used for encrypting satellite communications while CBC and CFB are most commonly used to authenticate

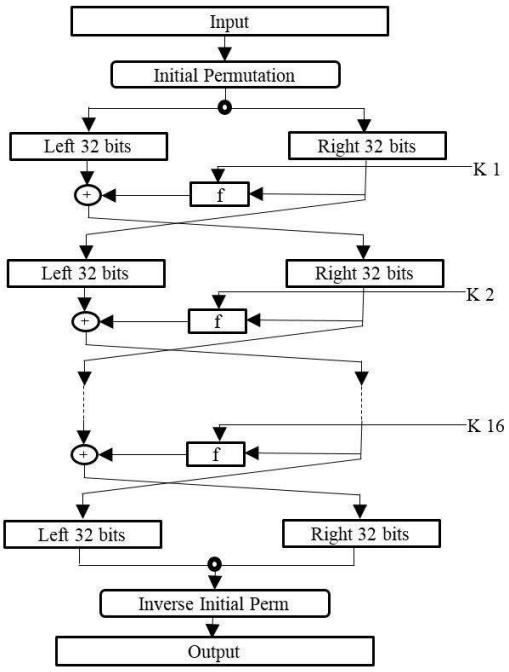

Figure 2.2 Detailed DES Overview.

data. [21], [25], [27]. Various applications of DES include “automated key management applications, file encryption, mail encryption, satellite data encryption, and other applications” [21].

Because of the way DES was developed and the fact that it is a publicly known algorithm, it has become a relatively easy encryption algorithm to break, making it vulnerable to attacks. Although DES is still widely used in a variety of applications and functions, in most cases it has been replaced with the newer and more secure Advanced Encryption Standard (AES) as well as others. A combination of the publicity of the DES algorithm and the easily identifiable 16 rounds of encryption contribute to why it has been chosen for side channel analysis in this paper. In addition to the vulnerability of the algorithm, it’s ease of implementation on reconfigurable hardware, specifically FPGAs,

is another motive for using such a cipher. The implementation of a well-known cipher will aid in analyzing the difference between native and variant circuits. In this work, it is important to remember our goal is not to break the cipher using SCA, rather it is to understand the effect of functional cipher variants on side channel characterizations; and for this reason, we have chosen DES for the analysis.

2.2 Related Works

In researching topics related to software protection and circuit variance, there were two papers that stood out as integral related works. The first is a conference paper by authors Madou et al. [28] entitled *Software Protection through Dynamic Code Mutation*.

This paper focuses on techniques for circuit variance. The second of these papers was a graduate thesis, *Enhanced Software Security through Program Partitioning*, written by L. Whitehurst. In her thesis, Whitehurst aims to provide a technique for program partitioning. The methods proposed in each of these papers provide the basis for the research completed in this thesis.

2.2.1 Software Protection through Dynamic Code Mutation

In their 2005 conference paper, Madou et al. [28] propose dynamic code mutation as a way to secure software. They begin their paper by outlining just how negative the aftereffects of reverse engineering can be on a system. They make it clear that one of the most effective ways of deterring hackers and the like from attempting to illegally obtain access to a system is to make the attack surface as small as possible by making the code as incomprehensible as it can be while still retaining functionality. In this paper, they

focus on obfuscated code variants that change dynamically while the program runs as the way to do this [28]. The authors suggest that although there has been some research on dynamic code generation, the techniques described in their conference paper are considerably different because the previous research was not done with the intent to thwart hackers and reverse engineering.

Madou et al. [28] consider two types of code mutation in their paper, one-pass mutation and cluster-based mutation which both work through the use of edit scripts. They define one-pass mutation as a code mutation technique where a procedure or piece of code is made into a template and "scrambled" until right before it is to be executed. At this time, the code is put back into its original form for execution. Cluster-based mutation on the other hand is defined as a technique whereby one procedure or piece of code generates several different variants or a cluster of variants that will all map to the same memory locations. They assert that the larger the cluster, the greater the degree of obfuscation because the large number of variants all mapping to the same memory location in turn creates more confusion and entropy [28].

In their experiment, the authors created a prototype of the code mutation technique using the tool Diablos. They found that using both obfuscation methods, both one-pass and cluster-based, they are capable of protecting 92% of all (non-library) procedures in a program [28]. From this and various other results, they concluded that their dynamic code mutation technique gave the intended results so long as the variants or random generator are not broken. The discoveries made by Madou et al. in their research lends better insight into the realm of software protection through obfuscation which in turn enables MTD at the hardware level.

2.2.2 Enhanced Software Security through Program Partitioning

In her graduate thesis, Whitehurst [12] proposes that partitioning a program between a general-purpose processor (GPP) and an FPGA can mitigate attacks that rely on a stack. She proposes that this method of placing software vulnerabilities onto an FPGA can eliminate vulnerabilities altogether, specifically software vulnerable to buffer overflow attacks [12]. In her problem statement, she states that although FPGA's have been used to speed up the security of a system and several other applications, they have not actually been used for the purpose of providing security to a system. This method would be useful because by placing the parts of a piece of software vulnerable to attacks that require address spaces onto an FPGA which has no addresses to attack, the overall program is made more secure [12].

To test this hypothesis, Whitehurst created a vulnerable program and showed the vulnerabilities to attacks on two GPPs. She then tested two more partitioned programs against these same attacks to directly compare the security against the buffer overflow attacks. In her results, she discussed how the vulnerable program that did not utilize the partitioning method did in fact succumb to the buffer overflow attack; but the programs which did utilize the partitioning method prevented the attack and were successfully secured [12].

CHAPTER III

METHODOLOGY

To complete this research and determine if our hypothesis can be supported there are several objectives that need to be met. Those objectives and their sub-objectives are below:

1. Obtain native DES circuit implementation and generate variants
    a. Use Program Encryption Toolkit to generate variants and translate from BENCH files to VHDL
2. Implement three distinct hardware variants of DES onto an FPGA
    a. Program FPGA with Xilinx's Vivado Development Kit(s)
3. Capture related side channels from both native and variant(s) under operation
    a. Utilize the Riscure hardware and software (Inspector SCA)
    b. Capture electromagnetic signatures as well as power consumption behavior
4. Evaluate/analyze differences between the signals
    a. Determine if the implementation of circuit variance changes the circuit enough that its signatures present as a completely different circuit.

3.1 Explaining the Approach

The work for this research has been carried out in three distinct tasks. Task I describes the process of ascertaining an original DES circuit implementation and the generation of its variants. Task II summarizes programming the native and variant hardware implementations onto an FPGA. And Task III outlines side channel analysis and comparison of these analyses.

3.1.1 Task I

This task describes how the various DES circuits were ascertained as well as the method of language translation for the circuits. The three circuits we use in this research were originally BENCH netlists obtained from thesis committee member, Dr. Jeffrey McDonald. These circuits are implementations of DES encryption generated from Dr. McDonald's Program Encryption Toolkit (PET). We believe the PET-generated circuits are implementing DES encryption, but these circuits cannot be verified against other implementations of DES. It seems the input/output relationships within these circuits contain different byte ordering than those employed in verifiable DES implementations. Although the generated ciphertext from these circuits does not directly resemble DES encryption, the ciphertext for both native and variant circuits are equivalent. Therefore, we will continue utilizing these circuits in this research as the fundamental question and hypothesis center around side channel characterizations of functionally equivalent circuits and not the functionality of the circuits themselves.

Of these circuits, the native is the simplest of the three, and the variants are functional equivalents to the native circuit with added redundancies for obfuscation. The redundancies in the variant circuits were created by inserting multiple AND-trees to the

logic of the native circuit while allowing it to keep its original functionality. AND-trees are essentially trees that perform the logical AND operation upon all inputs into the tree [29]. With circuit inputs as inputs to the AND-trees, the addition of these trees adds only redundancy into the circuits. The functionality and integrity of the circuits themselves remains unchanged and it is for this reason they were chosen for obfuscation in this research. The two variants differ in that one uses 20 AND-trees for obfuscation and the other uses 50. Hereafter the native, 20 AND-tree variant and 50 AND-tree variants will be referred to as Native DES, Variant DES_20, and Variant DES_50 respectively.

Because the circuits were originally in BENCH netlist format, they need to be translated before they are used to program the FPGA. To do this, PET is again utilized. This software, while it has many functions, is used in this research to take the original netlists of the circuits, decompose any multi-fan-in, and translate them into VHDL so that they can be used in the Vivado Design Suite. The translation could be done by hand, but because the circuits all have upwards of 20,000 total gates, utilizing PET makes this process more efficient and reliable. A flow chart of this process can be found in Figure 3.1.

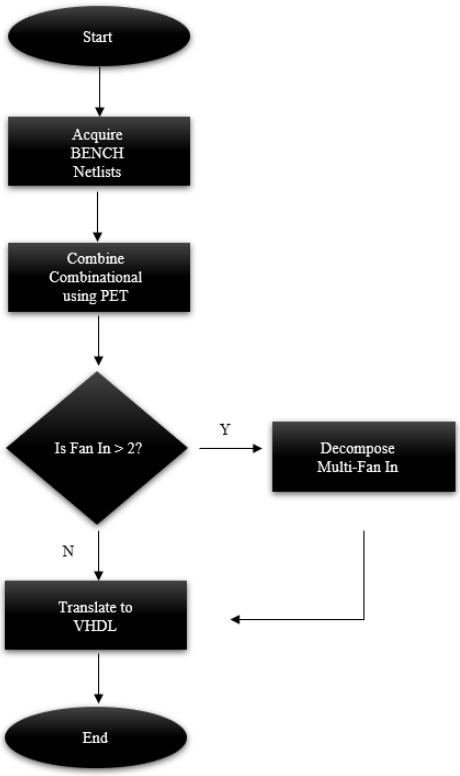

Figure 3.1 PET Translation Flow.

3.1.2 Task II

The following section summarizes the process of packaging the DES circuit(s) and programming them onto the Zedboard's FPGA. The details of the development environment can be found in Table 3.1.

Once Native DES, Variant DES_20, and Variant DES_50 were ascertained and translated to VHDL, they were then ready to be programmed onto an FPGA. The Vivado Design Suite was utilized to accomplish this. The initial programming of the board included producing a block diagram of the system, creation of a new IP for the project in the Vivado Design Suite, instantiation of the VHDL circuit, synthesis, implementation, and bitstream generation.

Table 3.1 Development Environment

Task	Hardware/Software Utilized	Reference
Language Translation	Program Encryption Toolkit	[30]
SoC / FPGA	Zynq-7000 SoC XC7Z020-CLG484-1	[31]
Design Suite	Xilinx Vivado Design Suite 2017.1	[32]
Software Development	Xilinx Vivado Software Development Kit 2017.1	[33]

Of the above steps for hardware programming and specification, the creation and instantiation of the IP requires the most hands-on approach and attention to detail. In the IP creation process, the VHDL translated circuits are used to create a new IP with inputs, outputs, and logic all packaged into one module. It is important to note that the logic within this module is tagged as “dont_touch” to keep the synthesizer from optimizing the circuit and removing the redundancies intentionally inserted in the logic. This module is then able to be inserted into a block diagram along with the Zynq ARM processing system and other modules necessary for the circuit to execute. A depiction of the block diagram created and utilized in for Native DES can be found in Figure 3.2. Here the created IP for Native DES (“des_ip_0” in the diagram), the Zynq processing system, processor system reset, and AXI interconnect, GPIO (pin trigger), and their interconnections are all clearly visible. With the exception of the IP utilized for Native

Upon completion of the hardware specifications, the project and most importantly the generated bitstream are then exported to Xilinx’s Software Development Kit (SDK) for the last steps in programming the FPGA. In the SDK, the FPGA is programmed to receive serial communication through the UART. A simple “HelloWorld” file is created

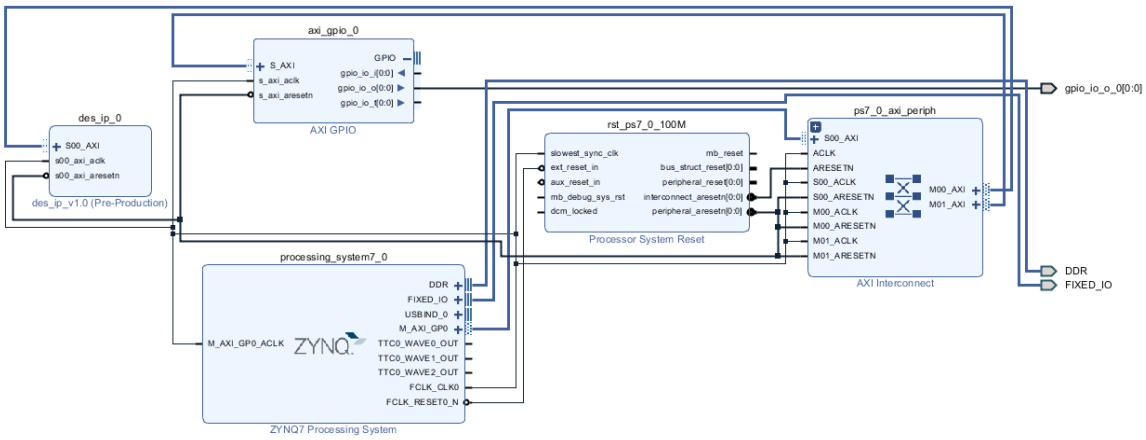

Figure 3.2 Native DES Block Diagram.

and written to initiate the encryption specified by the IP in Vivado (found in Appendix A). The board's UART is then connected, the board is programmed, and the ciphertext is visible in the SDK's terminal. Figure 3.3 shows the SDK terminal of Native DES just after encryption was initiated on the device. The project in the SDK is then transferred to a boot loader so that the device can be programmed from flash memory and remain programmed when power cycled. These same steps are taken to program all three circuits onto the Zedboard.

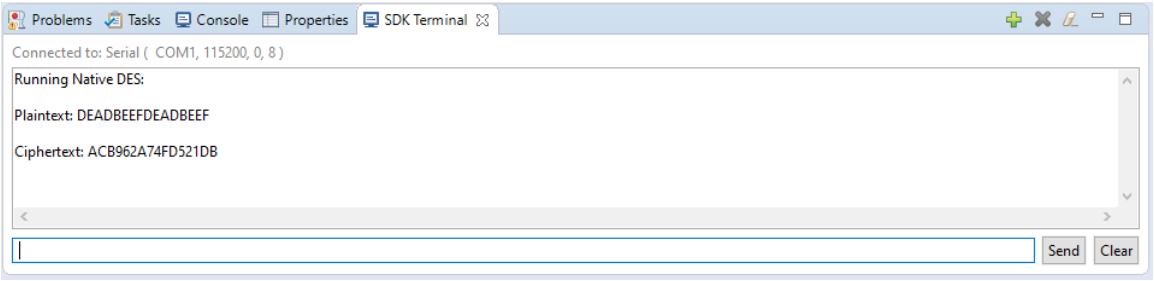

Figure 3.3 Native DES SDK Terminal.

3.1.3 Task III

This section outlines the process of gathering side channel traces and the subsequent comparison. For the analysis of these circuits, both power and electromagnetic traces are collected. Riscure's Inspector SCA is used to carry out these analyses and provide the traces necessary to make comparisons between Native DES, Variant DES_20, and Variant DES_50. Although the purpose of the research is not to break the DES encryption and ascertain the key, the traces gathered will provide insight into the inner workings of the circuits, which we believe will highlight their differences.

Before these traces can be collected, a trigger must be created. This is what alerts the software to begin taking traces of the system. Although not every analysis requires a trigger, it is necessary for this research since multiple circuits are analyzed and compared from beginning to end of the encryption process. The simplest and most easy to implement trigger is one that signals the software to begin capturing when a pin goes high and end capturing when that same pin goes low. The pin trigger will be utilized in both the power and electromagnetic analyses and can be seen in Figure 3.2 as the GPIO IP. The trigger can also be used to determine the execution time (t_e) of each circuit. Because a pin is set to go high just before encryption starts and low just after it ends, t_e can easily be ascertained by analyzing the width of the square wave pulse generated during every execution cycle. This calculation is essential to collecting useful traces because a precise execution time ensures traces gathered include the entire encryption process and not just bits and pieces.

With t_e determined, the software must be set up to take measurements of the device. The first acquisition measurements to be defined are the oscilloscope

specifications. The first of these is the sample frequency (f_s). Sample frequency is the number of samples taken per second of the signal and must be at least twice the operating frequency of the device. For our device, our sample frequency must be at least 200MHz, so for our acquisition the closest frequency greater than or equal to twice our operating frequency is 250MHz. The next acquisition measurement to be calculated is the number of samples to be taken. This figure is dependent on the sample frequency and the execution time of the implemented design. It can be evaluated using the formula below,

$$Num\ Samples = (f_s)(t_E).$$

The next acquisition measurements to specify are the trigger properties. The trigger is connected to the external channel of the scope so this should be specified in the acquisition. The voltage level must be set to 300mV and the slope set to rising edge to guarantee the software catches the voltage spike from the trigger when encryption starts. The delay and timeout values should keep their default values, 0.00s and 1.0s respectively.

The channel properties for the PicoScope are specified next. These measurements are specific to the type of measurement being taken (i.e. power, electromagnetic, etc.). For collections of power traces, the voltage range must be set to 20mV, the offset set to 0.0V, the bandwidth set to “Full”, and the coupling should be “DC”. This will ensure the traces taken yield the best view of the changing power measurements across the device. For electromagnetic traces on the other hand, the voltage range must be at least 500mV to capture the full trace and reduce clipping. The offset voltage, bandwidth, and coupling specifications for electromagnetic traces are the same as those for power traces. The

specific acquisition measurements for each trace collected are consolidated into tables in the next section.

Next in the trace collection process is the setup of the FPGA with the PicoScope and Riscure Inspector SCA. A representation of the experimental setup is shown in Figure 3.4. Here for power measurements, a differential current probe is placed in series with the power supply and PicoScope where the probe measures the current across the device's Current Sense. For electromagnetic measurements, the current probe is replaced with an electromagnetic probe to capture emanations across the device. This system will hereafter be referred to as system under test (SUT).

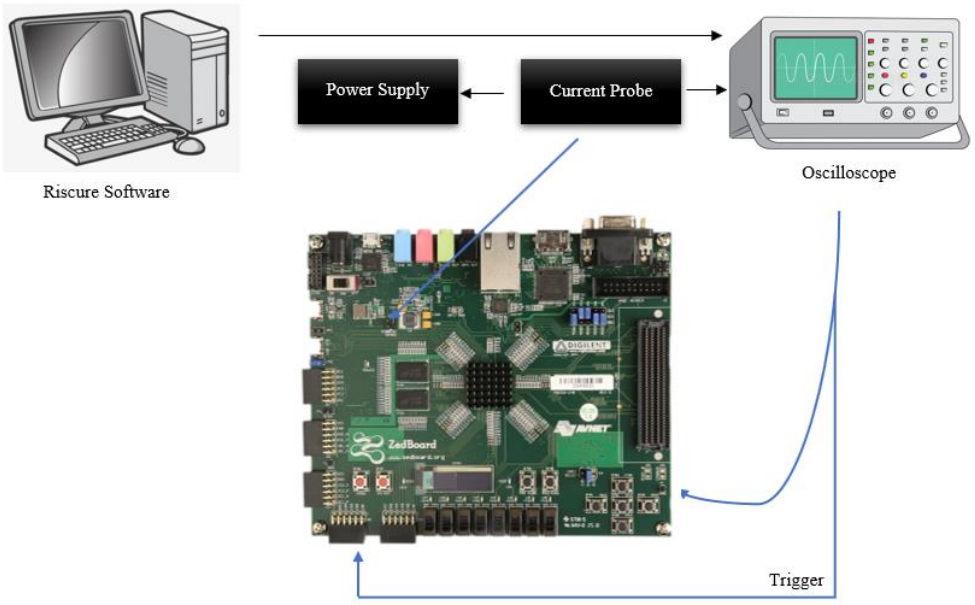

Figure 3.4 System Under Test (SUT).

Once analysis begins, traces of the encryption are generated. In this process, Inspector SCA takes “snapshots” of the system during different periods of the encryption process. These “snapshots” of power consumption and electromagnetic signatures are what will be used to compare the native to its variants. Sample traces obtained from Riscure SCA tutorials can be found in Figure 3.5. The data comes from a power trace of a software implementation of DES analyzed using Riscure’s Inspector SCA, which is the proposed analysis tool for this research. The figure shows an initial power trace of Software DES and clearly indicates the 16 rounds within DES encryption.

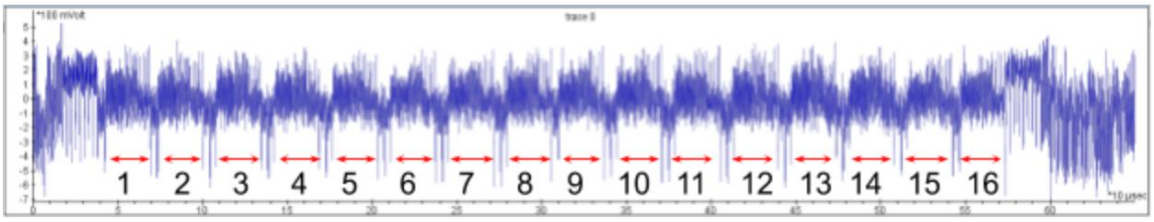

Figure 3.5 16 Distinct Rounds of DES [35].

With both power and electromagnetic traces gathered, they can then be compared, Native to Variant DES_20, Native to Variant DES_50, and Variant DES_20 to Variant DES_50 as needed.

CHAPTER IV

RESULTS

In the experimentation process, we collected and analyzed data in both the Vivado Design Suite and the Inspector SCA software. In Vivado, the data gathered focused more on hardware requirements, utilization, and design implementation of the DES circuits. Results compiled from Inspector SCA on the other hand centered around the collection of the power and electromagnetic side channels for each circuit. This section will examine in detail the data gathered in both Vivado and Inspector SCA. The remainder of this section is as follows, Section 4.1 will analyze the observations and data collection from the Vivado Design Suite; and Section 4.2 will evaluate those from Inspector SCA.

4.1 Observations in Vivado

Within the Vivado Design Suite, various useful reports and graphs are generated once synthesis and implementation of the design are completed. This section will examine several of these reports and any initial conclusions that may be drawn from them. For each report, Native DES, Variant DES_20 and Variant DES_50 will be examined and compared in that order.

In Vivado when a design is created, it is packaged and inserted into a block diagram. From here, the design must be synthesized and implemented to ensure it has

been correctly employed on the device and does not exceed the limits of the hardware. During the implementation process, the design goes through logical, physical, and power optimizations to provide the most efficient logic cell placement and routing on the device. Because the bulk of this research depends on logical redundancies to create obfuscation in both Variant DES_20 and Variant DES_50, it is imperative that this optimization does not take place. The removal of the optimization step in implementation involves a simple Python script that places a “dont_touch” attribute on wires within the source VHDL for each circuit (found in Appendix B). This ensures that the optimizer does not change any of the logic within the source file therefore keeping the obfuscated logic intact.

4.1.1 Before Optimization Removal

Although optimization is not necessary for this research, some reports from the optimized designs do lend some insight into the overall design of the circuits and their similarities. For example, the Device View of each design shows the Zynq-7000 SoC with highlighted areas representing the utilized logic cells and their placement on the chip. A zoomed in view of a highlighted section can be found in Figure 4.1 where the utilized logic cells are look up tables (LUTs).

Keeping this in mind, we can compare the Device View of the implemented design for Native DES to that of Variant DES_20 and Variant DES_50 shown in Figure 4.2, Figure 4.3, and Figure 4.4 respectively. In these figures it is important to note that the orange highlights in the design are located where the Zynq processor is on the chip. Conversely, the blue highlights are in the dedicated portion of the chip for programmable

logic (PL). Here from initial observations, when optimization is performed on the designs, the PL of both variants very closely resemble that of our native circuit,

Figure 4.1 Logic Cells.

with some small variations in logic cell placement. This is expected as the only difference in the three circuits is the addition of AND-trees in Variant DES_20 and Variant DES_50 while retaining original functionality.

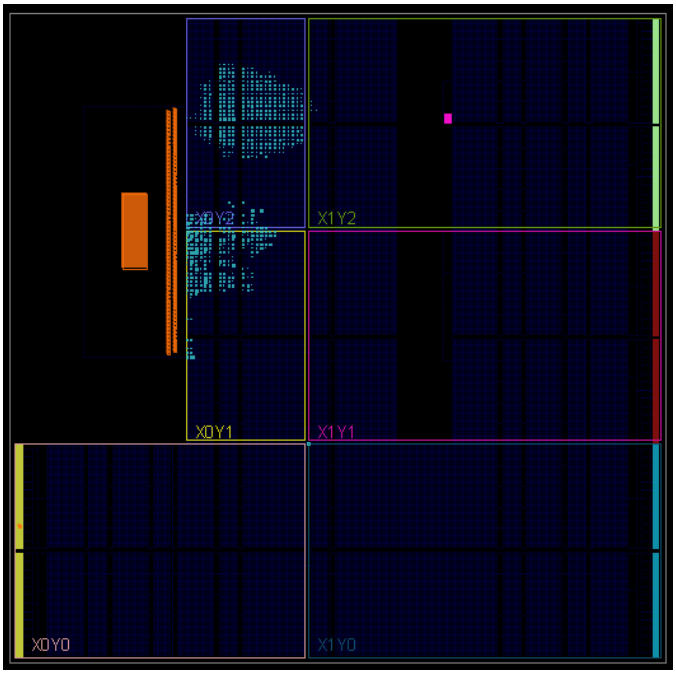

Figure 4.2 Device View: Native DES with Optimization.

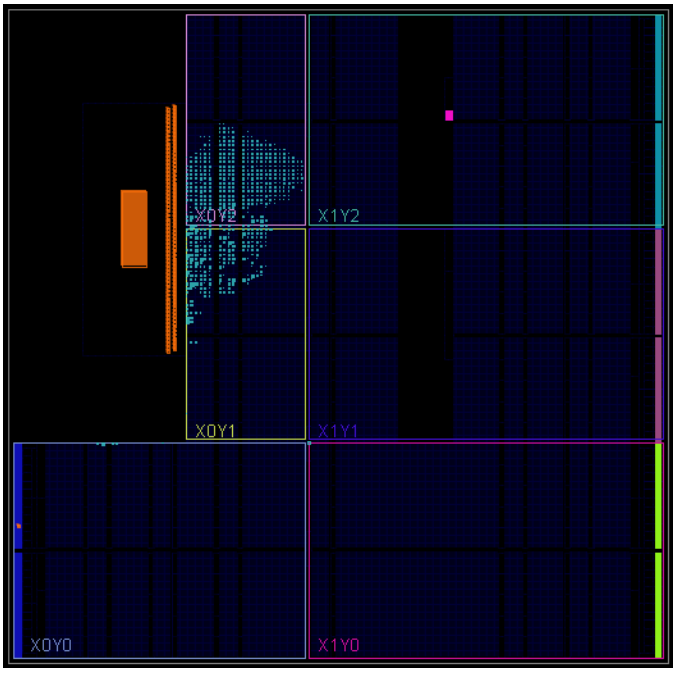

Figure 4.3 Device View: Variant DES_20 with Optimization.

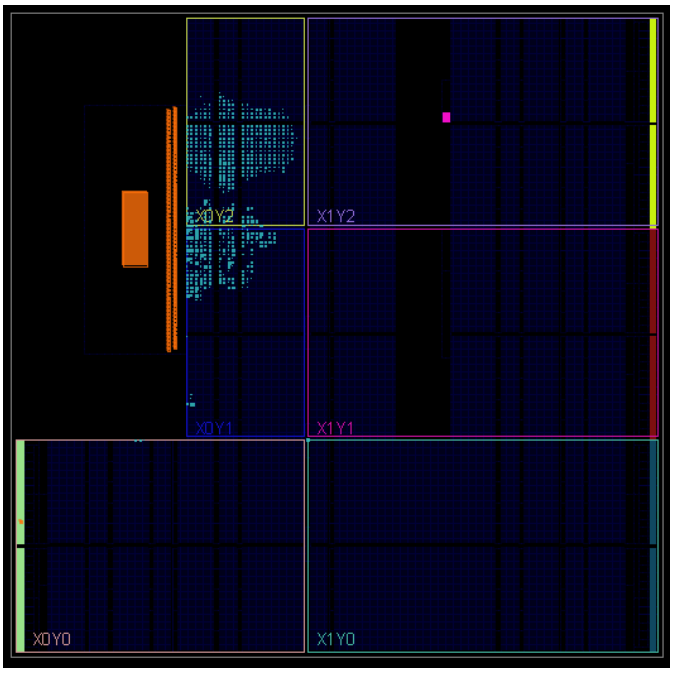

Figure 4.4 Device View: Variant DES_50 with Optimization.

4.1.2 After Optimization Removal

As mentioned in the previous section, to keep the development software from optimizing our variant circuits, a script was utilized that added in “dont_touch” attributes to all wire declarations in the design. This prompted the optimizer to skip over these parts of the logic in the implementation process. The reports that were generated as a result will be examined below. We will begin by again comparing the Device View of all three circuits after implementation. The Device View for Native DES, Variant DES_20 and Variant DES_50 can be found in Figure 4.5, Figure 4.6, and Figure 4.7 respectively. From these images we can see that all three circuits look different from one another in logic placement and size. In comparing these images to those in the previous section, the difference optimization makes in circuit design and logic placement is evident. Without

the optimization that would normally occur, the circuits retain their size and redundancies allowing them to present as three functionally different circuits as opposed to ones that are functionally equivalent. Interestingly, if we look closely at the logic cell placement for the variant circuits, we can see parts of our original Native DES in the sections marked X0Y1 and X0Y2. These similarities are expected since Variant DES_20 and Variant DES_20 do still contain the basic logic of Native DES with only AND-trees added for obfuscation. Even with these similarities in logic placement, the circuits appear to be distinct designs and not variant implementations of the same circuit.

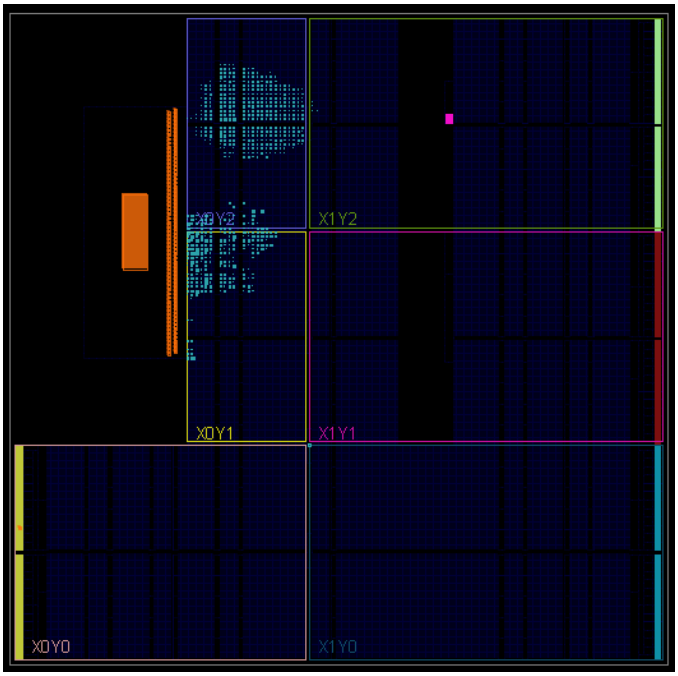

Figure 4.5 Device View: Native DES.

A comparison can also be made from the chip utilizations of each circuit. For each design, a utilization report is generated that provides specifics on the utilized slices

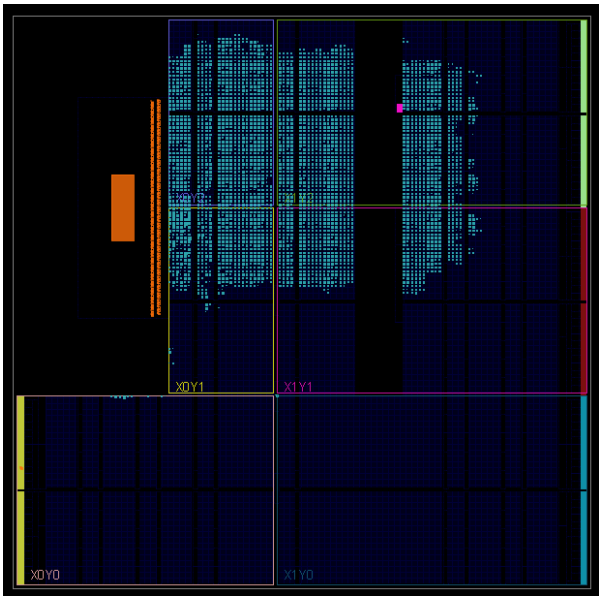

Figure 4.6 Device View: Variant DES_20 without Optimization.

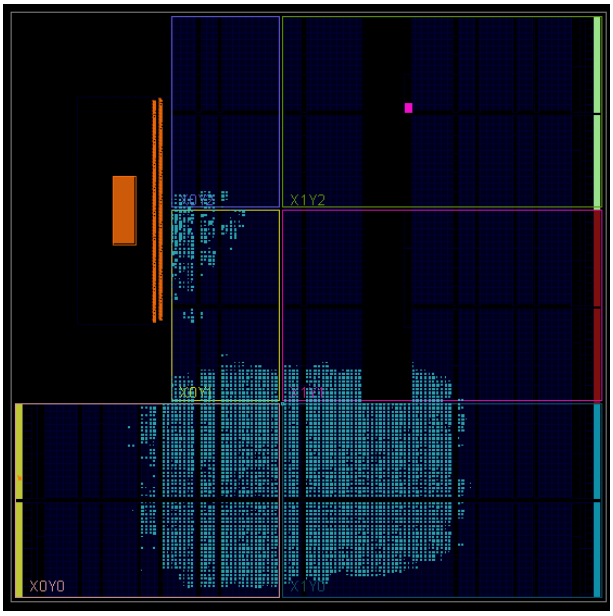

Figure 4.7 Device View: Variant DES_50 without Optimization.

in the PL of the device. In the PL, a configurable logic block (CLB) in the Zynq-7000 SoC is made up of two slices where slices consist of a combination of LUTs and flip

flops (FFs). The hardware resources an FPGA has is determined by the number of slices the device has [34]. Table 4.1 outlines the various logic cell types available on the chip as well as their utilizations for both native and variant circuits. In the table, we can see that from Native DES to Variant DES_20 and Variant DES_50 there is a significant increase in total utilization. The first Variant DES_20 is approximately 5.29 times larger than Native DES when placed on the chip while Variant DES_50 is around 5.55 times larger than Native DES and just slightly larger than Variant DES_20 (roughly 5% larger).

In this table, the main contributors to differences in utilization totals are the LUTs used as logic within the PL of the device. The amount of LUTs as logic used in Variant DES_20 is roughly 7.38 times more than those used in Native DES. Similarly, the LUTs used in Variant DES_50 are around 7.78 times more than that of the Native DES implementation. Again, as expected, Variant DES_20 and Variant DES_50 employ considerably more LUTs as logic than Native DES which can be attributed to the redundancies added into the logic for obfuscation. The LUTs as memory, registers as flip flops, and I/O (input/output) for each circuit yield the same values, so it follows that these areas of the chip include slices that are required for the base functionality of each DES implementation.

Table 4.1 Device Utilization without Optimization

	Site Type	Used	Available	Utilization (%)	Total Utilization (%)
Native DES	**LUTs as Logic**	1898	53200	3.57	5.31
	LUTs as Memory	62	17400	0.36	
	Register as FF	938	106400	0.88	
	I/O	1	200	0.5	
Variant DES_20	**LUTs as Logic**	14021	53200	26.36	28.1
	LUTs as Memory	62	17400	0.36	
	Register as FF	938	106400	0.88	
	I/O	1	200	0.5	
Variant DES_50	**LUTs as Logic**	14771	53200	27.77	29.51
	LUTs as Memory	62	17400	0.36	
	Register as FF	938	106400	0.88	
	I/O	1	200	0.5	

4.2 Observations in Inspector SCA

Riscure's Inspector SCA is used in this research to capture both power and electromagnetic side channels of the SUT during execution. As stated in Chapter III, to begin taking traces of a device, the acquisition measurements must first be defined and set within the Riscure software. The first of these required calculations is the execution time for each circuit. To determine this value, we connected our device, which had the circuit design booted from flash memory, to the PicoScope. With the device on and executing in an infinite loop, we could clearly see the square wave pulse generated by the pin trigger in the scope software. A snapshot of one of these pulses can be seen in Figure 4.8.

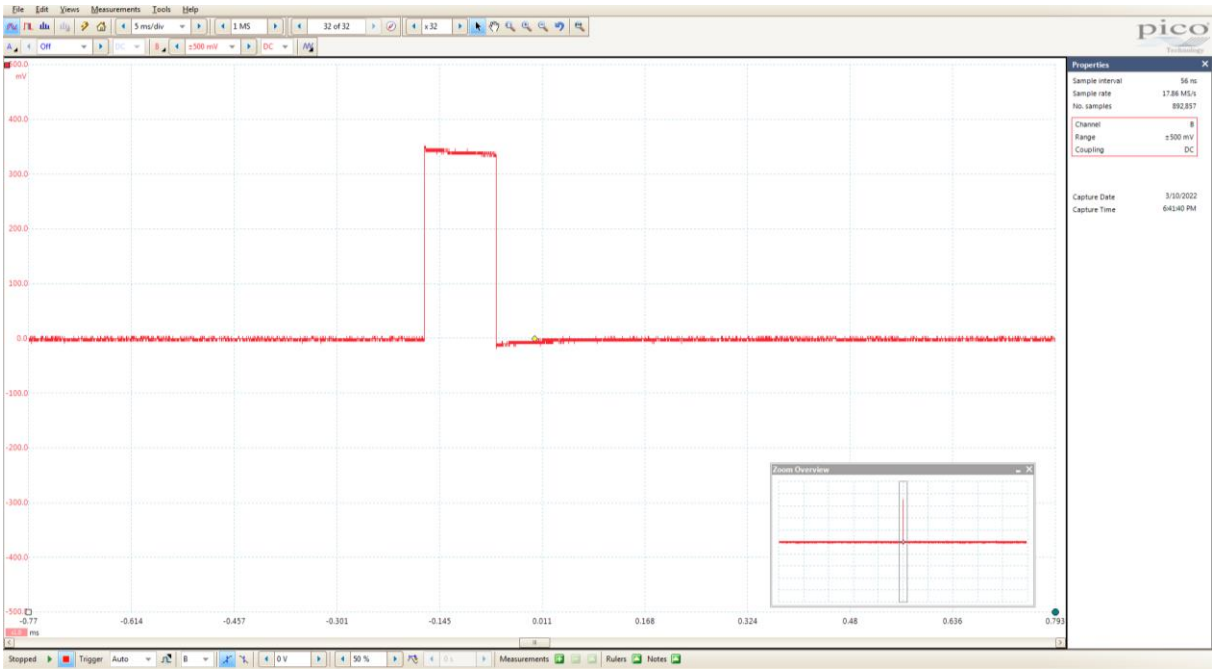

Figure 4.8 Execution Time Measurement.

This process was repeated for each circuit implementation to determine the execution time. Although the exact execution time of each design was calculated, this value is not what was utilized to compute “Num Samples”. Because we want to ensure the entire encryption process is included in each trace, we rounded the encryption times up to the nearest tenth of a millisecond. This guaranteed at least one and at most two iterations of encryption could be seen in our collected traces. Once this value was determined, the other acquisition measurements were defined. These values can be found in Table 4.2.

Table 4.2 Trace Acquisition Measurements

	Circuits	Calculated (ms)	Utilized t_s (s)	f_s (MHz)	Samples	Traces
Power	Native DES	0.1094	0.0002	250	55000	10
	Variant DES_20	0.10838	0.0002	250	55000	10
	Variant DES-50	0.10932	0.0002	250	55000	10
	Native DES	0.1094	0.0002	500	100000	10
	Variant DES_20	0.10838	0.0002	500	100000	10
	Variant DES-50	0.10932	0.0002	500	100000	10
Emag	Native DES	0.1094	0.0002	250	55000	10
	Variant DES_20	0.10838	0.0002	250	55000	10
	Variant DES-50	0.10932	0.0002	250	55000	10

All samples collected adhere to the acquisition measurements outlined in the table. The subsequent traces and scans are analyzed and evaluated in this section. The power side channels of the native and variant circuits will be examined first followed by the electromagnetic side channels and subsequent traces.

4.2.1 The Power Side Channel

In collecting the power traces for this research, the system was set up just as that in the SUT depicted in Chapter III, and the acquisition measurements used were those in Table 4.3. The first set of traces gathered had a sample frequency of 250MHz and 55,000 samples. The resulting trace for Native DES is shown in Figure 4.9.

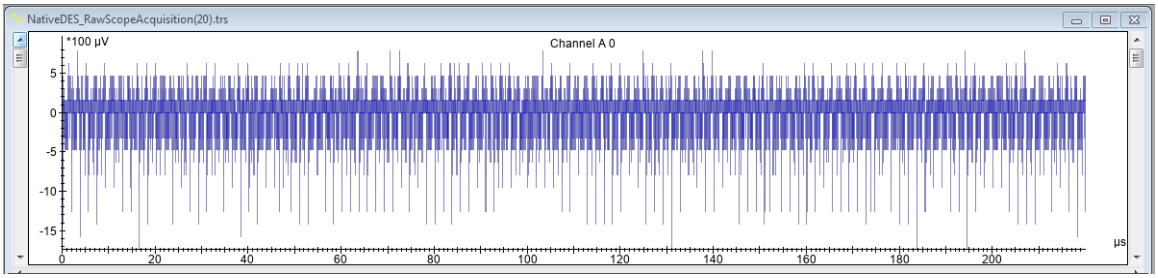

Figure 4.9 Power: Native DES – 250MHz.

Because this trace is very noisy and not much can be inferred in its raw state, we applied both a moving average filter and spectrum analysis to make things a bit clearer. A moving average filter is one that analyzes data points by calculating various averages of subsets of the full dataset. With this filter we can average the samples in the trace and therefore reduce the noise in the raw collection. The Native DES power trace with the moving average filter applied is in Figure 4.10.

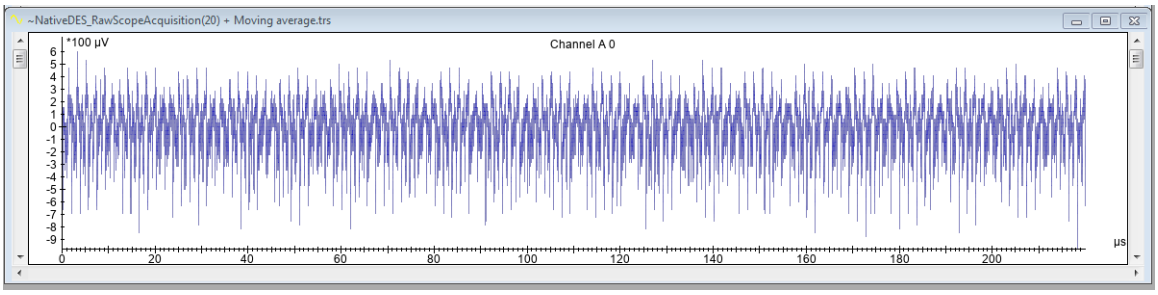

Figure 4.10 Power: Native DES – 250MHz – Moving Average.

Here we can see that the trace is much clearer but still very noisy with little to no clear indicators of the encryption on the device. Figure 4.11 shows the Native DES power trace with spectrum analysis applied. This process analyzes the frequency spectrum of a trace and verifies clock frequency. The resulting trace gives us an examination of time series within the frequency domain [35].

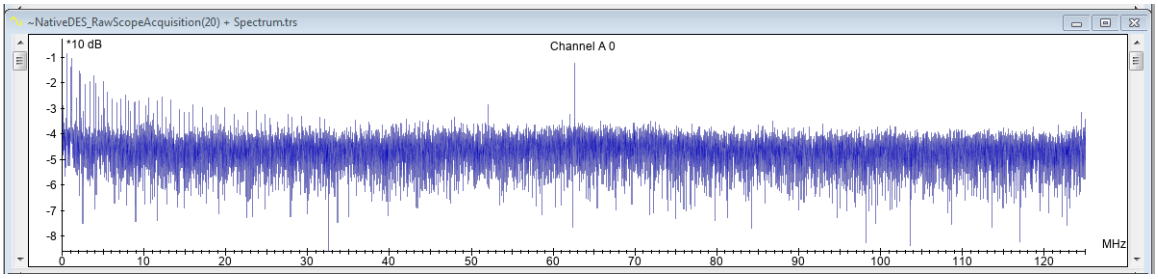

Figure 4.11 Power: Native DES – 250MHz – Spectrum.

In this trace there are various peaks present, but the most interesting is the one near 63MHz. This peak tells us that there is some significant activity going on in the chip near this frequency. We believe that this activity is the encryption running on the device,

but this cannot be verified as the Zynq-7000 processor is very noisy and is controlling more than just our implemented design.

Comparable to the traces collected for Native DES are those compiled for Variant DES_20 and Variant DES_50. The same filter and analysis conducted on Native DES were performed on each variant. The raw trace, spectrum analysis, and moving target filter for Variant DES_20 can be found in Figure 4.12, and those for Variant DES_50 are depicted in Figure 4.13. Similar to those in Native DES, the raw traces for each variant are noisy and almost indecipherable. The moving average filter applied on the raw traces for each variant again makes the traces clearer but are still difficult to read. The spectrum analysis conducted on the traces, on the other hand shows us a similar graph to that generated in Figure 4.11. We can clearly see a distinct peak near 63MHz for each variant. Because this is present in each implementation, we believe this represents our encryption but still cannot verify this with these power traces. In comparing the filtered and analyzed traces to each other directly, it is difficult to distinguish one circuit from another. This does not altogether disprove our hypothesis because we believe this is due to either the influence of the processor creating added noise in the traces or the extremely small changes in current through the current sense that we are measuring over.

The second set of power traces collected had a sample frequency of 500MHz and took 100,000 samples. With this sample frequency the traces should have 2 iterations of encryption present in them. These traces, like the first set, all had noisy raw traces and needed filters and analysis to make them more legible. Both the moving average filter and spectrum analysis were applied to Native DES, Variant DES_20 and Variant

DES_50 traces at 500MHz. These traces, filters and analyses for native and variant circuits are shown in Figures 4.14, 4.15, and 4.16.

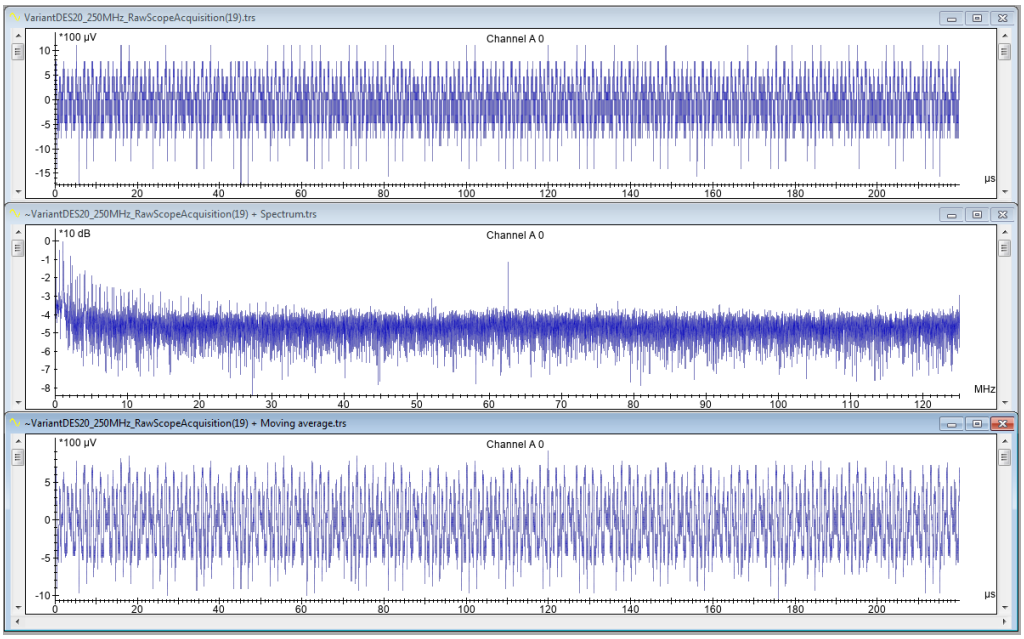

Figure 4.12 Power: Variant DES_20 – 250MHz.

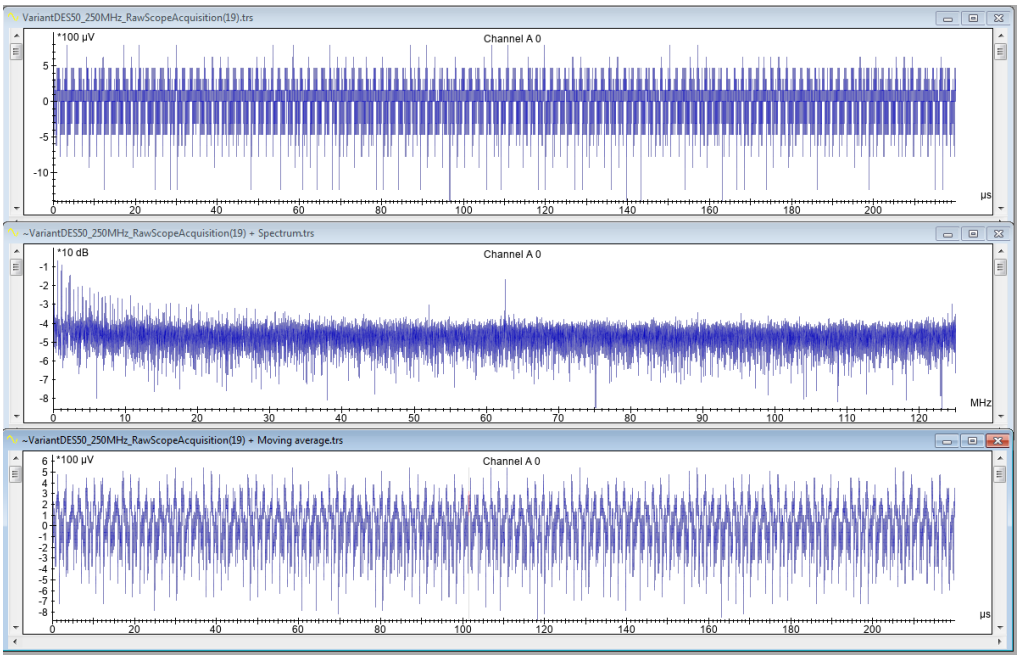

Figure 4.13 Power: Variant DES_50 – 250MHz.

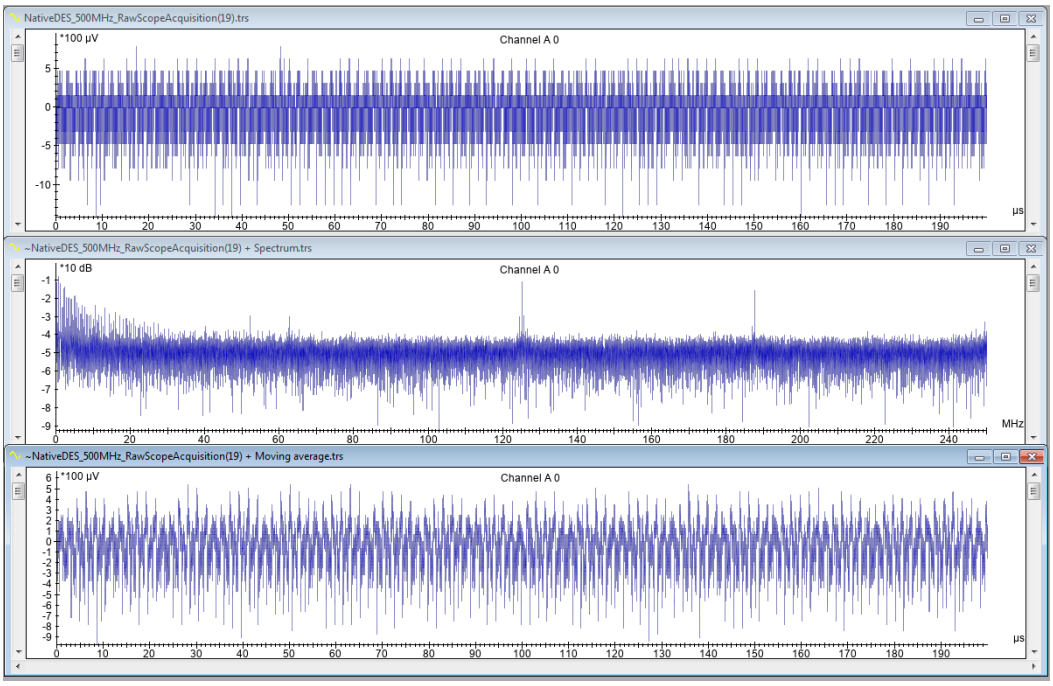

Figure 4.14 Power: Native DES – 500MHz.

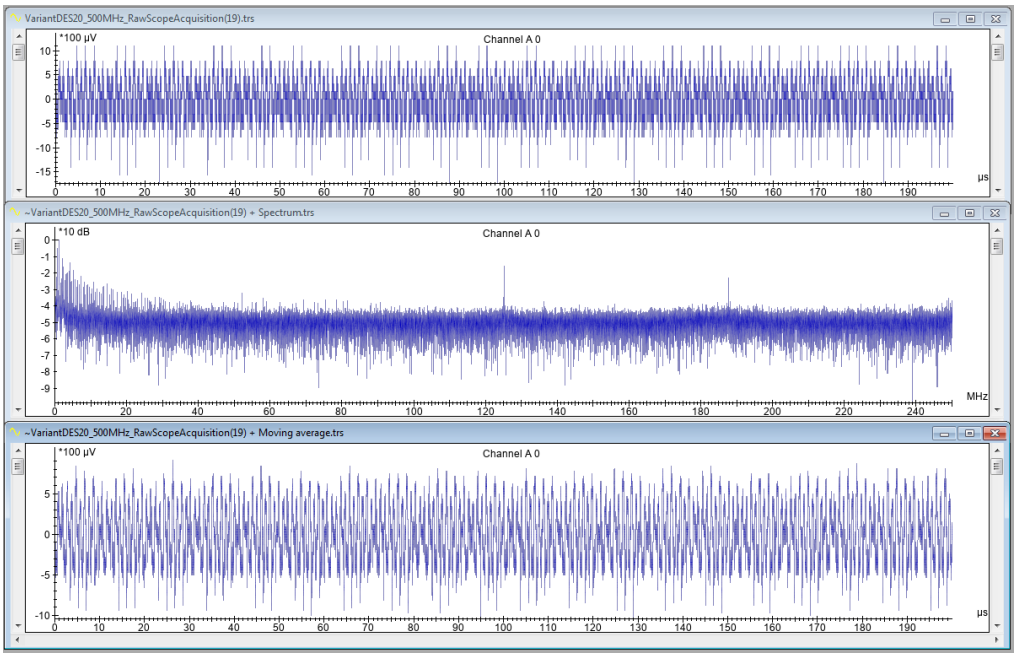

Figure 4.15 Power: Variant DES_20 – 500MHz.

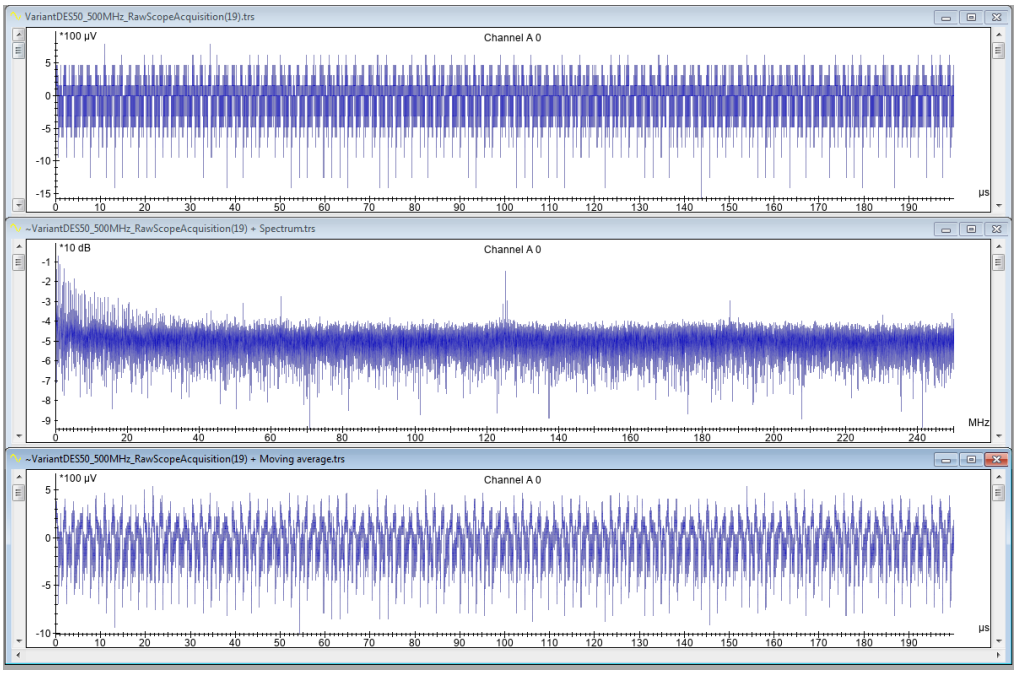

Figure 4.16 Power: Variant DES_50 – 500MHz.

In each of the traces generated from the spectrum analysis, two distinct peaks are present. If we look at them a bit closer, it seems that they are first and second harmonics. One large peak is near 63MHz and a smaller peak is located around 188MHz. As in the previous set of traces, we believe these two peaks are representative of the two rounds of encryption on the device within the selected period. In addition to this similarity, there are little to no major differences in the graphs from one circuit to another. For this reason, it is difficult to declare these are three separate circuits if previous knowledge about the logic is not readily available. Again, we believe this is due to the Zynq-7000 processor. Therefore, the bulk of our analysis and conclusions will be based on the electromagnetic side channel measurements as these are based on emanations throughout the device and not the power consumed during execution. These samples are examined in the next section.

4.2.2 The Electromagnetic Side Channel

The electromagnetic traces obtained for this research were collected using a slight variation of the SUT where the current probe is replaced with the electromagnetic probe. Before these traces could be taken, an XY scan of the device had to be collected first. The XY scans produce 20x20 spectral intensity graphs that indicate various hotspots on the device. Once all four hundred device locations and their spectral intensity are gathered, traces can be taken on user selected areas of the scan. The spectral intensity scans for Native DES, Variant DES_20 and Variant DES_50 are shown in Figures 4.17, 4.18, and 4.19 respectively. For each scan, the locations of the hotspots tell us that each design is executing in approximately the same area of the device. If we compare these scans to the

logic cell placement found in the device view for each design (Figures 4.5, 4.6, and 4.7), it is evident that the hotspots align with the placement of logic blocks on the device.

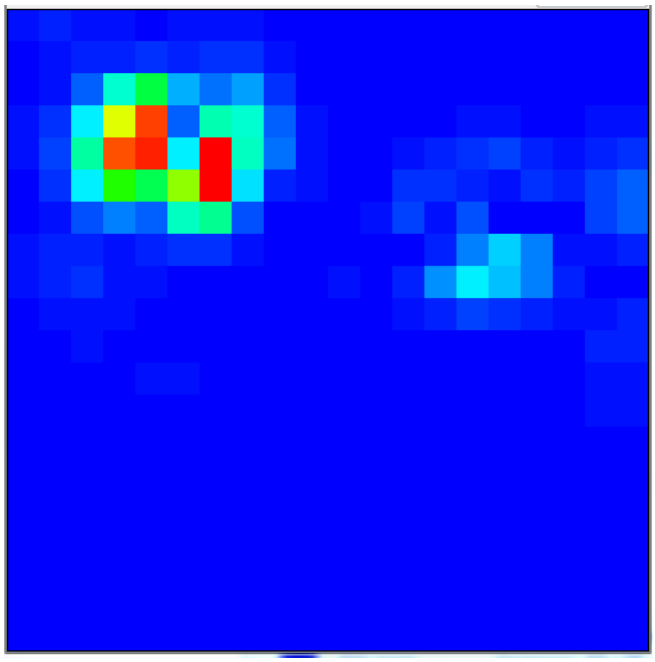

Figure 4.17 Spectral Intensity – Native DES.

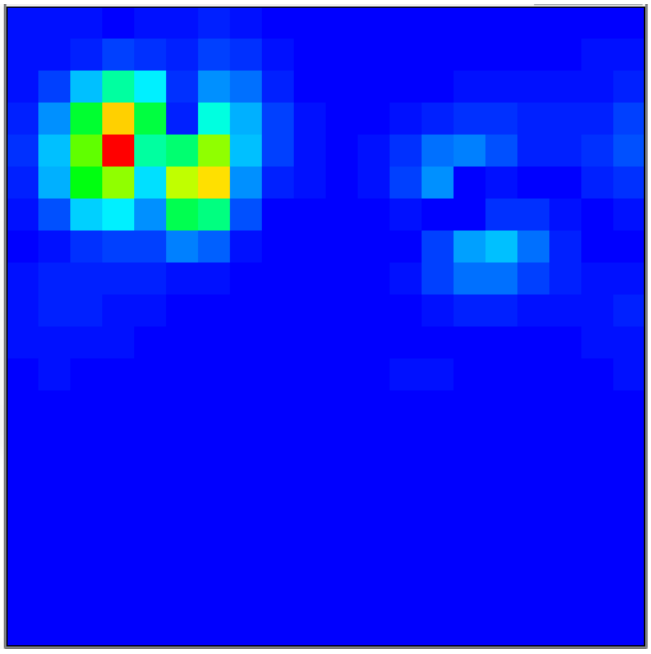

Figure 4.18 Spectral Intensity – Variant DES_20.

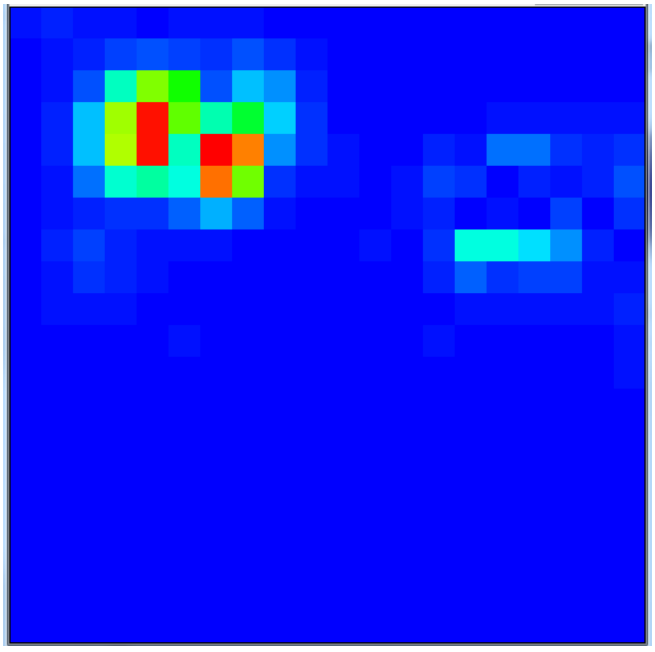

Figure 4.19 Spectral Intensity – Variant DES_50.

For the trace collections and subsequent comparisons, the traces to be collected were determined based on hotspots in the Native DES spectral intensity scan. Because we are comparing the variants to the native implementation, it follows that we compare the same locations while each device is executing. The sites on the scan we decided to collect are site 84, site 89, and site 104 which all fall within or near the center of the hotspots for Native DES. The location that produced the clearest traces was site 89 so this will be the focus of our analysis.

The traces had a sample frequency of 250MHz and 55,000 samples. The raw trace for Native DES at site 89 can be found in Figure 4.20. This trace has several peaks and troughs approximately 7.5µs apart, but this width is too small for these to be the triggered encryption on the device.

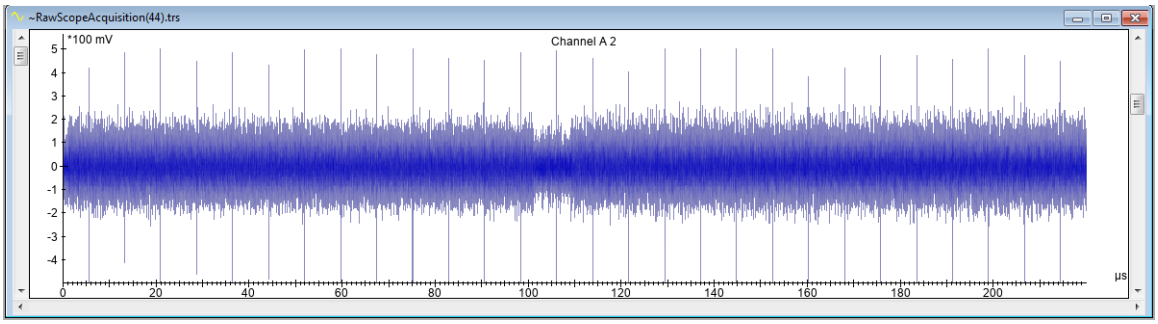

Figure 4.20 Emag: Native DES.

In addition to these peaks and troughs is a small section in the middle of the trace that is about 8µs in length. A zoomed in view of this section of the trace can be seen in Figure 4.21. At first glance it seems like this may be the encryption executing on the device, but again the length of the section is too small for the execution time we

calculated for Native DES. However, it is possible that this small section represents only a part of the execution, and the remainder lies within the noise on either side of the highlighted area. Because the site this trace was gathered from lies right in the heart of the device's hotspot, we do believe that the encryption does lie within the trace, but without several filters and analyses it is difficult to definitively make that assertion.

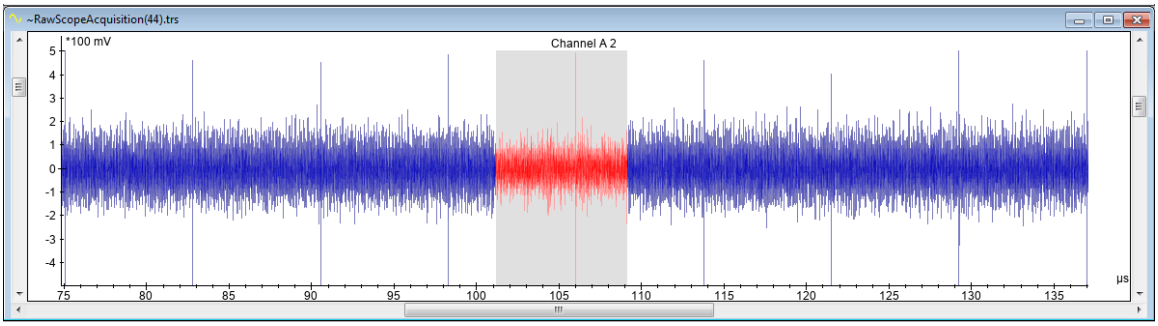

Figure 4.21 Emag: Native DES – Zoomed.

To clear up some of the periodic peaks/troughs in the raw trace, a harmonics filter was applied. The resulting trace is shown in Figure 4.22. In this trace we can see that those peaks/troughs are no longer visible on the graph. Although cleaned up considerably, the trace still needs some more maintenance, so spectrum analysis was applied to the harmonics trace. This new spectrum analysis graph can be found in Figure 4.23.

In Figure 4.23 we can see that we have a much clearer graph with one very distinct peak around 100MHz. Because these are not power measurements that may be impacted by the lingering power fluctuations across the device, we are more confident that the peak in this graph signifies the DES implementation on the chip. It is important

to remember the electromagnetic side channel carries information about emanations in the device when current flows and is based on changing logic states during device execution.

The traces from site 89 collected from Variant DES_20 and Variant DES_50 can be found in Figure 4.24 and Figure 4.25 respectively. In these figures, the top graph is the raw trace for site 89, the middle graph is an intermediate filtering stage as in Figure 4.22, and the bottom graph is spectral analysis performed on the filtered graph.

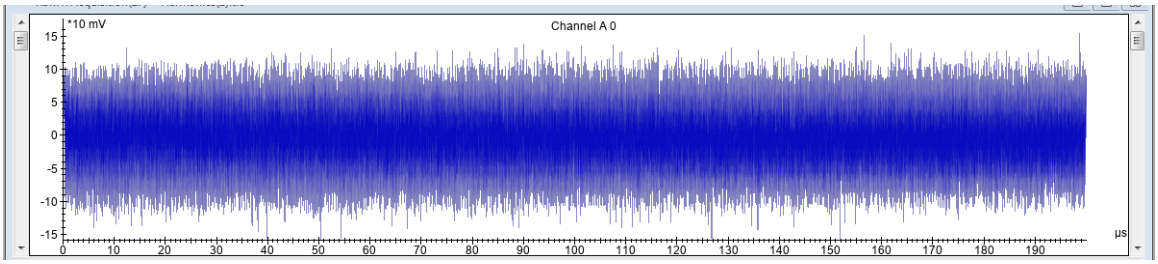

Figure 4.22 Emag: Native DES – Harmonics.

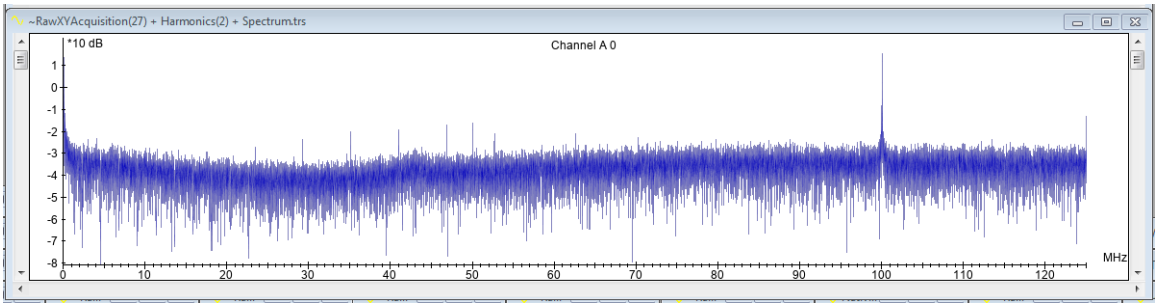

Figure 4.23 Emag: Native DES – Spectrum.

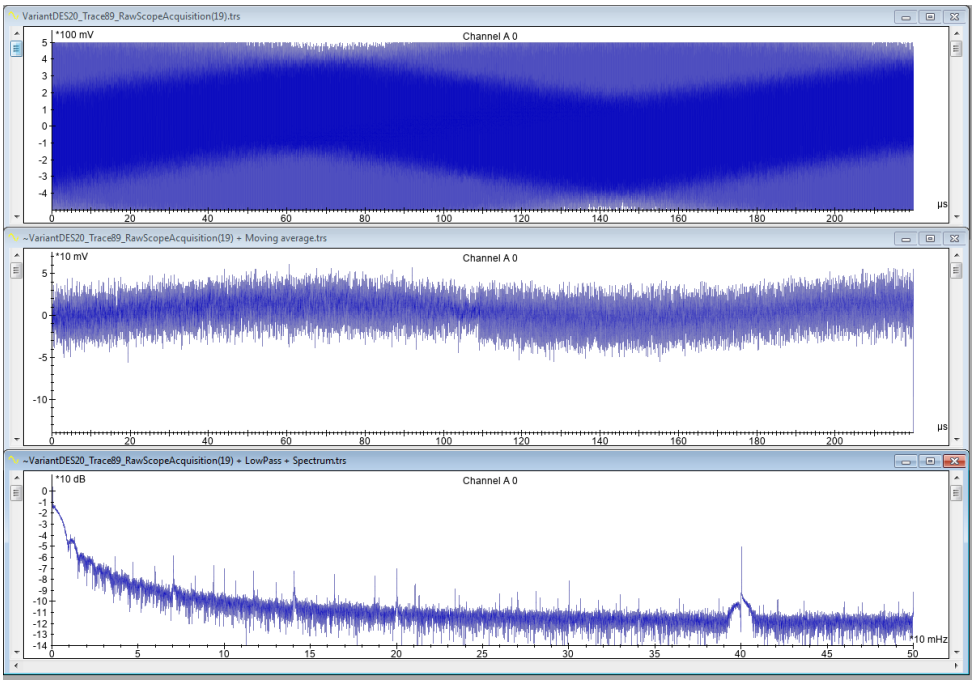

Figure 4.24 Emag: Variant DES_20.

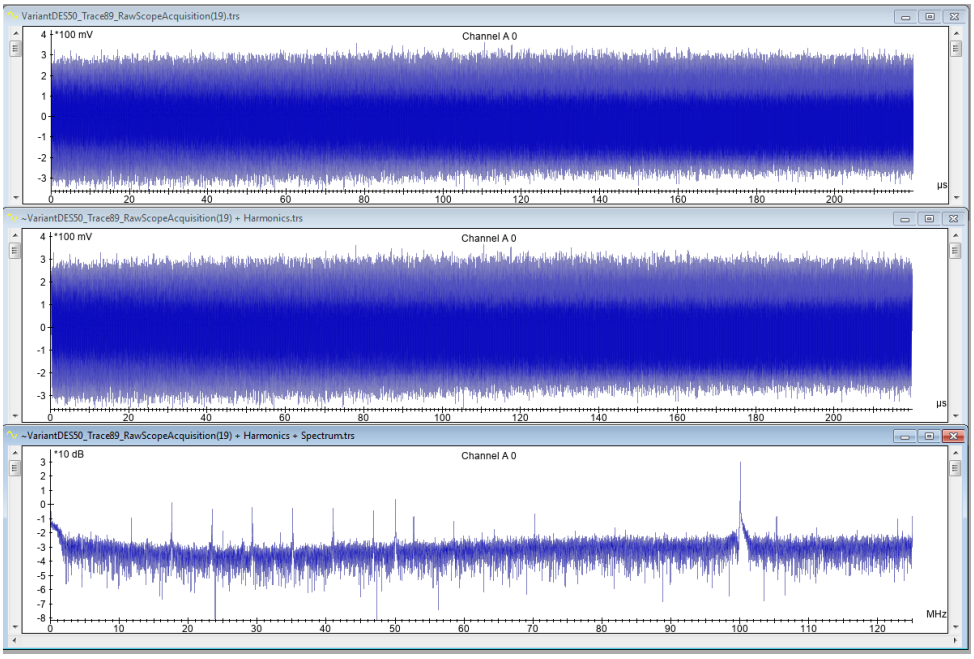

Figure 4.24 Emag: Variant DES_50.

In analyzing these traces and their filtered and analyzed versions, like Native DES they too have a large peak around 100MHz in the spectrum analysis graph that we believe is the encryption. Although these traces are very similar to each other in this respect, unlike the power traces, these traces (both raw and filtered/analyzed) when compared one to another do have minor differences mainly in the ranges of the signals and the amplitudes of the peaks present in the graphs. When analyzed by a competent adversary, we believe these differences would be apparent and would compel one to believe that each graph is representative of a functionally different circuit.

CHAPTER V

CONCLUSIONS

This chapter provides an overview of the results found in the experimentation process. Section 5.1 provides a summary of our results while Section 5.2 offers an outline of the future work proposed if this research were to be continued.

5.1 Summary of Findings

In a world where technological advancements rise and fall and at an alarmingly fast rate, system security and integrity are ever-growing fields. Because of these advancements, reverse engineering techniques such as side channel analysis are also advancing and growing into powerful threats to the security community. With dangers like these on the horizon, protection methods such as moving target defense are becoming a major research focus. This technique at its most basic level seeks to lessen the attack surface available to an adversary in a variety of ways. This research, however, utilizes circuit variance as a method of moving target defense to not only lessen the attack surface but also to make it as incomprehensible as possible.

In this research, our main goal was to determine if side channel characterizations of functionally equivalent native and variant circuits would manifest themselves differently enough for the circuits to be distinguished as three separate circuits. To

accomplish this goal, native and variant implementations of DES were ascertained and programmed onto an FPGA. The power and electromagnetic side channels of each circuit were then collected and analyzed for any indication that the circuits, although functionally equivalent, were three distinct implementations.

In concluding our experimentation, we found that while the goal was to collect both power and electromagnetic side channels, the power side channels collected were not effective. The power traces we gathered, though informative, did not yield conclusive results as the traces were too noisy. This noise, we believe, is due to a combination of the almost imperceptible changes in current flows throughout the device that may not be detected by the analysis software, and the ARM processor running and creating excess noise in the traces that unfortunately could not be filtered out. We did find, however, that our electromagnetic traces did yield results. In these traces we saw that both our raw and filtered traces there were differences in the native and side channel characterizations. Although small, these differences could aid in lessening the attack surface for these circuits as the analysis of three distinct circuits would require no less than three distinct approaches for reverse engineering. Based on these findings, we can conclude that side channel characterizations of functionally equivalent circuit variants can be used to distinguish native from variant. Though more analyses should be completed, and more traces taken, our preliminary findings allow us to state that the presence of circuit variance does have an impact on side channel characterizations. We believe that these findings do support the further analysis on the impact of circuit variation on side channel representations.

5.2 Future Work

There are several areas of this research that have been defined as vectors for continued study. The first of these is the DES implementation utilized in this research. This implementation as we stated before could not be verified against other traditional implementations of DES because of a byte ordering issue within the logic. Future work on this vector would include rectifying this byte ordering issue and verifying the DES implementation against traditional DES. Although the functionality of the circuits used in this research are not what is tested, ensuring the desired circuit executes correctly would add to the confidence of any results and conclusions reached.

In addition are the power traces collected in this research. Although these traces did not yield the results expected, it would be beneficial to take more power measurements. There are two approaches that we see could be taken for these measurements. 1: The power traces could be taken again, and more filtering and cleanup could be done on the individual traces to pinpoint areas for comparison; 2. New traces could be taken on the same chip with the elimination of the excess noise (e.g., similar to a pinata implementation of the ARM processor). Either of these approaches would aid in giving concrete results of the power measurements of the executing device.

And lastly, in future work are the electromagnetic traces taken for this research. In our approach, we only collected traces for 3-4 sites on the chip whereas the chip has 400 sites designated in the spectral intensity scan. Future work in this vector would include taking traces of considerably more sites on the chip (at least 20%). While this was not a timely option in our approach, the addition of more traces would strengthen the results and make the subsequent conclusions more reliable.

REFERENCES

[1] "The Equifax Data Breach: What to Do," *Consumer Information*, Sep. 08, 2017. https://www.consumer.ftc.gov/blog/2017/09/equifax-data-breach-what-do (accessed Oct. 31, 2017).

[2] J. Dombrowski, T. R. Andel, and J. T. McDonald, "The Application of Moving Target Defense to Field Programmable Gate Arrays," in *Proceedings of the 11th Annual Cyber and Information Security Research Conference*, New York, NY, USA, 2016, p. 20:1-20:4. doi: 10.1145/2897795.2897820.

[3] R. Zhuang, S. A. DeLoach, and X. Ou, "Towards a Theory of Moving Target Defense," in *Proceedings of the First ACM Workshop on Moving Target Defense*, New York, NY, USA, 2014, pp. 31–40. doi: 10.1145/2663474.2663479.

[4] S. Jajodia, A. K. Ghosh, V. Swarup, C. Wang, and X. S. Wang, *Moving Target Defense: Creating Asymmetric Uncertainty for Cyber Threats*. Springer Science & Business Media, 2011.

[5] M. Carvalho *et al.*, "Command and Control Requirements for Moving-Target Defense," *IEEE Intell. Syst.*, vol. 27, no. 3, pp. 79–85, May 2012, doi: 10.1109/MIS.2012.45.

[6] S. Garg, C. Gentry, S. Halevi, M. Raykova, A. Sahai, and B. Waters, "Candidate Indistinguishability Obfuscation and Functional Encryption for All Circuits," *SIAM J. Comput.*, vol. 45, no. 3, pp. 882–929, Jan. 2016, doi: 10.1137/14095772X.

[7] J. T. McDonald, Y. Kim, and D. Koranek, "Deterministic Circuit Variation for Anti-tamper Applications," in *Proceedings of the Seventh Annual Workshop on Cyber Security and Information Intelligence Research*, New York, NY, USA, 2011, p. 68:1-68:1. doi: 10.1145/2179298.2179376.

[8] C. S. Collberg and C. Thomborson, "Watermarking, tamper-proofing, and obfuscation - tools for software protection," *IEEE Trans. Softw. Eng.*, vol. 28, no. 8, pp. 735–746, Aug. 2002, doi: 10.1109/TSE.2002.1027797.

[9] T. R. Andel, L. N. Whitehurst, and J. T. McDonald, "Software Security and Randomization Through Program Partitioning and Circuit Variation," in *Proceedings of the First ACM Workshop on Moving Target Defense*, New York, NY, USA, 2014, pp. 79–86. doi: 10.1145/2663474.2663484.

[10] I. Kuon and J. Rose, “Measuring the Gap Between FPGAs and ASICs,” *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 26, no. 2, pp. 203–215, Feb. 2007, doi: 10.1109/TCAD.2006.884574.
[11] P. Marchal, “Field-programmable Gate Arrays,” *Commun ACM*, vol. 42, no. 4, pp. 57–59, Apr. 1999, doi: 10.1145/299157.299594.
[12] L. N. Whitehurst, “Enhanced software security through program partitioning,” 2015.
[13] C. Bolchini, A. Miele, and M. D. Santambrogio, “TMR and Partial Dynamic Reconfiguration to mitigate SEU faults in FPGAs,” in *22nd IEEE International Symposium on Defect and Fault-Tolerance in VLSI Systems (DFT 2007)*, Sep. 2007, pp. 87–95. doi: 10.1109/DFT.2007.25.
[14] E. J. McDonald, “Runtime FPGA Partial Reconfiguration,” in *2008 IEEE Aerospace Conference*, Mar. 2008, pp. 1–7. doi: 10.1109/AERO.2008.4526368.
[15] G. Stitt, R. Lysecky, and F. Vahid, “Dynamic Hardware/Software Partitioning: A First Approach,” in *Proceedings of the 40th Annual Design Automation Conference*, New York, NY, USA, 2003, pp. 250–255. doi: 10.1145/775832.775896.
[16] R. Frankland, D. Demirel, J. Budurushi, and M. Volkamer, “Side-channels and eVoting machine security: Identifying vulnerabilities and defining requirements,” in *2011 International Workshop on Requirements Engineering for Electronic Voting Systems*, Aug. 2011, pp. 37–46. doi: 10.1109/REVOTE.2011.6045914.
[17] C. Clavier, J.-S. Coron, and N. Dabbous, “Differential Power Analysis in the Presence of Hardware Countermeasures,” in *Cryptographic Hardware and Embedded Systems — CHES 2000*, vol. 1965, Ç. K. Koç and C. Paar, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 252–263. doi: 10.1007/3-540-44499-8_20.
[18] H. Gamaarachchi and H. Ganegoda, “Power Analysis Based Side Channel Attack,” *ArXiv180100932 Cs*, Jan. 2018, Accessed: Mar. 13, 2022. [Online]. Available: http://arxiv.org/abs/1801.00932
[19] P. Kocher, J. Jaffe, and B. Jun, “Differential Power Analysis,” in *Advances in Cryptology — CRYPTO’ 99*, Berlin, Heidelberg, 1999, pp. 388–397. doi: 10.1007/3-540-48405-1_25.
[20] D. Agrawal, B. Archambeault, J. R. Rao, and P. Rohatgi, “The EM Side—Channel(s),” in *Cryptographic Hardware and Embedded Systems - CHES 2002*, Berlin, Heidelberg, 2003, pp. 29–45. doi: 10.1007/3-540-36400-5_4.
[21] M. E. Smid and D. K. Branstad, “Data Encryption Standard: past and future,” *Proc. IEEE*, vol. 76, no. 5, pp. 550–559, May 1988, doi: 10.1109/5.4441.
[22] R. Morris, “The data encryption standard–Retrospective and prospects,” *IEEE Commun. Soc. Mag.*, vol. 16, no. 6, pp. 11–14, Nov. 1978, doi: 10.1109/MCOM.1978.1089783.

[23] “Data encryption standard (DES) | Set 1,” *GeeksforGeeks*, Aug. 17, 2018. https://www.geeksforgeeks.org/data-encryption-standard-des-set-1/ (accessed Feb. 22, 2021).
[24] S. Oukili and S. Bri, “FPGA implementation of Data Encryption Standard using time variable permutations,” in *2015 27th International Conference on Microelectronics (ICM)*, Dec. 2015, pp. 126–129. doi: 10.1109/ICM.2015.7438004.
[25] R. Davis, “The data encryption standard in perspective,” *IEEE Commun. Soc. Mag.*, vol. 16, no. 6, pp. 5–9, Nov. 1978, doi: 10.1109/MCOM.1978.1089771.
[26] “Fig. 5. Data Encryption Standard (DES) Algorithm.,” *ResearchGate*. https://www.researchgate.net/figure/Data-Encryption-Standard-DES-Algorithm_fig4_321494910 (accessed Feb. 22, 2021).
[27] B. Schneier, “Algorithm Types and Modes,” in *Applied Cryptography, Second Edition*, John Wiley & Sons, Ltd, 2015, pp. 189–211. doi: 10.1002/9781119183471.ch9.
[28] M. Madou, B. Anckaert, P. Moseley, S. Debray, B. D. Sutter, and K. D. Bosschere, “Software Protection Through Dynamic Code Mutation,” in *Information Security Applications*, Aug. 2005, pp. 194–206. doi: 10.1007/11604938_15.
[29] M. A. Forbes, “Digital Logic Protection Using Functional Polymorphism and Topology Hiding,” M.S., University of South Alabama, United States -- Alabama, 2017. Accessed: May 25, 2020. [Online]. Available: http://search.proquest.com/docview/1891353067/abstract/FFA3A7079EB0481EPQ/1
[30] J. McDonald T., “Program Encryption Toolkit User Guide | Release 1.0,” University of South Alabama.
[31] “ZedBoard | Zedboard.” http://zedboard.org/product/zedboard (accessed Mar. 26, 2020).
[32] “Vivado Design Suite.” https://www.xilinx.com/products/design-tools/vivado.html (accessed Mar. 26, 2020).
[33] “Xilinx Software Development Kit (XSDK).” https://www.xilinx.com/products/design-tools/embedded-software/sdk.html (accessed Mar. 26, 2020).
[34] “Slices on an FPGA Chip.” https://www.ni.com/en-us/support/documentation/supplemental/18/slices-on-an-fpga-chip.html (accessed Mar. 08, 2022).
[35] “Spectral graphs.” https://jdemetradocumentation.github.io/JDemetra-documentation/pages/case-studies/spectralgraphs.html (accessed Mar. 13, 2022).

APPENDICES

Appendix A: Vivado SDK HelloWorld.c (Encryption)

```
#include <stdio.h>
#include "platform.h"
#include "xil_printf.h"
#include "xil_types.h"
#include "xparameters.h"
#include "xil_io.h"
#include "sleep.h"
#include "xgpio.h"

static XGpio Trigger;

int main() {
	init_platform();
	int gpio_status;

	uint32_t key_upper = 0xfeedfeed;
	uint32_t key_lower = 0xfeedfeed;
	uint32_t plaintext_upper = 0xdeadbeef;
	uint32_t plaintext_lower = 0xdeadbeef;

	xil_printf("Running Native DES: \r\n");

	 /*************************
	    * GPIO initialization      *
	    *************************/
	   gpio_status = XGpio_Initialize(&Trigger, XPAR_AXI_GPIO_0_DEVICE_ID);
	        if (gpio_status != XST_SUCCESS) {
	           printf("GPIO Initialization Failed\r\n");
	           return XST_FAILURE;
	        }

	XGpio_SetDataDirection(&Trigger, 1, 0); //Set Trigger as output

	xil_printf("Plaintext: %x%x\r\n", plaintext_upper, plaintext_lower);


	while(1) //Encryption loop
	{
```

```
                usleep(10000);
                usleep(10000);
                usleep(10000);
                usleep(10000);
                usleep(10000);
                usleep(10000);

                XGpio_DiscreteWrite(&Trigger, 1, 1); //set trigger high

                // encrypt
                Xil_Out32(XPAR_DES_IP_0_S00_AXI_BASEADDR, key_upper);
                Xil_Out32(XPAR_DES_IP_0_S00_AXI_BASEADDR + 4, key_lower);

                Xil_Out32(XPAR_DES_IP_0_S00_AXI_BASEADDR + 8, plaintext_upper);
                Xil_Out32(XPAR_DES_IP_0_S00_AXI_BASEADDR + 12, plaintext_lower);

                usleep(100);

                uint32_t ciphertext_upper = Xil_In32(XPAR_DES_IP_0_S00_AXI_BASEADDR + 16);
                uint32_t ciphertext_lower = Xil_In32(XPAR_DES_IP_0_S00_AXI_BASEADDR + 20);

                xil_printf("Ciphertext: %x%x\r\n", ciphertext_upper, ciphertext_lower);
        }

        XGpio_DiscreteWrite(&Trigger, 1, 0); //trigger low

        cleanup_platform();
        return 0;
}
```

Appendix B: “Dont_Touch” Script

```
import re

#DES20 files
DES20path = "\sydne\OneDrive\Documents\Research\CircuitsinVivado\DES20\DES1.vhdl"
DES20 = open(DES20path, 'r')

DES20editspath = "\sydne\OneDrive\Documents\Research\CircuitsinVivado\DES20\DES20edits.txt"
DES20edits = open (DES20editspath, 'w')

#DES50 files
DES50path = "\sydne\OneDrive\Documents\Research\CircuitsinVivado\DES50\DES50.vhdl"
DES50 = open(DES50path, 'r')

DES50editspath = "\sydne\OneDrive\Documents\Research\CircuitsinVivado\DES50\DES50edits.txt"
DES50edits = open (DES50editspath, 'w')


#DES20 dont_touch
for line in DES20:
    match = re.search(r'wire[0-9]+' , line)
    string = "  attribute dont_touch of " + match.group() + " : signal is \"true\";"

    DES20edits.write("\n")
    DES20edits.write(string)

#DES50 dont_touch
for line in DES50:
    match = re.search(r'wire[0-9]+' , line)
    string = "  attribute dont_touch of " + match.group() + " : signal is \"true\";"

    DES50edits.write("\n")
    DES50edits.write(string)


#flush files
DES20edits.flush()
DES50edits.flush()

 #close files
DES20edits.close()
DES50edits.close()

DES20.close()
DES50.close()
```

BIOGRAPHICAL SKETCH

Name of Author: Sydney L. Davis

Graduate and Undergraduate Schools Attended:
University of South Alabama, Mobile, Alabama.

Degrees Awarded:
B.S. in Computer Engineering, May 2020.
M.S. in Computer and Information Sciences, May 2022.