On studying Whitenoise Stream-Cipher against Power Analysis Attacks

by

Babak Zakeri
B.Sc., University of Tehran, Iran, 2005

A Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of

MASTER OF APPLIED SCIENCE

in the Department of Electrical and Computer Engineering

© Babak Zakeri, 2012
University of Victoria

All rights reserved. This thesis may not be reproduced in whole or in part, by photocopying or other means, without the permission of the author.

On studying Whitenoise Stream-Cipher against Power Analysis Attacks

by

Babak Zakeri
B.Sc., University of Tehran, Iran, 2005

Supervisory Committee

Dr. Mihai Sima, Supervisor
(Department of Electrical and Computer Engineering)

Dr. Michael McGuire, Departmental Member
(Department of Electrical and Computer Engineering)

Supervisory Committee

Dr. Mihai Sima, Supervisor
(Department of Electrical and Computer Engineering)

Dr. Michael McGuire, Departmental Member
(Department of Electrical and Computer Engineering)

ABSTRACT

This report describes the works done since May 2010 to December 2012 on breaking Whitenoise encryption algorithm. It is mainly divided into two sections: Studying the stream-cipher developed by Whitenoise lab and its implementation on a FPGA against certain group of indirect attacks called Power Analysis Attacks, and reviewing the process of development and results of experiments applied on a power sampling board which was developed during this project. For the first part the algorithm and the implementation would be reverse engineered and reviewed. Various blocks of the implementation would be studied one by one against some indirect attacks. It would be shown that those attacks are useless or at least very weak against Whitenoise. A new scenario would then be proposed to attack the implementation. An improvement to the new scenario would also be presented to completely hack the implementation. However it would also be shown that the complete hack requires very accurate equipment, large number of computations and applying a lot of tests and thus Whitenoise seems fairly strong against this specific group of attacks. In the next section the requirements of a power consumption measurement setup would be discussed. Then the motivations and goals of building such a board would be mentioned. Some important concepts and consideration in building the board, such as schematic of the amplifier, multilayer designing, embedding a BGA component, star grounding, inductance reduction, and other concepts would be presented. Then the results of applied tests on the produced board would be discussed. The precision of the measurements, some pattern recognition along with some other results would be illustrated. Also

some important characteristics such as linearity of measurements would be investigated and proved to exist. In the end some topics as possible future works, such as more pattern recognition, or observing the effect of masks on the power consumption would be suggested.

Contents

List of Figures

ACKNOWLEDGEMENTS

I would like to thank:

Dr Mihai Sima, for closely helping and guiding me in this project and many other matters of my degree. It was a pleasure working with a knowledgeable and accurate person as him.

I would also like to thank:

ReCoEng Lab, and Whitenoise Lab, for supporting this project.

My thanks and regards to the defence committee for spending time reviewing this thesis and attending the examination. I'd also like to thank the University of Victoria and specially ECE department and all the professors, technicians and staff for providing the opportunity of studying and doing research here: To Steve Campbell, Kevin Jones and Erik Laxdal, the professional staff, for providing the necessary software and solving occasional problems. To Paul Fedrigo, Rob Fichtner and Brent Sirna, the technical staff, for their help during the process of preparing the board by providing material, occasional suggestions on designing the peripheral blocks and helping in assembling some parts. And to Lynne Barrett, Moneca Bracken and Janice Closson, the office staff, for processing all the formal works related to my defence and other matters of my degree.

DEDICATION

To my parents, who have devoted much of their life to me.

Chapter 1

Introduction

Since mid 90s and due to works such as those of P. Kocher[13, 14] it became obvious that cryptographic algorithms no matter how strong they are in theory need to be examined in implementation against other forms of attacks. Before this work cryptographic algorithms were only examined against *direct* attacks, the algebraic or computational attacks applied on the algorithm itself. But these new so called *indirect* attacks use external information, such as power consumption or electromagnetic emissions, obtained from the characteristics of the implementation rather than the algorithm itself to find the secret parts of the algorithm. Since then there have been many efforts both in attacking the implementations and finding countermeasures for them using different methods and the topic is still one of the fresh and open discussions in cryptography.

In the challenge of finding a strong algorithm which can't be broken Whitenoise Lab[12] has proposed a new stream-cipher which is patented[6]. While the concept behind the algorithm is straightforward and it can be implemented using HDL languages at high level, it is claimed in some previous works that it is resistant to direct attacks[29, 28]. Whitenoise has also provided a VHDL based implementation of the algorithm which was designed for FPGAs[1]. However no experiment, before this project, was performed on it to study its strength against indirect attacks. Hence one goal of the project was defined to examine Whitenoise implementation and comment on its strength or weaknesses against some common indirect attacks, namely the more common forms of Power Analysis Attacks.

In a parallel study a measurement setup then was desired to be built to have

[1]Field Programmable Gate Arrays

an experience of its design process and to obtain some real experimental results. Investigating the requirements of such a design, doing the actual design process, applying tests and observing the accuracy of measurements along with finding other characteristics of the setup were all defined to be the second part of this project then.

Thus the two main goals of this project can be summarized as: Examining the strength of the Whitenoise against some known attacks, and building a measurement setup and applying different tests on it. It should be noted that although the report studies Whitenoise for some of the most well known attacks, notes the strong or weak points about the implementation, suggests a new attack scenario for breaking the Whitenoise and gives notes about improving it, it shouldn't be considered as a complete inquiry of Whitenoise against all indirect attacks. Also while the design process of measurement board and some interesting results would be presented and discussed, the setup should not be seen as an end-use tool for hacking Whitenoise or any other algorithm.

1.1 Claims and Agenda

There have been many types of indirect attacks introduced in the past two decades. Some of them such as SCA[2] attack the implementation using leakage information such as power consumption, electromagnetic emanations or timing characteristics, while others such as *Fault Tolerant* attacks do this by inducing faults and alterations to the digital circuit. Among indirect attacks, SCAs have shown to be most effective and among SCAs, $PAAs$[3] are more common since they are stronger than timing attacks and require less analysis comparing to EMAs[4].

Thus for both goals of this project, power consumption is chosen as the extra source of information. In studying Whitenoise, first the common methods are discussed against it and it would be shown that Whitenoise seems resistant to them. Then an attack scenario, which is again based on studying samples of power, is proposed. The proposed attack itself is capable of reducing the search space and with an improvement to it, which also is presented, the implementation is easily breakable in a polynomial time. However since the proposed attack is applicable only on the provided implementation, it doesn't question the strength of the algorithm and the

[2]Side Channel Attacks
[3]Power Analysis Attacks
[4]Electromagnetic Attacks

designer might be able to improve the implementation to counter this attack.

About the setup and the board, one way to sample the power is to deploy a resistor between the ground terminal of the device and global ground, and measuring the voltage drop across the resistor. However this resistor reduces the noise immunity of the designed circuitry. Thus, small values of resistors are needed to maintain the voltage drop within the acceptable limits of the device($0.2v$ for the device used in this project with the voltage supply of $1.2V$). So there is also need of amplification since the alterations of voltage on such a small resistor(less than 200Ω) would be small. These together with all the other considerations for embedding the digital section requires a multilayer board. Such a board requires many components such as coupling capacitors and ferrite beads for stabilizing the power and many concepts such as star grounding, and inductance reduction to be considered in it. These are discussed in the related chapter. The results of running tests on the fabricated board would then be presented and it would be shown that the board and the device are functioning well. The precision in measuring the power, along with some other characteristics such as linearity of measurements are all presented and discussed.

1.2 Outline

The rest of this report would go through the details of the above topics. It is organized as follows:

Chapter 2 contains some background knowledge about cryptography, power modelling and SCAs which are necessary for understanding the rest of the report.

Chapter 3 describes the Whitenoise algorithm and implementation. It then examines the implementation against known PAA attacks. It then provides an attack scenario for that implementation. And at the end the requirements, efficiency and possible improvements to the attack would be discussed.

Chapter 4 goes through the process of designing the board. It discusses some main considerations and issues in designing such a setup board. Then the results and the tests would be presented and concluded.

Chapter 5 is a restatement of the claims and results of the thesis in more detail. It also talks about the possible future works that can be done.

Chapter 2

Background

In this chapter some of the background topics necessary for the main discussions later in chapter 3 and 4 are studied. These include basic concepts of cryptography, what attacking an algorithm or implementation means, reviewing the power consumption modelling in digital devices, and some of the most common attacks, most of which based on PAAs. Some other attacks and methods are also discussed at the end, to introduce the readers with other available techniques of indirect attacks.

2.1 Basics of Cryptography

Encryption has been a way to pass and store the data in a secure format since a long time ago. Its origins goes back to ancient Roman empire and even before that. A famous example of an encryption application in the past century might be the Enigma machine which was used in WW II by Germans to code and decode the secret messages. Nowadays encryption has found a whole new area of applications in digital devices and networks. Smart cards, data streaming in networks and secure storage of information on a station are examples of such applications. The idea in developing a cryptographic algorithm is to provide a method to combine the input (plain-text) with the secret part (key) to obtain the output (cipher-text). As commonly agreed within the cryptography community the cryptographic algorithm should have the following characteristics:

1. The cipher-text is obtainable in a reverse process of decryption by having the key, whether it is the same key or some other relevant hash information.

2. The algorithm specification except the key should be open source, and known to everyone.

3. The algorithm should be reliable hard to break with known methods.

Based on these ideas, many encryption algorithms have been developed. However, the building blocks and operations which are used in most of them are still almost the same. There are algorithm specific operations too, but most often any other complex operation can be implemented using few of these operations:

1. Right or left shifting of bits, bytes, or words of the plain-text, or a combination of plain-text and key.

2. Swapping or reordeing bits, bytes, or words of plaintext, or a combination of plain-text and key.

3. Applying addition (XOR) to all or parts of the plain-text and key.

4. Substituting bits, bytes, or words of plaintext, or a combation of plain-text and key, with a new value based on a LUT[1].

5. Merging and combining some of these operations together and doing them in consecetive rounds.

Note that all of these operations except the substitution operation are linear operations, meaning that $F(A+B) = F(A)+F(B)$. Substitution block or box, shortly known as S-Box, is normally the block responsible for creating non-linear output. If S-Box is removed from the algorithm, then the encryption is actually impossible; since by having a pair of input and output, output for all the other inputs would be easily computable, so the non-linear section plays a very important role in the algorithm.

Cryptographic algorithms in general can be categorized in different ways. But one particular categorization which is of interest for this thesis is *block-ciphers* vs. *stream-ciphers*. In block-cipher algorithms, the key is set and fixed during a whole run of encryption. Key is not meant to change in applications related to this category of algorithms except on special occasions. For example there might be a monthly or yearly plan of changing the key for increasing the security or the key might be changed because of recognized break to the encryption device. But most of the times

[1]Look Up Table

key is fixed and this means the process of encryption should be complicated instead so hacking the information is hard. Normally in these cases the encryption algorithm includes a long key (64, 128, 256 bits or longer) of a length similar to that of the plain-text. So if the attacker wants to obtain the secret information by a brute force method, that is trying all the possible cases, it would be nearly impossible to do it even with a very fast system.

For example if an encryption algorithm is using 64-bit words key, there would be 2^{64} cases to be checked which is a number in the order of 10^{20}. Testing each case means setting the key to that value, apply input to the algorithm, calculate the output and compare it with the output of the DUA[2]. Thus testing each case takes at least the time for a complete encryption process. If testing each case would take an average of 1000 instructions for example, 10^{23} instructions are needed for all possible cases. The clock frequency of a processor of a common PC would go as high as a few GHz. Assuming there is a super computer which is sharing $1,000,000$ of such systems in the most optimum and parallel way, and assuming instruction at assembly level only takes 1 clock cycle, such a super computer would be able to perform $1,000,000 * 10^9$ instructions in a second. And again if assumed that the code is optimally implemented so all the instructions can be executed in parallel, it takes 10^8 seconds to complete the task, which in short means 317 years to finish it up! Note that many unrealistic and optimistic assumptions have been made here and yet such a result is obtained.

For this reason while the algorithm is built strong enough so the attacker can not obtain the secret information by some other method, having a constant key is not an issue in block-ciphers. Block-ciphers are very efficient in applications where access to the encryption device might be possible. Such an example would be a smart card which can be lost or stolen. However the trade-off of having such ciphers is that long words of data should be encrypted/decrypted before plain-text or cipher-text is available. Also once the key is hacked, the device can be used for all of the previous and later streams of data until the key is changed. This second weakness in some situations is not affordable. In a military environment where the encrypted information can be of high value and there might be efforts to break the cipher-text, it is more valuable to have ciphers which are not dependant on a single key so even if the key is hacked the information can be decrypted for just a short amount of cipher-text. This is an example of one of the cases where stream-ciphers are a better option.

[2]Device Under Attack

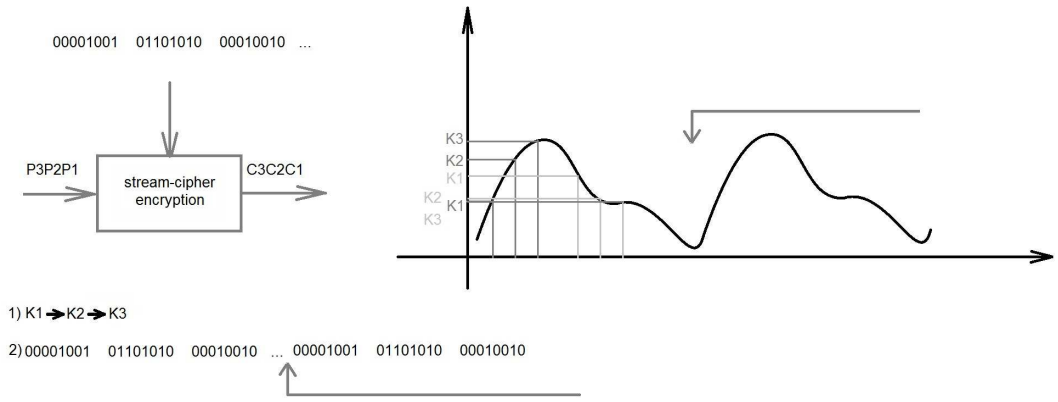


Figure 2.1: Repeating period and correlation of keys in stream-cipher

Stream-ciphers are meant for fast encryption of long streams of short width data. An example of their applications as said before, would be secure wireless network or storage of large amounts of data in a single station. An assumption in these class of applications is that the device is not directly accessible to the attacker, or the access is very limited. Attacker might have access to short amount of samples and some leakage data instead. A stream-cipher algorithm is mostly made of a simple operation (such as addition) between bytes, or a few bytes, of plain-text and the key. The algorithm is very simple, however the key is changing for each word of plain-text and in fact designing the algorithm means designing the key generation part.

When a stream-cipher is used, the sender and receiver are normally synchronised with each other using some sort of counter to ensure the same key is being used for both sides of encryption and decryption process. The strength of a stream-cipher is based on how well the keys are built and randomized. The following are used as the main criteria for rating the level of randomization:

1. Correlation of the generated keys in different times: How two sets of generated keys in two different times correlate with each other.

2. Period: After how many number of generated keys the pattern is repeated.

Figure 2.1 illustrates these two concepts. For this example the figure shows how keys are changed differently in different times of encryption process. Two sets of keys (K1, K2 and K3 in dark and light Gray) are indicated in the time plot. K1 for the two sets occurs at different times, but the time gap between occurrence of K1, K2 and K3

are the same for the two sets. It is obvious from the figure that the changes of the key from K1 to K3 is different for the two sets, which means a low correlation between the generated keys regarding the time. The repeating period of keys is also shown. Ideally, for a stream-cipher the repeating period should be infinite, and the correlation between the generated keys at any time for a set of keys should be as low as possible. Irrational numbers such as square root of a positive integer are examples of sources of number generation patterns with the infinite repeating period. For examining the correlation of generated keys, statistical models such as covariance exist to explain this criteria in more mathematical language.

2.2 Breaking an Algorithm/Implementation

In this section the concept of breaking a system of cryptography at algorithm and implementation level would be discussed. By breaking an encryption one might imply different things. The algorithm might be the only available target and attacker may want to examine mathematical approaches, namely the direct attacks. Or maybe the actual DUA is available to the attacker and she wants to find the its key. The level of detail that attacker has access to can be different in this case. She might only have access to a behavioural description of the implementation, or she might also have access to a gate level netlist. She might even the layout of the design with all the related information including the mapping and place and route schematics. In another case, the DUA might not be accessible to the attacker and she might only have access to some set of encrypted information (cipher-text) along with the leakage corresponding to them.

Another form of attack is where access to DUA is limited, but a rather experimental device is provided to him. This alternative device has exactly the same characteristics of the DUA except secret key and key is in fact programmable. The attacker can perform as many tests as she wants and study the design in detail to deduce patterns from the leakage, which then later can be used to attack the DUA. Such device is usually called an *IED*[3]. Attacks based on this approach would be discussed in more detail later, when *Template Attacks* are being reviewed.

From another point of view, the amount of samples available is another limitation for the attacker. In one scenario, an attacker might be able to apply as many tests as

[3]Identical Experimental Device

she wants on DUA monitoring the input and output. However in some other scenario only limited samples might be available. The attacker might only have access to the output and leakage information of a few samples of DUA, so any further pattern recognition should be done on IED. Most of the SCAs are based on the hypothesis that the number of samples are limited, or they observe the accuracy of the *guessed key* based on the number of samples available. The emphasis of this discussion is to note that there are many parameters involved in the process of choosing a certain type of attack, and attacks differ not only in the type of the leakage, but also in the way the above parameters are chosen.

Now lets start from the algorithm itself. Typically, the encryption algorithm is known to the attacker. Many standards today provide the algorithm along with some code as an open source for designers. As said, the assumption is that algorithm is strong enough to resist attacks, and besides some rare custom algorithms for specific and limited applications, the algorithm is disclosed. There are many points to consider in developing an algorithm. For example, if the algorithm is a block-cipher, the key should not be easily obtainable from the plain-text and cipher-text. This means that, although the function $F(K, P) = C$ is invertible regarding P so the decryption can be done ($P = G(K, C)$), but the function $H(C, P) = K$ is not derivable. It either does not exist, or there should be no direct mathematical way to obtain it. In fact, this is the reason why the attacker tries to find the key with different method. Otherwise by having a single set of plain-text and cipher-text key would be revealed.

It was also mentioned that the keys in a stream-cipher and cipher-texts in a block-cipher should not correlate. High correlation between the generated keys makes it possible for the attacker to find the next keys based on some previously generated ones. But besides low correlation regarding time, there is one other important factor too, that should be considered. The variation of distribution of keys (cipher-text in block-ciphers) should also be high. For example if in a block-cipher, for close values of key the cipher-texts are also close, then the attacker can easily break the encryption. She applies some tests, finds the range of the correct key, and checks are the possibilities in that range.

As one other requirement, the different bits and bytes of the cipher-text (the final key in the case of stream-cipher) should not be totally independent either. Imagine a case where for a 1024-bit plain-text and key, each ith bit of cipher-text is only produced using the ith bit of plain-text and key. In such a case the length of the plain-text and key actually don't matter and with applying a single test on DUA

the value of the key would be revealed. If the ith bit of the computed and expected cipher-text is equal to the ith bit of the result of applying the test on DUA, then the hypothesis for the ith bit of the key was right. If it doesn't match it means that the inverse of that bit is the right asnwer. So to find the key, attacker does not need to have the whole expected cipher-text and cipher-text generated of DUA to be equal and one single test with an assumption on key would reveal the right value.

For the above reasons there is normally a non-linear part in the algorithm. The above requirements, which are low correlation, high variance and bit-wise dependency can all be met using a non-linear function. The S-Box introduced in the first section of this chapter is the part of the algorithm where this non-linear operation is performed. Field Arithmetic, such as field multiplication or inverse, is one category of the options available for generating such blocks, while there are many other methods for implementing S-Boxes for encryption algorithms.

At the end, the algorithm is tested by standardization organizations using some known mathematical measures to observe its strength from different points of view. AES[4][20] is an example of such algorithms which was elected in 2001 by NIST[5] among some other competitors as the successor of the old DES[6].

In 2.4 some of the well known attacking scenarios specially built based on power consumption as the leakage would be discussed. But first it needs to be clear that how the power consumption is modelled in a digital device. That is discussed in the next section.

2.3 Power Consumption Modelling of Digital Devices

From the attacker's point of view, the power consumption of a digital device can be divided into a sym of three terms, as follows:

$$P(total) = P(el.noise) + P(sw.noise) + P(desired) \tag{2.1}$$

$P(el.noise)$ (the power consumption of electrical noise) corresponds to the portion of power which is always present independently from the input or the state of

[4]Advanced Encryption Standard
[5]National Institute of Standardization and Technology of United States
[6]Data Encryption Standard

the circuit. This power consumption might be caused by the environment around the workstation, or other components in the DUA that are not directly part of the implementation. Some oscillator, regulator and similar components might introduce noise of their own to the circuit even when the device is not encrypting. Since this part is assumed to be irrelevant to the key and input, attacker can remove it from the total power consumption simply by averaging the sampling results of a number of tests for the same input, key and state of the circuit. This of course is only if the SNR[7] is not so low that the signal is masked in the noise. If this is not the case then actions should be taken to reduce the effect of noise in the system. These can include filtering, shielding, finding the source of noise or any other method which helps noise reduction.

The remaining two parts of the power are what is actually related to the encryption process. One part, $P(desired)$, denotes the portion of power related to the component, or operation, that the attacker wants to find its consumed power. The other part, $P(sw.noise)$ (the power consumption of switching noise), denotes the consumed power of all the other blocks in the encryption process which consume power in parallel with the target operations.

Among the components that produce the switching noise, some of them are also working with the exact data that is the target of attack (the data that is used in the target operation of $P(desired)$). For example, at some moment when an addition operation between part of the key and input is occurring, there might be also other operations working on the same portion of the key. These operations generate one part of switching noise (and would be referred to as *type 1* switching noise from now on). At the same time other operations might also be executed, but they are operating on different parts of input and key. These form another part of switching noise (and are refereed to as (*type 2* switching noise from now on).

These two types of noise are the most troublesome factors for the attacker. Type 1 switching noise is better for the attacker since the noise is correlating with the desired data and thus not reducing SNR as much. However, type 2 switching noise can kill the signal and is the part that the attacker might be most concerned about. Next section would discuss this more thoroughly, when presenting some of the well known attacks and how they approach this problem. But as a general rule, parallelism and pipe-lining are known to be countermeasures for PAAs because they actually increase the amount of switching noise and specially the type 2 of switching noise.

[7]Signal to Noise Ratio

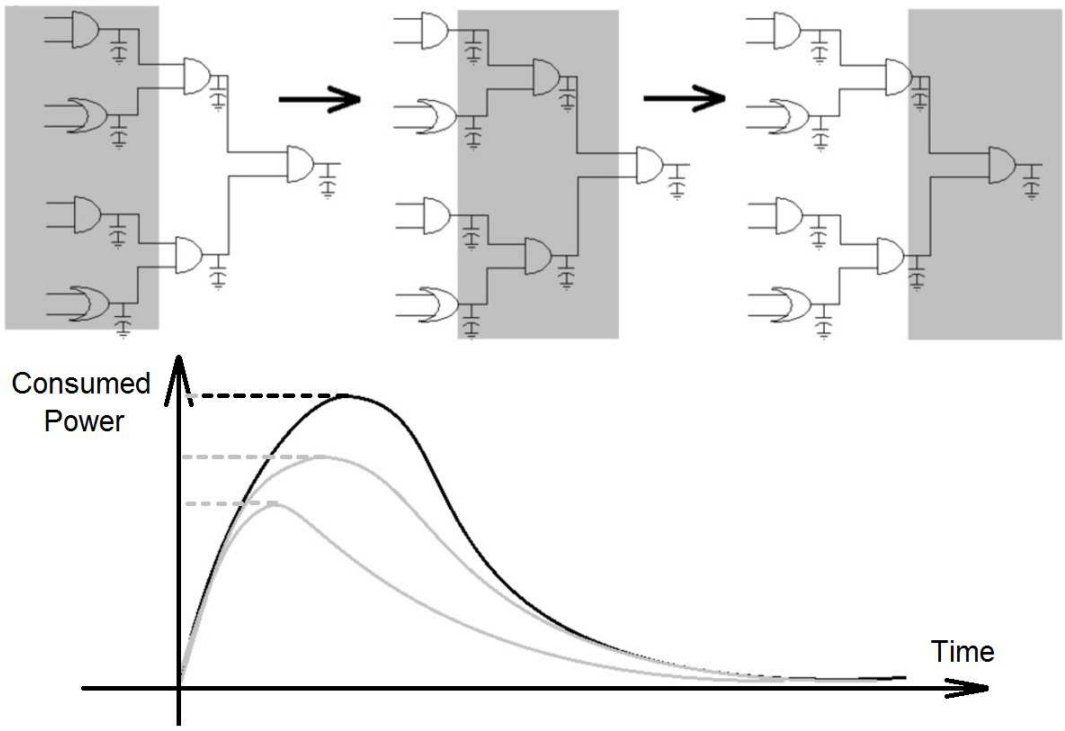


Figure 2.2: Data propagation and power consumption in combinational logic

Now let's discuss the power from another point of view. The power consumption of any device, including digital devices, can be written as:

$$P(total) = P(static) + P(dynamic) \tag{2.2}$$

In a digital device, the static portion of the power consumption is normally noticeably lower than the dynamic part. Most of the power is consumed in the transitions between the states, and the small leakage in the CMOS transistors creates $P(static)$. The dynamic portion, which is the main part, can again be split to:

$$P(dynamic) = P(combinational\ logic) + P(sequential\ logic) \tag{2.3}$$

$P(combinational)$ is part of the dynamic consumed power which refers to the combinational or asynchronous logic of the circuits. What is consumed in transmission data buses, ALUs [8], and other asynchronous blocks are what make this part. These transitions happen in different parts of the circuit with different delays, as the changed data propagate through the circuit. Figure 2.2 shows how this happens when the input

[8] Arithmetic and Logic Units

data is changed for some simple circuit. In this figure it is assumed that the delay is the same for all of the gates. When the input is changed, the effect propagates through the circuit. The capacitors at the output of first stage of the circuit are charged, and after a certain threshold they turn on the transistors at the input of the second stage. This process continues until the capacitors of output of the last stage are charged. A sample digram has been plotted showing the changes in power as time passes.

As shown in the plot some spikes are occurring with short gaps from each other. Depending on what the delay (amount of capacitance) is, these spikes might happen close to each other, and thus amplifying the effect of previous spikes (the case happening in this figure), or far from each other with the effects not overlapping. If there is another digital block which is functioning in parallel with this block, its effect would also be merged with the power consumption of this block. Overall the summation of effects of all these blocks creates the total power consumption at any moment. Most of the power consumed in combinational part of the circuit is of this type. However there is another source for the dynamic consumed power in combinational part of the circuit too.

The source of that portion of power is the transitions called glitches which happen temporary, and for a very short time, when the state of some intermediate signal changes from one stable state to another. Figure 2.3 presents an example of why glitches may happen in a digital circuit. In the figure red values show the new transition on some intermediate signal. As can be seen, the temporary transitions of the state of the circuit can cause change of the state of the output for a very short amount of time. But this change is enough to produce a spike on the total power consumption of that part of the circuit. There are methods based on Logic Design theory[17] to avoid glitches in a digital circuit, and normally RTL compiler should be capable of creating a glitch free netlist too. But still, though with a short ratio to the total power consumption of combinational blocks, the glitches might happen.

The consumed power in the combinational part of the circuit, has different ratios in the total dynamic consumed power, for different devices. In ASIC implementations for example, it is assumed that the design is highly optimized and thus glitches are reduced to minimum. The wiring circuitry doesn't consume much power either is ASIC devices. So the consumed power of combinational blocks, is generated in logic blocks such as LUTs, multiplexers, decoders and etc. This is totally different from the case that a reconfigurable device, such as a FPGA, is used. In a FPGA, normally

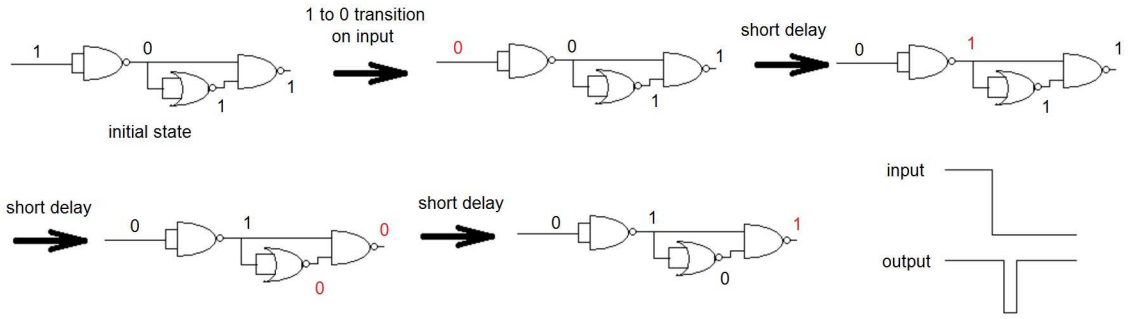


Figure 2.3: An example of occurrence of a glitch inside of a digital circuit

a huge portion of its circuitry is dedicated to connection and switching circuit, which is programmable and an active part of the circuit. A FPGA normally has buses of different lengths, each of which supporting short, medium or long connections. The longer connections pass through more switching circuitry, and they would have more delay, and accordingly more consumed power. For the long paths the delay and power consumption can be as high as 10 times of those of a logical block of the FPGA.

These long wiring and switching circuitry are commonly known as global interconnect. The global interconnect if used in an implementation can make the combinational portion of power consumption comparable, and actually higher than the consumed power of the sequential portion. The RTL compiler in FPGA normally tries to avoid using this circuitry in mapping the design as much as possible. But when the design gets bigger, using the global interconnect is unavoidable. Figure 2.4 shows the architecture of a typical FPGA (a matrix based pattern FPGA) in a simplified form. Vendors such as Xilinx provide their FPGAs in this form of architecture. The CLBs[9], and the wiring and switching circuitry are shown in this figure.

The other part of consumed power as said, is the sequential part. This part of power refers to all the transitions that occur at the edge of the clock in registers. A register is built using some number of Flip-Flops and the state of each Flip-Flop can change on the edge of some internal clock of the circuit. If the state of the Flip-Flop changes, some power would be consumed because charge/discharge of its internal capacitors. The clock inside of the circuit is assumed to be distributed in a way that there is no difference in the delay of the received clock in different regions of the device (this is done using a special branching technique[10]). Consequently, the

[9]Configurable Logic Blocks

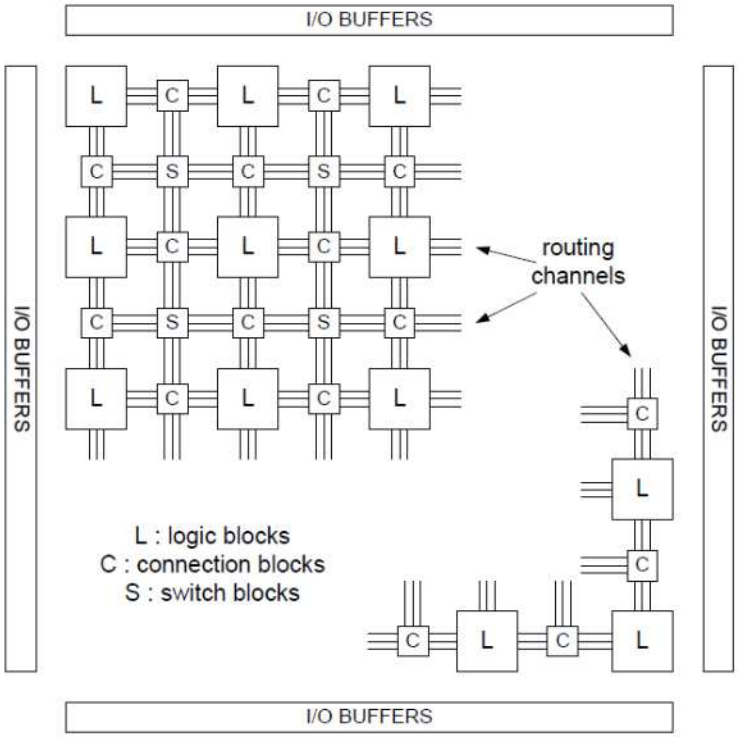


Figure 2.4: Simplified block diagram of a FPGA (matrix based architecture)

transitions of the Flip-Flops which are synchronized with the same clock happen at the same time.

For a Flip-Flop for which its input has changed, two transitions happen: The transition in the positive edge (if the Flip-Flop is sensitive to positive edge) which propagates the data through its first latch, and the transition in the second edge (negative edge) of the clock which propagates the changed data through the second latch and to the output. In other words there would be one spike in the power consumption in the positive edge, and another spike in the negative edge. Since there are normally many registers that are synchronized with the same clock in a design, noticeable spikes are expected to be seen right at the edges of the clock. The quantity of these spikes then is expected to be related to the number of the changes of the states of registers.

For example if the value of some register was 01101110 before the clock edge happening, and after the clock edge its new value is 01011100, then the consumed power is expected to be three times of the consumed power for a state change of a single Flip-Flop. The notion of *number of changes in the state of the register* can

be expressed by the known function HD[10]. $HD(a, b)$, where a and b are two digital numbers of the same width, is the number of bits that differ in the two values. Note that in the discussion above it is assumed that the transition from 0 to 1 consumes same power as 1 to 0. Although this in practice is not true, but the difference is normally much lower than the quantity of the transition itself. Thus even if this difference can be modelled, it would be an improvement to the HD model rather than distorting the results.

Hamming Distance is the main modelling tool in PAAs. The power consumption of registers are pretty much the only cause of the first peak right at the edge of the clock, so if this peak can be sampled then the total summation of all the Hamming Distances of the registers changing at that point would be obtained. Although the combinational part of the circuit can be modelled too, and in fact that part might reveal more information about a certain block of the design, but that requires intensive pattern recognition on the device. Moreover the design can always be rerouted in a way to reduce the delay between certain combinational blocks, and thus merging and masking their effects. But changing the behaviour of the sequential portion needs more attention and a new plan for the implementation. In next section it would be shown that how HD is used for obtaining the secret part of the implementation in some known attack methodologies.

2.4 Side Channel Attacks

SCAs as mentioned are the most common methodology used in attacking an implementation of an encryption algorithm. There are different kinds of SCAs and only some of the most common ones would be discussed here. While some of the attacks only rely on the visual observations of power consumption diagram some others use statistical models. *SPAs*[11] are from the first category, while DPAs and Template Attacks are from the later.

To express the idea behind SPA the same example is used as Kocher used in his first work[13] on this topic. In this implementation, a DES[19] block-cipher was used which was the most common block-cipher at its time. Rather than a FPGA or ASIC, a micro-processor was simply used to emulate the encryption process. Based on the diagrams provided, and since it is not mentioned otherwise, it is assumed that neither

[10] Hamming Distance
[11] Simple Power Analysis Attacks

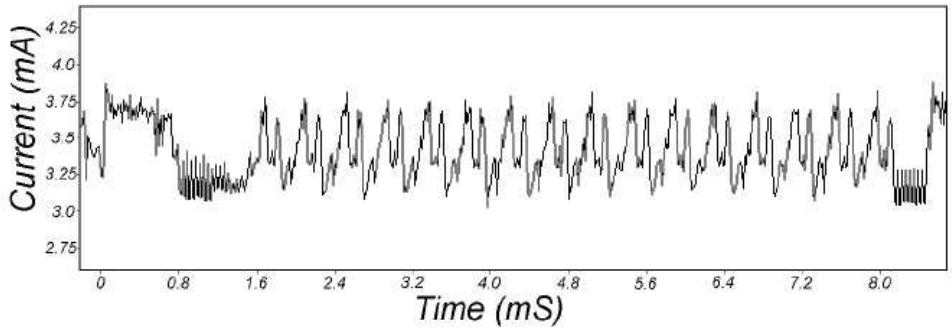


Figure 2.5: Encryption process for 16 rounds of DES[15]

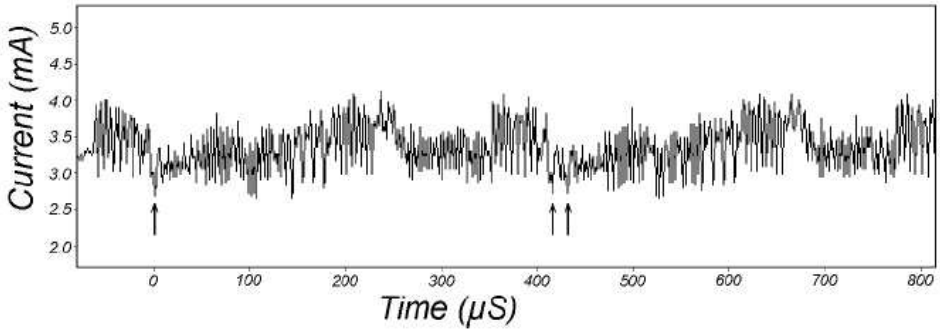


Figure 2.6: Enhanced plot of for one round of encryption[15]

the processor nor the assembly code benefit from any parallelism or clustering feature. Thus the processor executes instructions one at a time with no pipe-lining except its internal 4 stage pipeline (fetch, decode, execute and memory write back).

Figure 2.5 shows the power trace of the complete encryption. The DES algorithm consists of 16 rounds of expansion, key mixing, S-Box and permutation. As can be seen from the figure, the 16 rounds are quite visible in the diagram. Figure 2.6 shows the second and third rounds of the encryption in more detail. Amazingly the power traces of the two rounds are almost the same with some noticeable differences at some points. One of these exceptions is marked in the figure, which happens at the beginning of the two rounds. The next figure, figure 2.7, enhances this point more for the two rounds. This difference occurs at the 6th clock of each round.

Kocher in his work refers this to a branch instruction in the code which might/might not happen based on the value of some bit of the key. The upper trace is when jump has occurred, and thus more power is consumed, and the lower trace indicates a normal operation. Any conditional statement in the software code is normally compiled

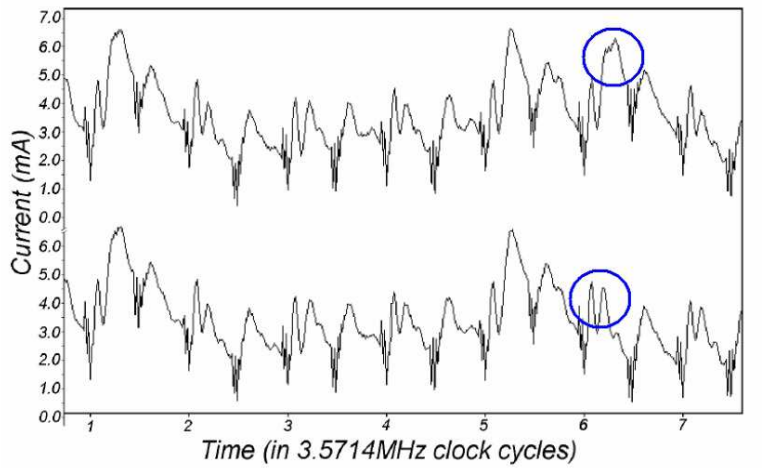


Figure 2.7: Difference in the power trace for two values of the key bit [15]

as a branch instruction at assembly level, and a branch instruction requires the flush of the pipeline and a load from cash or memory, which takes more power consumption too. So if there is an *if-statement* based on key bits in the code, the implementation would be highly vulnerable to SPAs.

The code developer should normally be aware of this, but at the time the method was proposed SPAs were not known. Avoiding the problem however is simple. With a little creativity and rewriting the code a little differently, such statements can be avoided in the encryption code. For example if the code below is included and is the source of the branch and extra power consumption:

if (k==1) perform function output=F(P) else output=P

This can be changed to $output = F(P)*k+P*\bar{k}$. This way two multiplications and an addition replace the branch statement. The average computation time is more in this case, but operations take the same amount of power and time no matter what the key is. One might think that the multiplication is itself vulnerable since in a normal multiplier there are conditional shifts based on the bits of multiplicand. However this different amount of consumed power in hardware core is way lower than what is being discussed here about software and the power consumed in a whole instruction. Thus the implementation would be safe against SPAs.

In case of unsuccessful SPAs however, other methods of PAAs are examined. DPA is the next option and is are the first example of statistical models for attacking an implementation. They come in different levels of strength, and the process unlike

SPAs is not that simple and requires a large number of samples and some analysis on the sampled data.

The first step in a DPA attack is for attacker to choose an intermediate value to attack to. The intermediate value needs to be a function such as $f(d, k)$ where d is a known value and k is all or part of the key. This for example can be the output of the last round where the cipher-text is known, or it can be the first register after the first operation in the first round where a known plain-text is combined with parts of the key.

In either case the intermediate value should be obtainable based on some known input or output and parts of the key. There might be more than one points in the encryption process that attacker is using for DPA. For example at some other stage of the encryption process, another function $g(d', k')$ might exist for which d' is also known and k' is the same or some other part of the key. Having more sampling points would increase the strength and is known as the order of the DPA. But in the following example DPAs of 1st order are only reviewed.

In a first order DPA attack, the attacker performs some number of tests on DUA, namely D of them. This number is normally 1000 samples or more and can go as high as 100,000 samples. One of the features of the DPA is that it would eventually show whether the number of samples was enough, or more tests need to be applied. Of course the number of tests should remain reasonable or else it means DPA is not effective. For each of these runs, the attacker would sample the power as t_i^1, t_i^2, ..., t_i^T, where T denotes the sampling rate and every t_i^j is referring to jth sample in the trace number i. T can vary and it is up to the attacker to how to sample data from a trace. She might just sample them at the edges of the clock, or she might increase them to a point that the samples seem continuous. Bigger T gives better results but comes with more analysis and calculation.

One advantage of DPA is attacker doesn't really need to know when the desired operation occurs, and the outcome reveals it automatically. She just has to choose the sampling rate high enough, so the important incidents in the trace are sampled. After capturing the samples, the matrix S, of size $D * T$, can be formed including all the values acquired from sampling.

The attacker then guesses the key, namely k_1, k_2, ..., k_K, as all the possibilities for the key. By knowing the cipher-text or plain-text for each run, and having a guess for the key, the intermediate values can be computed. She creates a hypothetical model based on the computed values. Another matrix V of size $D * K$ then is formed, where

$v_{ij} = f_i(d_i, k_j)$, and it denotes the expected values based on the encryption run and the guessed key. V itself is not useful since the samples of power are not directly correspondent to the values of the registers. So some model is used to transform the hypothetical matrix to one which can actually relate to the samples of power.

HD as discussed is one of such models which is generally used and is considered the strongest. If for example the registers holding the value were containing 0 before the function happening, and this is the first time they are loaded with data, the HD model of these changes would be the number of 1s in the register. This new value, as known as HW[12] of a binary number, can now replace the values in matrix V. The new values h_{ij} form the new matrix H which can be used for performing the attack. Note that in forming the matrix H some, and in fact many of the values, would be repeated since HW has a lower valid range than the actual values. An eight bit register for example can hold values between $0 - 255$ while the HW of it is a number between $0 - 8$.

Now using the two sets of information (H and S) the attacker compares the results using some method of comparision. Most often for doing so the correlation coefficients are used. The correlation coefficients of the two matrices are computed as follows:

$$r_{ij} = \frac{\sum_{d=1}^{D}(h_{di} - \bar{h_i}).(t_{dj} - \bar{t_j})}{\sqrt{\sum_{d=1}^{D}(h_{di} - \bar{h_i})^2 . \sum_{d=1}^{D}(t_{dj} - \bar{t_j})^2}} \tag{2.4}$$

The matrix of coefficients R, can give much information about the hypotheses and how good the DPA attack has worked. For any guess of the key, it examines the effect of all the D samples as a whole, rather than comparing each one of them individually. Also by using this matrix, there is no need to know the precession of sampling. In other words and for example, there is no need to know $HD = 3$ corresponds to what amount of power since the correlation coefficient compare the results in a relative way. It has some other advantages that would be discussed too.

[15] provides a series of examples of DPA attacks applied on software and hardware implementations of AES algorithm using different models, and provides the results of correlation coefficients. Some of them are used here too to explain how these values can be used. Each entry of R is a comparison measurement of the pattern of sampled signals with the pattern of hypotheses for a certain key and for a certain time. So one

[12] Hamming Weight

way of analysing the results is to plot columns of R versus time domain. Figure 2.8 is an example of such a plot. It refers to a software implementation of AES. The DUA here uses 256 bit AES, and the function used for the attack is the S-Box of the first round along with the plain-text. The model used for transforming the hypothetical values to matrix H is not Hamming Distance and is a weaker model, but that doesn't change the generality of the discussion.

As can be seen, in one of the hypothesis for the key, significant spikes have happened around a certain point, while for the rest of the time and rest of the plots the diagram is almost stable around 0. That point is indeed referring to when the target operation is happening in the implementation and indicates the its occurrence time. The plot shows that the right value for the key here is 255.

From the figure, two other things can be noticed: First, there is no other point in the same plot where a spike occurs. And second, there is no other spike for any other hypothesis either. The HW values in each column of H represent the computed hypotheses for D samples for that specific function of the implementation. So no matter what the pattern and the results of such computation is, it is only expected to see a correlation between the sampled results with these computed ones at the point that the function is being executed for the right value of the key. Having another spike in the same plot means somewhere else there is an operation performing almost the same function on the key and input.

There are actually such points in figure 2.8, where some spikes very close to the main spike are visible. These are happening when micro-processor is operating on the result of the function, and for example it can be happening because some other register or a memory location is loaded with the result of the function. If however the spikes happened for different key hypotheses and in different times, this would normally be an indicator that the DPA wasn't successful. One reason for such a case is that the target function is chosen poorly. For example if the target function is a linear function, normally the built hypothetical model doesn't say much about that function and its pattern can be seen in many other operations too.

Overall DPA itself would reveal how good the primary choices of attacks are. There is also the possibility that the plot is so jammed that no spike is observed. This might be solved with increasing the number of samples. Figure 2.9 shows an example of how increasing the number of traces might help in getting better results. In the figure the traces for less than 100 samples give close results for different hypotheses, while for about 500 traces the right key is quite distinguishable.

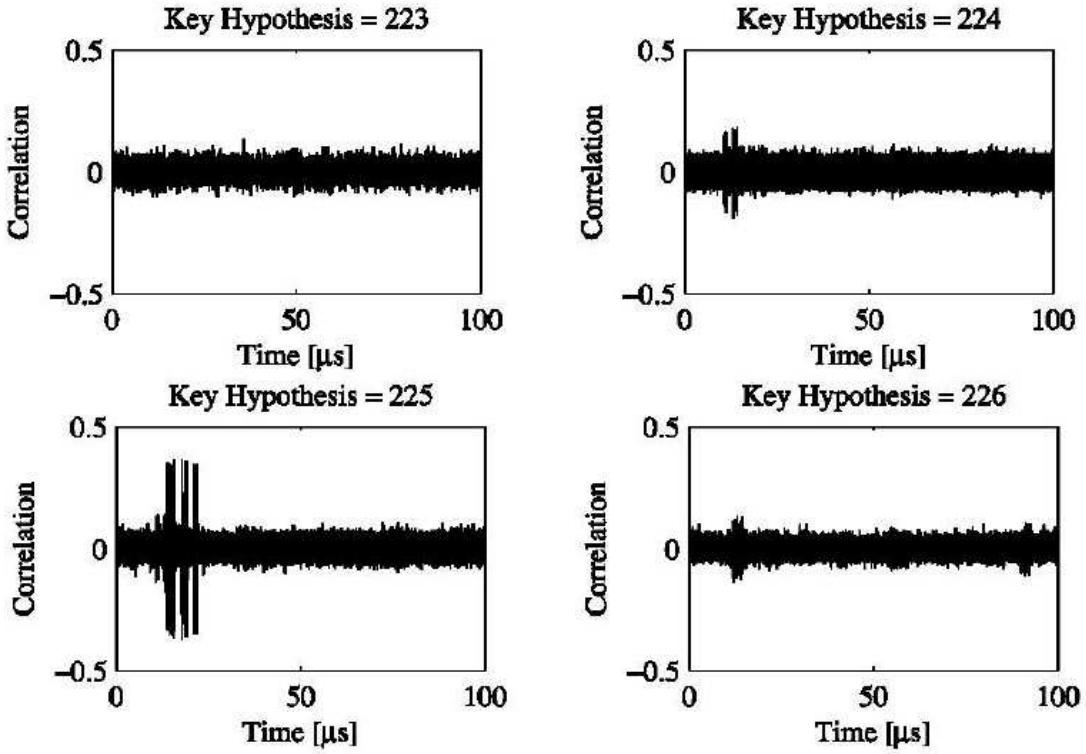


Figure 2.8: Rows of matrix R corresponding to some key hypotheses[15]

Now let's examine the a hardware implementation. It is expected to see more resistance in this case. Figure 2.10 shows the result when HD model is used, which comparing to the model used for the software implementation, is much more powerful. Also here, there are 100,000 samples used instead. The figure shows the correct hypothesis for all 16 rounds and as can be seen, although the spikes are quite visible, but their amplitude is lower than the software implementation case. Using less samples or a weaker model as the authors claim, leads to unsuccessful attack and no spikes to detect. However 100,000*256 samples (100,000 samples for each guess of the key) is still a reasonable number comparing to the all the cases in brute force method which is $2^{128} = 10^{13}$ cases and thus the implementation can be considered successfully attacked.

An important thing to notice here is the key is actually hacked here and not just narrowed down to a smaller search space. HD and HW when used alone, can only limit the search space. But here what has happened is for each possibility of key a pattern is generated and then transformed to HD or HW quantities. Though patterns are transformed values, but a matched pattern shows the right value of the key.

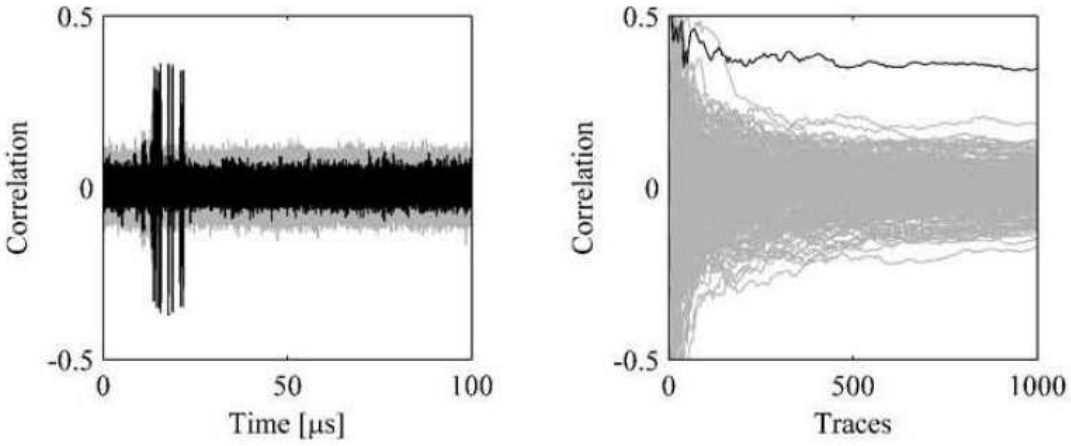


Figure 2.9: Increasing number of traces and better detection of correct key[15]
left: Correct key hypothesis=225 in black and other hypotheses in gray,
Right: Peak correlation for all hypotheses for different number of traces

And as a final note let's see how DPA deals with noise. As mentioned there is a switching noise in the digital circuit which is the main problem for the attacker. In a DPA the hypothetical model built for each guess only gives a pattern which would be correspondent to the power consumption of a specific function, rather than all the block affecting the power consumption at some moment. However the hypothetical model gives a pattern and a set of hypothetical values for each guess of the key. So now attacker has a list of values for each guess and she is hoping to able to find some correlation in the sampled data for the right guess of the key. However if the noise is too much, say because of high number of parallel logic blocks, she wouldn't be able to apply this technique. For more details and further studying about DPA attacks, reader can refer to other published works including [16, 18].

In cases where DPA doesn't work, DPAs of higher order or Template Attacks may help. Template Attacks are specially stronger in cases such as stream-ciphers where the key is changing as encryption proceeds so even if the final key is deduced for some cipher-text, it wouldn't help in finding the key for the rest of the encryption. The number of possible cases for the seed key in stream-ciphers, by which all the other keys are generated, is also usually high, so finding it using methods such as DPAs is impossible. The solution is building templates for some intermediate value or the final key, so for each key which is used for new plain-text word, the template can be compared to the sampled data and the key is obtained.

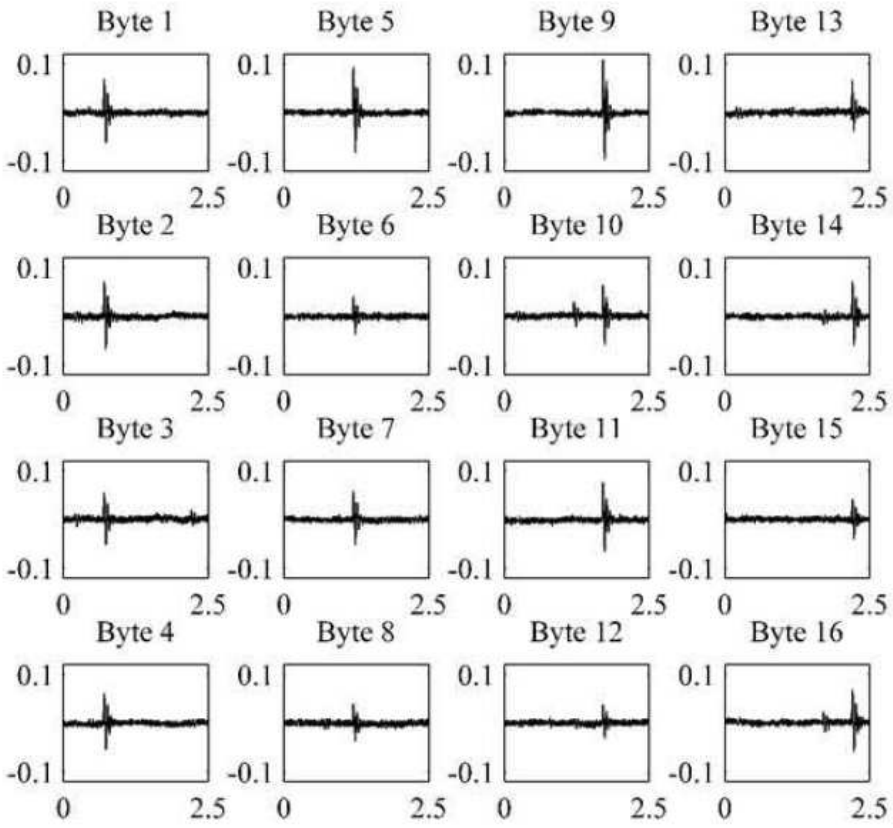


Figure 2.10: Correct hypotheses in hardware implementation of AES[15]

Template Attacks are based on multivariate normal distribution model which is defined by a mean vector and covariance matrix (m, C). Assume that a device runs a set of functions for various values of d and k. For every possibility of d and k a template is built and later the template is compared with the sampled data to find the right key. Building a template means being able to also change the key besides the plain-text. In other words for Template Attacks having an IED is necessary.

The exact specification goes as follows: For each set of d and k a number of samples, namely D of them, are applied and then the attacker takes average on the obtained samples. The result is $d * k$ traces such as m_1, m_2, ..., m_{d*k} where $d * k$ is the number of possibilities for input and key. Now for each run of the test with the trace T, attacker considers the *noise vector* $n = T - m_i$ for building the covariance matrix of m_i. Since the size of this matrix can grow rapidly, unlike DPAs, only limited number of points are chosen. Normally what attacker does is she subtracts each pair of average traces and considers the points in $m_i - m_j$ where a noticeable

peak happens. These points are more likely to be related to the attack, since the areas where $m_i - m_j = 0$ mostly denote that the power is not really dependant on the pairs of key and input.

By choosing these points, C_i can be formed where $c_{uv} = cov(n_i(u), n_i(v))$. The matrix will represent the distribution of values of the noise vector relatively. After building up the templates the attacker can perform the comparison. The probability function below is what is used:

$$p(t, (m, C)_{d_i,kj}) = \frac{\exp\left(-\frac{1}{2}.(t - m').C^{-1}.(t - m)\right)}{\sqrt{(2.\pi)^T.det(C)}} \tag{2.5}$$

The function might seem rather complicated and the mathematical computations behind it is beyond the scope of this thesis. However it is basically a better measurement for likelihood of two traces based on the covariance matrix. The assumption here is that the noise vectors form a multivariate normal distribution and the above equation is derived from properties of such a distribution. Ofcourse such an assumption is only true if the templates include some information regarding the operations on d and k. Otherwise if the target operations are chosen poorly, or the noise is so much that the effect of operations is masked in it, the results wouldn't form a multivariate system either.

In anyway, the results themselves would show that if the attack has been successful or not. The above function gives the probability that some template (m_i, C) is a match for the trace t. The equation above would also denote the number of necessary samples (T) to achieve a certain probability. There are other considerations about how to make sure that the covariance matrix is invertible, and how to prevent the small results for the exponential function which make the comparison difficult, but these are not much of the interest of this thesis and the reader can refer to [7] for more details about these topics.

Template Attacks in general are shown to be stronger than DPAs [7, 15]. One reason as said is better comparison mean. The other reason as might be apparent from the technique is that Template Attacks use samples from actual device to build templates, comparing to DPAs where hypothetical model is computed on paper. A template is a trace versus time corresponding to some value of the target function, rather than hypothetical values of the function. Also Template Attacks can be used in cases such as stream-ciphers where DPA is hard to be applied. For example, imagine in a stream cipher some long key is used as the seed and that, along with the state

of the circuit, generate some intermediate keys which eventually end up in building the final key of each stage. Because of the number of possibilities, attacking the seed or state of the circuit might not possible. However the attacker can take another approach. She can consider building templates for the final key, which has way less number of possibilities, and then compare the samples of DUA with templates to find the keys for each word of plain-text.

Overall Template Attacks have shown to be stronger than other forms of PAAs in breaking an implementation. However this comes with the cost of large amount of preparations and also availability of IED. More about template attacks can be studied in [3, 9, 22].

2.5 Other Attacks

In this section and just as a very brief introduction some other forms of attacks would be discussed. These include Fault Tolerant attacks as an example of non-SCA attacks, and EM and timing attacks as other forms of SCAs. Since these types of attacks are not referred elsewhere in this thesis, the overview would be very brief. The interested reader can look for more references[2, 8, 11, 21, 26] along with many other resources available on various other forms of attacks.

Let's start with Fault Tolerant attacks. The idea behind these sorts of attacks is that by introducing optical beams such as optical laser or photo flash of a camera, a fault, which is a change in state of Flip-Flop or register, can be induced. In [25] the authors have used cheap photo flash lamp of a camera, along with a microscope and were able to induce faults to individual cells on a SRAM separately. The building block of the SRAM of this example is a 6 transistor base Flip-Flop which is shown in figure 2.11. The illumination of the targeted area causes an ionization and opening the transistor T3 for a very short amount of time causing the Flip-Flop to change its state. This can be observed by programming all the bits of the memory to 1, downloading its content after illumination, and observing the changes in its values. The device used here was a PIC16F84 micro-controller which contains 68 bytes of such a SRAM.

The paper in short suggests that fault induction is easily possible on the hardware devices with high accuracy, using cheap material. Using the microscope to locate the cell to attack and using aluminium foil to cover the rest of the circuit, they were able to induce faults to each single cell. This induced fault is then so useful in finding

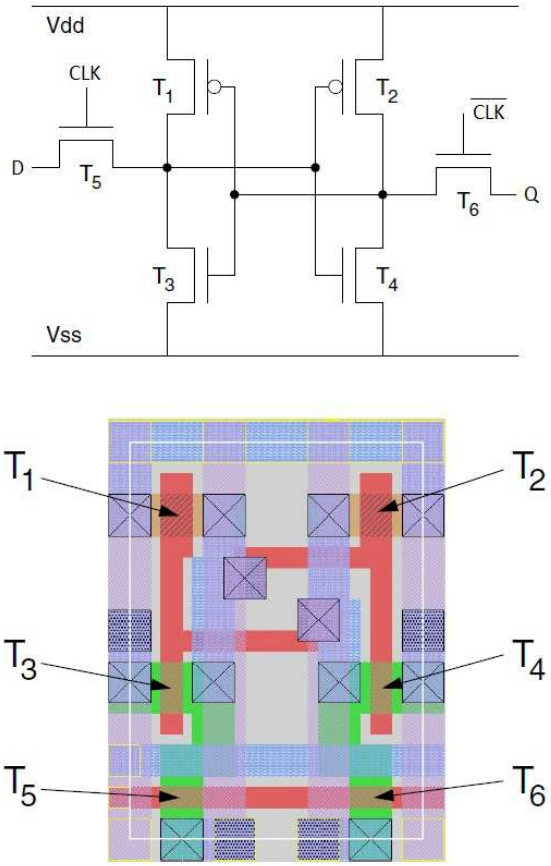


Figure 2.11: Circuit layout and schematic for a 6 transistor Flip-Flop [25]

the secret parts of the implementation. As a simple example assume an eight bit register for which some information is known based on some other kind of attack. This can be the HW of the register for example. If after inducing the fault on the first bit the HW is measured and it is less than the first case, this shows that the induced fault has caused the bit to switch to zero and so its previous (original) value was 1. If on the other hand the previous HW is less than the new obtained one this shows that the first bit was originally 0. With 7 tests and measuring the HW of thr register after inducing the fault, the value of the register is known. There are many other methods, some quite complicated, which use the idea of inducing faults in their attack strategy[4, 27].

The next topic to discuss is EM attacks. Since year 2000, EM attacks were started to be noted and different works were published about applying them to different algorithms and implementations. In 2002 some of these works were collected and presented as a survey[1], officially introducing and categorizing the concept. Here

some of the ideas used in that publication are described but the interested reader can also refer to [2] for more study on the subject.

EM emissions around a digital device can be categorized as *intentional* or *unintentional*. The intentional emissions are the ones caused by the current flows in the device while the unintentional ones can be caused by anything else. The environment can be one minor source, but the more important source, specially in CMOS devices, is caused by EM coupling between the different components in transistor level. The current flow in one transistor might have an EM effect in the neighbouring transistor, which shows itself as a coupling effect. This often shows itself as modulations of the carrier signal in the form of amplitude or phase modulation. A typical example of a carrier signal inside of a digital device is the clock and since normally the sources of power supply for the clock and the rest of the circuit are the same, the carrier is then modulated by the effects of rest of the circuit which includes valuable data. The modulated form can be demodulated, and practice has shown that useful results can be obtained of such an experiment.

[1] examines a DES encryption implemented in software and studies the frequency response of the demodulated signal for both amplitude modulation and phase modulation. These observations have been done both for far-field and near-field signals. Figures 2.12 and 2.13 provide the primary results of such experiments in near-field signals.

In figure 2.12 a 283KHz signal and its harmonics are quite visible (the high peaks) even in this non-scaled non-logarithmic plot. This frequency is the frequency of loop iteration which is 13 cycles of clock in this particular implementation. This plot tells that the information regarding encryption process can indeed be obtained from the demodulated signals and it can be used for performing known attacks such DEMA[13]. [1] provides examples of results of attacks based on this idea. Rather than the 283KHz signal a number of carrier signals were examined and yet the results were promising. In fact the authors have compared these results with a DPA attacks and in some cases DEMA has revealed more information about rest of the circuit.

In another example figure 2.13 presents a sample of phase modulation. The figure presents the frequency response of the result for two different values of LSB of key, which in that specific implementation, is the cause of this difference. Authors describe this result as a coupling effect between the LSB and the clock circuitry which in case of $LSB = 0$ slows down the clock a little bit. This by definition is an example of a

[13]Differential EM Attacks

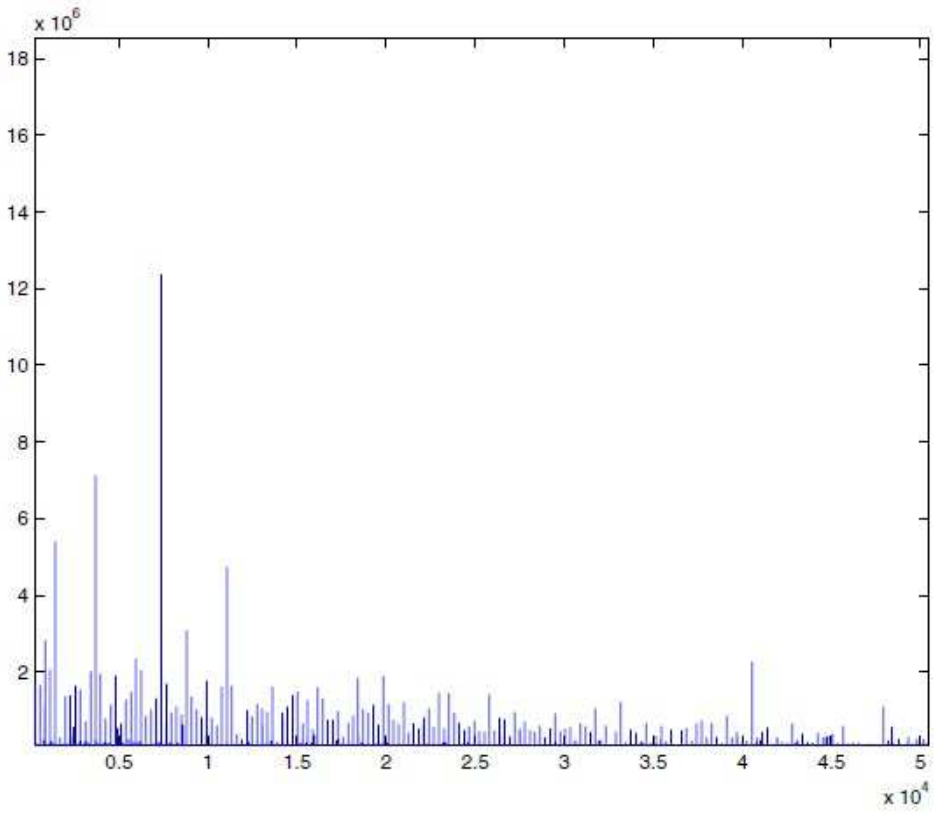


Figure 2.12: Amplitude of FFT of demodulated signal[1]

phase shift and would be detectable in a phase demodulated signal. Since in an EM attack sources of information are from different parts and come in different frequencies with different kinds of modulation, in general EM attacks are considered to be more effective. However processing the data normally needs more effort in these category of attacks.

As the final example of this section and this chapter let's study the timing attacks. Timing attacks are not as strong as other forms of SCAs, nor they are normally used in complicated statistical methods. But in the software implementations or implementations with variable processing time, they can be considered as an attack methodology. The concept behind these sorts of attacks was actually implied in previous sections. As an example assume the same application used for SPA attacks in section 2.4. As mentioned there is a conditional branch in the parsed code based on which some bit of the key may or may not cause a branch. A branch instruction as mentioned is treated differently and the pipeline is flushed and a memory load might occur if there is a branch. In some processors this means a few extra clock cycles

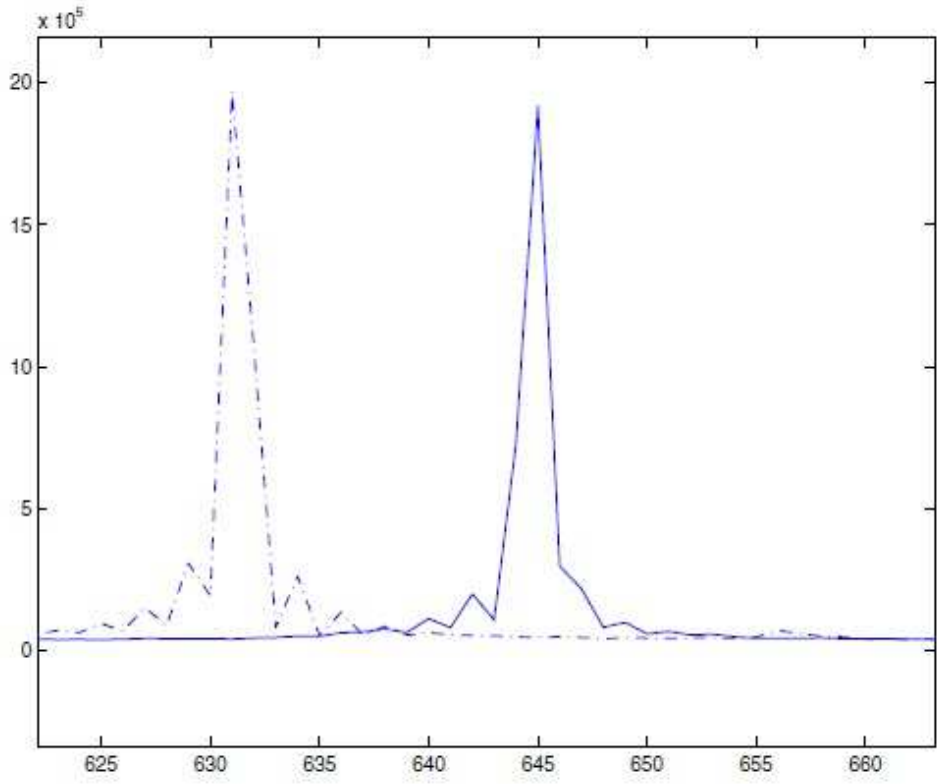


Figure 2.13: Amplitude of frequency response for two cases of LSB[1]

which can easily be detected.

For hardware implementations however it is not the same. Normally the time it takes, as the number of clock cycles for execution of some operation, is fixed for different inputs. A multiplication process, same as the one described in 2.4, takes either one clock cycle for all of the process or one clock cycle for each bit of multiplier to be processed. It is very unlikely that for the multiplier to take a different approach for two values of input. Some adders and multipliers use specific kinds of carry chain computation to reduce the required computation time, but even in those circuits the number of required cycles remains the same for different inputs and only the frequency of the clock can be improved. Overall hardware implementations seem to be not very suitable for timing attacks, unless for some reason the time of the process is variable for some operations. This however is unlikely, and most of times it is caused because of a poor design, rather than an intentional planned property of it.

Chapter 3

Whitenoise

In this chapter the Whitenoise algorithm will be discussed. First the algorithm itself is introduced. Then the specific implementation provided by Whitenoise Lab, which is meant to be a strong implementation against indirect attacks, is studied. Next the implementation will be examined against DPAs and Templates Attacks as two common attacks. It will be shown that these methods are not applicable on Whitenoise, or at best, they will give weak results, and thus a new approach is necessary. The scenario developed in this project for attacking the implementation then will be presented and its performance and requirements will be reviewed. In the end a conclusion on the topic will be presented.

3.1 Whitenoise Algorithm

The idea behind Whitenoise stream-cipher comes from this basic theorem in number theory that if there are n distinct prime numbers p_1, p_2, ..., p_n, the least common multiple of them would be $p_1 p_2 ... p_n$. Therefore, if there are n sub-keys (intermediate sets of key produced by some seed keys) k_1, k_2, ..., k_n, each of the length p_i bytes, the summation of them given as:

$$S(x) = k_1(x \ mod \ p_1) + k_2(x \ mod \ p_2) + ... + k_n(x \ mod \ p_n) \tag{3.1}$$

has a repeating period of $p_1 p_2 ... p_n$ for all $x \geq 0$. x denotes the cycle of execution, and for each x, $k(x \ mod \ p_i)$ represents the *next* entry of each sub-key, that is chosen for the summation. For a simple visualization, one can assume each sub-key as a circular shift-register, and in every cycle one shift operation to left (or right) is performed,

while some entry with fixed position in the shift-register is used for the summation. This simple idea provides a high repeating period for the summation of sub-keys, since for a series of small numbers of p_i the multiplication of them can be really large. Having shift-registers, it would also make it possible to have a highly parallel and pipelined implementation as it would be seen.

In Whitenoise algorithm up to 10 such sub-keys can be chosen, and length of each one can be a prime number between 2 and 255[5]. This is built as such for implementation purposes, so the length can fit in a single byte. The 10 smallest prime numbers between 2 and 255 include 2, 3, 5, 7, 11, 13, 17, 19, 23 and 29, and, for these values, the repeating period is in order of 10^{10}. However, the repeating period can be as large as 10^{23} for the 10 largest prime numbers between 2 and 255, which are 251, 241, 239, 233, 211, 199, 197, 193, 191 and 181. In a system with a fast clock of $1GHz$, which performs a full search on the whole possibilities of summation of sub-keys, it takes 10^{14} seconds, which is about 3 million years, to go through a whole repeating period. Thus the attacking strategy can't be based on repeating period of the initial state of sub-keys.

The summation of sub-keys in equation 3.1 are called *super-keys*, which are the next level of intermediate values before the final key is generated. At every cycle a super-key is generated and stored, and two of the generated super-keys, namely the one from a cycles before $(x - a)$, and the one from b cycles before $(x - b)$, are used as entries for a non-linear S-Box, which takes the two bytes and gives a byte as an output.

$$Z(x) = SBox(S(x - a), S(x - b)) \tag{3.2}$$

The result of this operation is summed up with some previously computed super-key to generate the final key of each stage.

$$K(x) = S(x - c) + Z(x) \tag{3.3}$$

The use of super-keys from previous stages implies having some sort of storage for those values and this can be done using a FIFO, as it is the case in the implementation available. The algorithm doesn't specify which super-keys of FIFO should be used in equation 3.2 and 3.3, and in the implementation available, there are control registers, which are programmable by the user, and identify the super-keys that are used as the inputs of S-Box. Since nothing is mentioned about them in the specification,

it is assumed here, that the super-keys used are independent of key and known to the attacker, who has access to the implementation. About the S-Box, again no information is provided, but since the strength of the algorithm against mathematical and direct attacks is not really the scope of this report, it can ignored here.

For generating the sub-keys, the algorithm uses two 32-bit seeds. The square root of one of the seeds is computed, and the digits after the decimal points are used for computing the values of each byte of every sub-key. The irrational numbers such as the square roots have an infinite repeating period after their decimal point, so for any different value of the first seed a different non-repeating set of values for the sub-keys is generated. The second seed is used in a pseudo-random function to choose the starting digit after the decimal point. So for different values of the second seed, the set of sub-keys can variate significantly. As a result, not only bytes within the sub-keys are different and have infinite repeating period, but also each two seeds generate a totally different sets of such sub-keys, too.

In this project however, the block which generates the sub-keys out of the two seeds is not going to be discussed. It wasn't provided as part of the implementation, and it was assumed that the values are embedded into the device or would be programmed in it. Thus this part of algorithm is outside the scope of this thesis. But for exact specifications one can refer to the software specification of algorithm provided as an open source[5]. The strength of the algorithm towards direct attacks has been previously examined[28, 29] and results show that algorithm is highly resistant to direct attacks. What is left is examining its resistance towards indirect attacks.

3.2 Whitenoise Implementation

The implementation originally was provided as a RTL[1] code for one of the Virtex II FPGAs which is a family of FPGAs from Xilinx[33]. However the FPGA was a little old for the time this project was under development. Thus another FPGA from Xilinx from the Spartan 6 family was used, and with a little bit of modification in device specific functions used in RTL, the new code was ported to the new device. The implementation provides two separate blocks for generating ciphers for two different sets of sub-keys, and each block contains logic for the main encryption function as well as for programming the control registers and contents of shift-registers and S-Box.

[1] Register Transfer Level

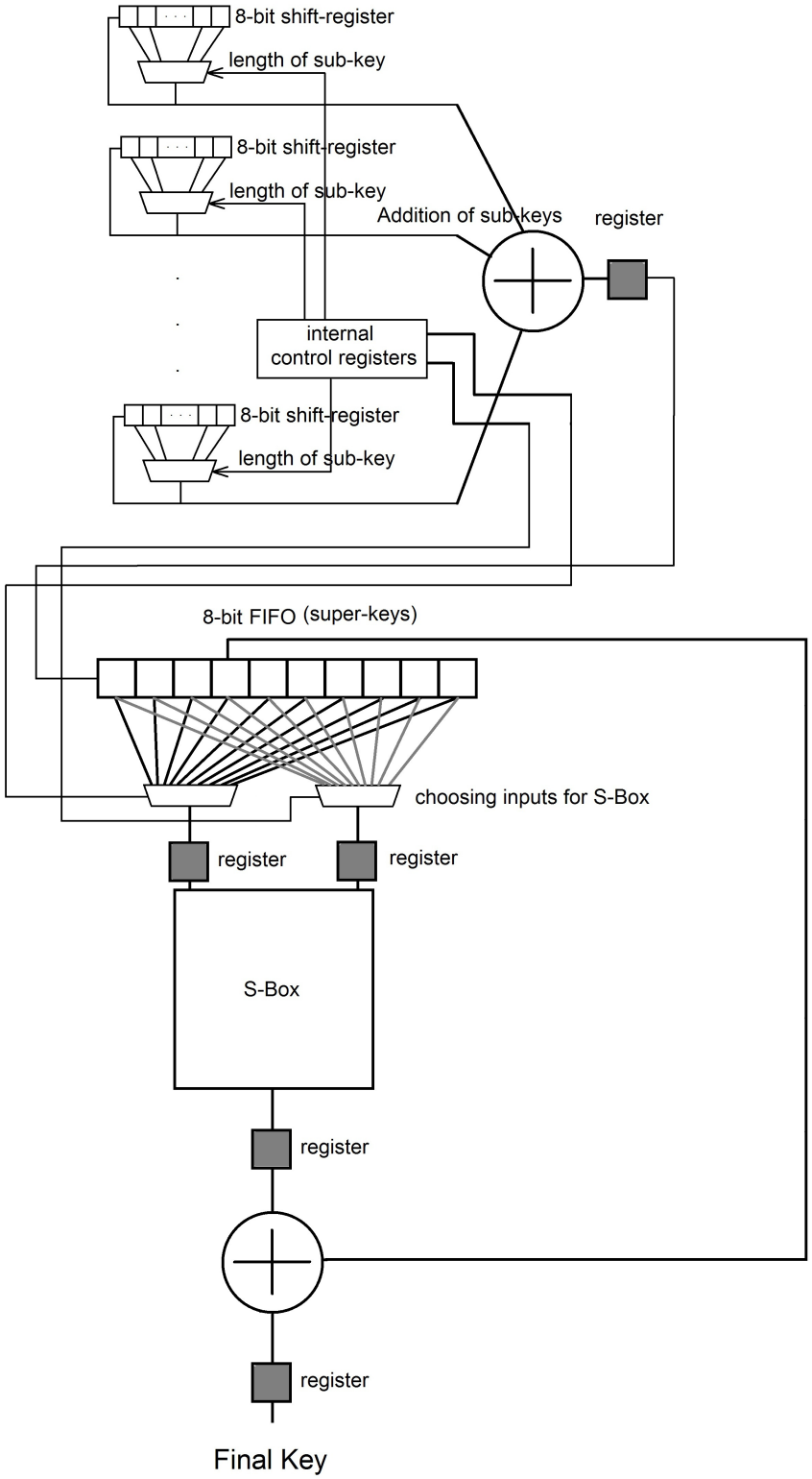


Figure 3.1: Block digram of Whitenoise implementation

Figure 3.1 shows the block diagram of the implementation for one encryption block and when programming circuit is ignored. This block diagram has been obtained by reverse engineering the code and the post place and route map of the design. As the figure suggests, the sub-keys are implemented using shift-registers of constant length. A shift-register of length 255 byte is used for all of the cases, and what actually identifies the length of the sub-key is where the feedback is connected, and this feedback also defines the output of each cycle. For each shift-register, a multiplexer takes the outputs of all of the bytes and based on the programmable control register, it chooses one of them to be used as the output and feedback value. The length of feedback as said, should be of a prime number between 2 and 255.

As can be seen in figure 3.1, the outputs are then added and stored in a register as the super-key of each stage. A FIFO of length of 10 bytes keeps track of these values, and these super-keys are used as inputs of S-Box and also to create the final key. Which super-key to choose is due to the user and programmable. Two internal control registers determine the indices of entries of FIFO that are used as the inputs of S-Box as equation 3.2 describes. Inputs of the S-Box are registered and fed to it, and the output of it is also registered. By using these registers, the implementation provides a pipelined design, and as can be seen, the pipeline is broken to the smallest blocks possible where in each stage only one shift, addition, or S-Box computation is done, and all of these operations are computed in parallel. In the end, the registered output of the S-Box is added to a super-key, then registered and finally provided as the final key of each cycle. Which entry of FIFO to use, is again determined by some programmable control register, which represents the implementation of equation 3.3.

The implementation has an internal *mode register* and a shift-enable pin. The mode register chooses between the program mode and run mode. In program mode the values of sub-keys, their length, the values of S-Box and the indices of super-keys used for the process can be programmed. A single data line in a two step process identifies the offset address of memory to write to, and then writes some value in it. The run mode on the other hand is the normal encryption mode. The shift-enable pin enables shifting in the shift-registers in both modes, and also shifting in FIFO in run mode. Later the importance of this implementation technique of circuit will be seen, as it is used in the proposed attack scenario.

3.3 Common Attacks on Whitenoise

Before talking about attacking Whitenoise, let's first look at the implementation from another point of view, the attacker's point of view. This will also reveal some of the reasons of why the algorithm and implementation are build as the way they now. The first question for an attacker is where to attack. Since the part of the implementation that generates sub-keys out the seed keys is not provided, and the sub-keys are assumed to be uploaded offline, attacker can not perform her attack on seed keys. She can instead apply it to find values of sub-keys, super-keys, or the final key. The number of sub-keys is large, as large as 2136 bytes for the worst case and largest prime numbers. But if she is able to find the sub-keys, she can predict the state of the circuit and of course the final key in any moment of the encryption process.

On the other hand, she might not be able to find the sub-keys, but be able to deduce some specific supper-key in the FIFO. In this case she still would be able to deduce the final key with some delay, or predict it for some future cycle based on the position of enrolled super-keys. For example, assume that for some reason, attacker is able to reveal the content of the super-key of the second register from the left in the FIFO. And assume that the fourth, sixth and seventh super-keys from the left in the FIFO are used in the encryption process (the sixth and seventh registers are used as the inputs of S-Box, and the fourth register is used in the final addition). Now if the attacker obtains the content of the second register in the FIFO at the current clock cycle, at the next clock cycle, and the next fifth cycle from current moment, then by performing the necessary calculations (S-Box operations and an addition), she would be able to find the final key of the eighth clock cycle from current moment.

This scenario is shown in figure 3.2. In the figure, the second register in the FIFO is measured at three different clock cycles, and these measurements are indicated with dark shade. The light shaded registers show the propagation of the measured contents through the circuit in the next cycles. The counter shows the clock cycle index of every step, and as can be seen, at the ninth cycle the final key, which is related to measured data, is obtained. Using equations 3.2 and 3.3 the attacker can find the final key based on this measured data. However for doing so, she needs to know the content of S-Box. Since nothing has been mentioned about the S-Box dependence on the key, and since S-Box, as part of the algorithm, is normally known to the attacker, it is assumed here that the content of S-Box is known to her.

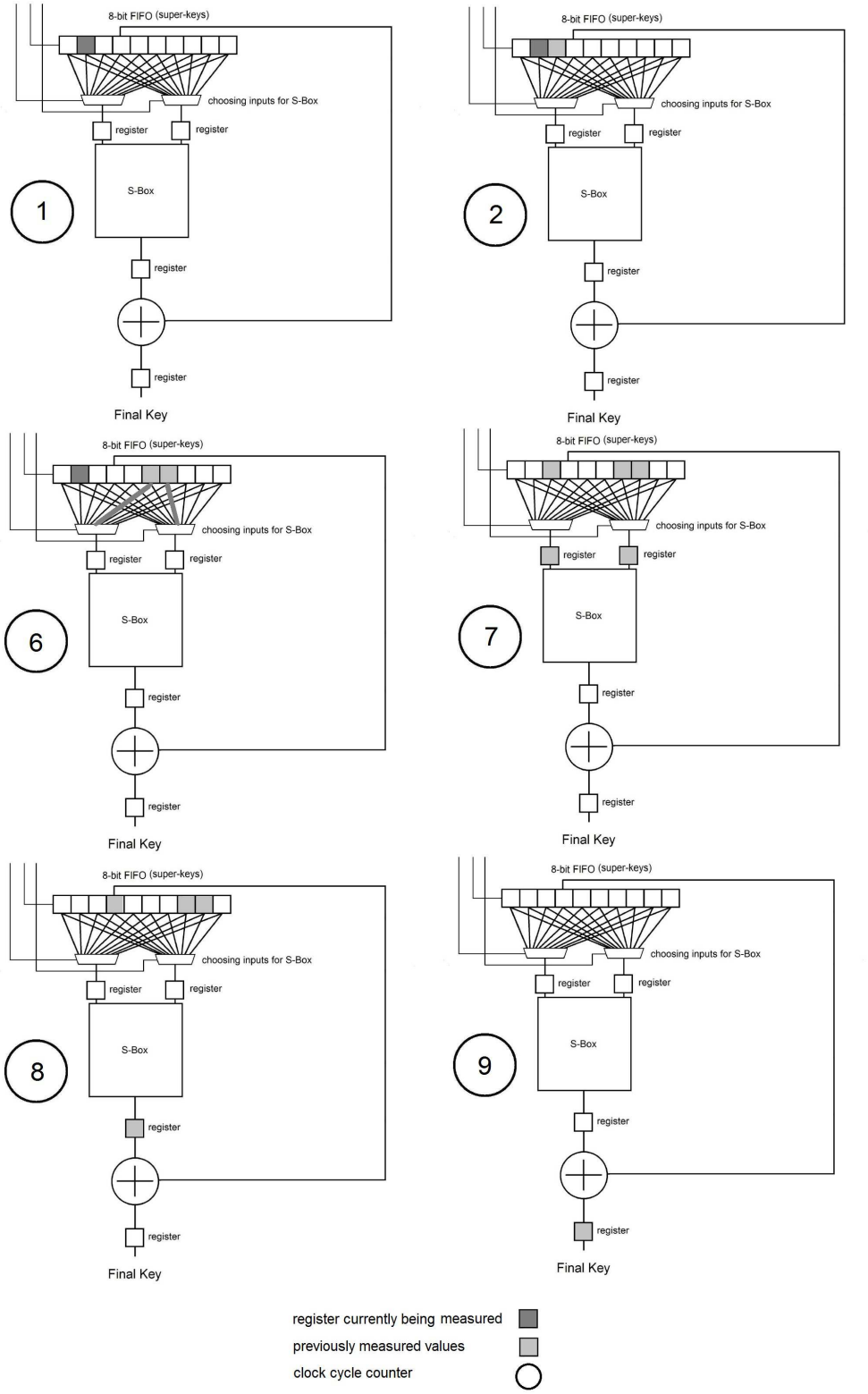


Figure 3.2: Propagation of data when a certain entry of FIFO is the target

The last option as the target value for the attacker, is the final key itself and attacker may try to build models or Templates based on the values of final key. This case would also be discussed in common attacks too.

One might think of the output of the S-Box as another target to attack. S-Box, as mentioned in previous sections, is necessary for making the algorithm resistant to direct attacks, but is a vulnerable point when considering indirect attacks such as Side Channel Attacks. However, in a stream-cipher such as Whitenoise, choosing the output of the S-Box as the target of the attack is not so helpful. The reason is, the key can't be obtained by obtaining the output of S-Box. Equation 3.3 shows how the result of a S-Box operation is added to some previous super-key to generate the final key, and thus besides the output S-Box, that super-key needs to be known too. It might appear that this value can be calculated by measuring the output of S-Box in some previous cycle, but there exist a problem in this case, as described below.

In a stream-cipher such as Whitenoise there is no guarantee that the S-Box is invertible. In block-ciphers, the S-Box should be invertible since the decryption process needs it to decrypt the cipher-text. Here however, all the decryption side needs to do is generating the same final key for each plain-text, and adding it to the cipher-text to find the plain-text. As a result, the same S-Box is used on both sides and being invertible is not a requirement. Since no particular information is provided about content of S-Box, the worst case is assumed here that attacking output values of S-Box is useless and the final key can not be computed by just having the output of S-Box.

What is next important to the attacker is the amount of time, or in other words number of cycles that the sampled data are useful for performing the attack. For each of the cases discussed, this can be different. If sub-keys are the target of the attack, the time value ranges from 0 to infinity, since sub-keys are enrolled in all operations continuously. In shift operations within the shift-registers as well as the addition of outputs of shift-register, sub-keys are enrolled directly, and in shift operations performed within the FIFO, the S-Box operation, and the final addition, the sub-keys are included indirectly. This is because every value in the circuit is built using the sub-keys.

On the other hand, if super-keys are the attacking target, the number of available samples is limited. From the first cycle that a super-key is generated until it goes through the FIFO and rest of the circuit, there are 14 clock cycles in which the super-key value is enrolled, directly or indirectly, in the operations. These operations include

the addition of the last byte of shift-registers where the super-key is generated for the first time, the 10 shift operations in the FIFO from that moment, S-Box operation, and the final addition.

There is not much point to consider other cycles rather than these 14 clock cycles as available samples. Before these 14 clock cycles, the super-key value is not generated yet. The 10 sub-key bytes that have created this super-key value, have some correlation with it too. But to consider the correlation of a single sub-key byte, to the super-key value it is partially enrolled in generating it, is useless and this correlation is very weak. So samples before these 14 cycles don't give much information.

As for the case of attacking the final key, the situation is somehow similar to attacking super-keys, with this difference that all the related operations occur before generating the final key, thus the value of final key is indirectly implied in those operations, and correlation of obtained samples to the final key is less and weaker.

Finally let's examine the noise in each of the cases above. When attack'ing the values of sub-keys, the operation on a specific byte of the sub-key, which is a right shift to the next cell, occurs simultaneous with the shift of all of the other bytes in the same (or different) sub-keys. This operation also occurs in parallel with the rest of the operations in the circuit: updating registers of pipeline and FIFO. The noise in this case is really high and most of this noise is totally independent from the value of that particular byte of sub-key. The values of the FIFO are in a way related to that byte, as small this correlation might be. But the values of the other sub-keys are totally independent. So most of the noise here is switching noise of type 2 with a very high SNR ratio, which can be as high as 1/2150. This is in case that largest prime numbers are chosen as the length of the sub-keys.

For the case where a super-key is the target of attack the amount of noise is a bit reduced. The number of parallel registers is still the same but the ratio of correlating registers is more which means more switching noise of type 1 and less of type 2. This of course comes with the drawback of having less samples available for the attack. For the last case where the final key is under attack, most part of the noise is switching of first type with the effect of the key indirectly implied in all of them.

Let's now talk about Whitenoise against Power Analysis Attacks. DPA and Template Attacks are discussed here. DPA attacks are in fact not so useful against Whitenoise as they aren't for many other stream-ciphers, and there are many reasons for this. As a reminder in DPA attacks first an operation is chosen as the target of the attack. The operation should be between the input (plain-text), or output(cipher-

text), and the key. Attacker runs a number of tests, with known inputs (or outputs), and captures the results. Since she knowns the inputs (outputs), she can compute the results of the target operation for the captured samples by assuming a value for the key. Thus for each guess of the key, she has a vector of results of the target function for different inputs (outputs). Using some model such as HD, she then transforms this vector to a hypothetical model. This hypothetical model is in correlation with samples of power and she finds the correct key by observing which vector matches the samples of power the best (and thus for which the guess of the key is correct).

Now let's see why DPA is not useful against Whitenoise. The first problem is which part should be assumed as the unknown part (the key) which attacker desires to find. The large number of sub-key bytes make them not be good candidates. As said, there can be up to 2136 bytes of sub-keys, and since each case can have up to 256 possibilities, the total number of possible cases (guesses of key) is huge (256^{2136}) in this case, and therefore, DPA is not applicable on sub-keys.

Super-keys and the final key are also not good candidates as the secret part of the implementation. For applying the attack, the key for a number of runs of encryption should remain constant. The super-keys and final key are continuously changing, and since it can not be predicted when these keys have the same values, there is no way to distinguish the samples of encryption which correspond to same value of the super-key or final key.

There is one other problem too. As mentioned in DPA attacks, attacker runs the tests for some known input (or output) values. She needs the input (or output) to be known, so by guessing the key and using the known parameters she can build the hypothetical models. The only operation here between the key and plain-text (input), or the key and cipher-text (output), is a simple addition (XOR), which as mentioned before, is a linear function and can't really be used as the target of the attack (since many operations have linear characteristics, a hypothetical model for a linear target would cause spikes for many guesses, besides the correct key).

The next option is the Template Attacks. In Template Attacks, first and as mentioned, the target of attack, which is an intermediate value of the implementation, or output of some operation, is chosen. Then a number of tests, for which the target value is the same, is applied. The power traces of those tests are then averaged, and used in a covariance matrix, to build templates for various possibilities target of attack.

About choosing the target of attack, sub-keys again are not a good choice. For

each byte of sub-key at least 9 templates, corresponding to 9 possible HDs, needs to be built. 2136 bytes of sub-key means 9*2136 cases of templates which is about 20,000 templates. This is a large amount of templates, since for building each template a number of tests should be run and averaged. The number of required tests for each template can be as high as 100,000 runs, and therefore overall it requires 2,000,000,000 number of tests which is to large to be of practical use.

The attack based on such templates will not be strong either. As seen, the switching noise for such a case, where one sub-key byte is addressed, is very high. Therefore the effect of sub-key byte in the power consumption diagram would be buried within the total power consumption. The number of necessary samples for building a template would be really high and even then there is no guarantee that the effect is distinguishable. Rather than building templates for each byte alone, one might think to consider all sub-keys as a whole (containing 2136 bytes), or at least each sub-key set (of at most 251 bytes) for building templates. But then the number of templates will be even larger. For a sub-key of length p, the number of possible cases would be 2^{8*p}, which can be a really high number. For example if $p = 251$ then this number is about 10^{600} of cases of required templates. For the case where all the sub-keys are the target as whole, the result of course is worse.

Thus the only remaining and real options are the super-keys and the final key. As mentioned in section 2.4, an Identical Experimental Device, is required for a Template Attack. This device as mentioned, has the same characteristics of the Device Under Attack, with this exception that, it does not include the secret key, and in fact the sub-keys and other internal registers can be programmed to the values that attacker desires. In the case of Whitenoise, the identical device can also be used to monitor the values of internal registers including super-keys and the final-key. Therefore by monitoring these values, she can observe when in the encryption process the same values of a certain super-key in the FIFO, or final key, are being generated, and use the related power traces to build templates.

This however is not enough for performing a successful attack, and if an Identical Experimental Device is used, there is no guarantee that the sub-keys are identical to the sub-keys in DUA. For this reason, the attacker needs to run a variety of tests for different sub-keys, when building templates for super-keys or final key. The high number of possible sub-key bytes suggests a very high number of runs and samples for the templates is required. To avoid this problem, there is a need to be able to separate the portion of consumed power related to shift-registers from the rest of the

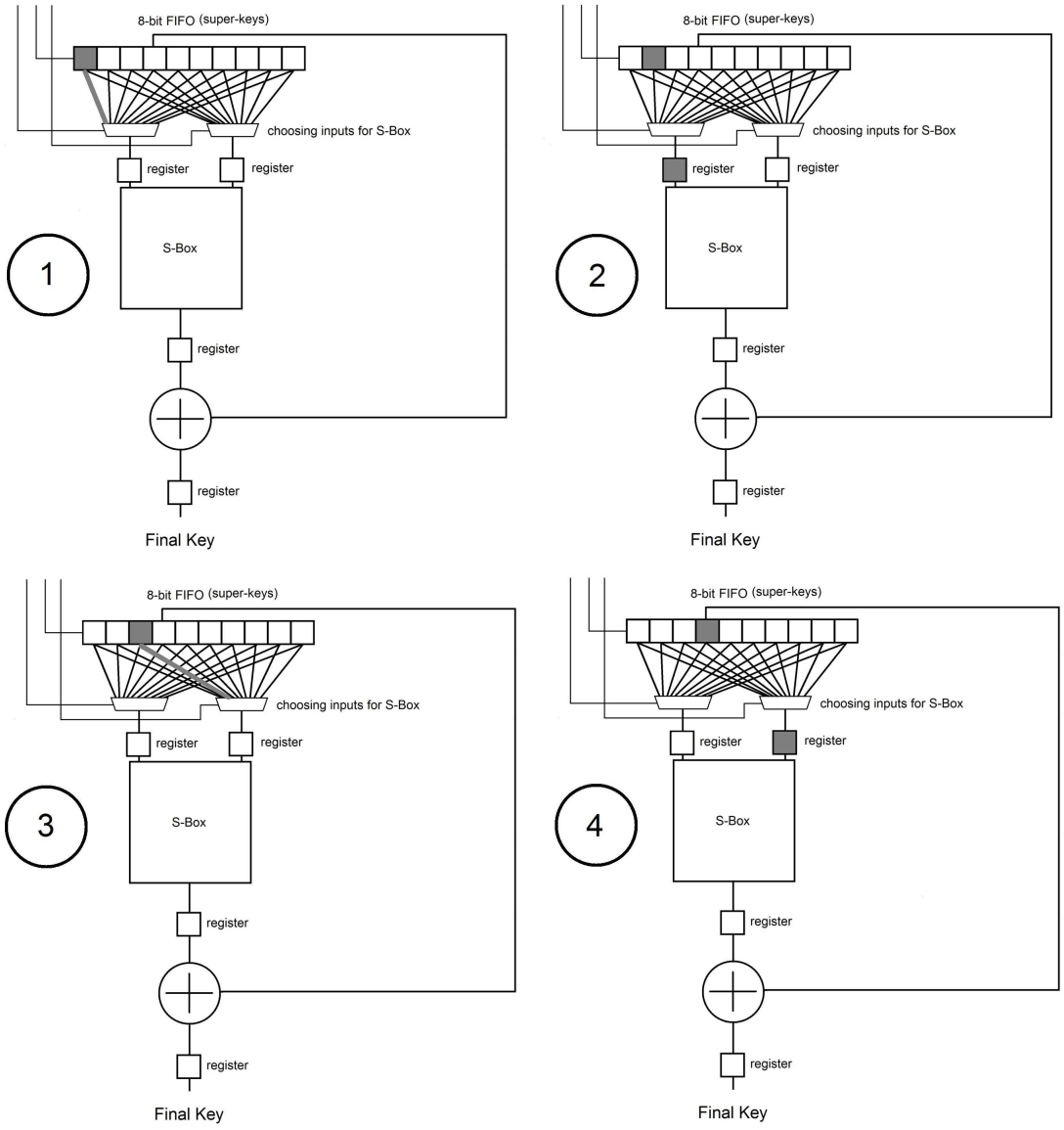


Figure 3.3: Propagation of a super-key in a few consecutive cycles

power. In this project, a method was indeed found to separate these two portions of consumed power and it would be discussed in more detail later. But for now, let's see if the Template Attack is applicable if these two portions of power are separable.

As mentioned, the attacker has 14 cycles of related samples for each trace she captures. These 14 cycles are from the moment that a super-key is generated, until it goes through the FIFO and rest of the circuit and finally flushed out of FIFO. Having the effect of shift-registers and sub-keys removed from the power consumption, the attacker has now access to the remaining part, which is the effect of registers of pipeline and the FIFO. Although this might sound promising, there is another factor here that causes problems. Figure 3.3 shows the propagation of a certain super-key through the circuit. In specific cycles the super-key value becomes one of the inputs of the S-Box, and thus participates in a non-linear operation performed by S-Box. There are two of such cycles, and these are in fact the cycles that she hopes would appear in the built templates as distinctive peaks, and help her to find the secret information.

But the problem is that for the remaining cycles, the S-Box operation is not off, and it is performing a non-linear operation on other independent values. This non-linear operation, causes spikes to appear on the other cycles of the averaged power traces too, where if the S-Box wasn't operating, would show a linear pattern, quite distinguishable from the target cycles.

In summary, and as it is concluded from available literature [7, 15], a Template Attack seems very weak against this implementation of Whitenoise. In most of the designs discussed in literature, the value that is the target of the attack, is involved in a sequence of operations rather than the case here, in which only two cycles partially include the effect of target value. Also, in none of the published work such a high level of parallelism is seen. Here both in the cycles in which the target value goes through a non-linear function, and the cycles that it is just being shifted, there many other linear and non-linear operations occurring in parallel.

The number of required samples also seems to be much higher too. For example in [15] a design is mentioned which includes series of operation and one of these operations, the S-Box namely, is performed on all of the key bytes using 16 S-Boxes in parallel. Even for this case when the design is still sequential, has low parallelism (only 16 bytes in parallel are updated simultaneously) and most of the switching noise is of type 1, 100,000 samples are used to build each of the templates.

Thus Template Attacks don't really seem to be effective against Whitenoise or

at the very least such attacks are weak. A scenario specific for Whitenoise was developed during this project and will be discussed in next section. It uses some special characteristics of Whitenoise to attack its implementation.

3.4 A new proposal for attacking Whitenoise

The idea behind this attack scenario is that, although the process of generating super-keys by adding the output bytes of shift-registers containing the sub-keys has a very long repeating period, the power trace of such a process includes the effects of periodic patterns with smaller periods. Thus, if one can find a way to separate those patterns from the total trace, she is able to perform an attack against the implementation. This would be clarified in the rest of the discussion.

To start, some power model needs to be used, and as usual, Hamming Distance (HD) is what is used here, which best describes the activity within registers. Let's denote $hd_i(x,t)$, as the HD value of the i^{th} shift-register, at its x^{th} byte, at t^{th} cycle of the encryption process, with regard to its left neighbour byte in the same shift-register. Since, as mentioned before, power consumption of a register in every clock cycle, is proportional to number of changes in the state of the register, $hd_i(x,t)$ denotes the consumed power of the x^{th} byte of the i^{th} shift-register, at the t^{th} cycle.

At every clock cycle and in every shift-register, there is one right shift operation for all of the bytes, and the $(p_i - 1)^{th}$ byte will be fed to the 0^{th} byte. So it can be said that:

$$hd_i(x,t) = \left\{ \begin{array}{ll} hd_i(p_i - 1, t-1) & \text{if } x = 0 \\ hd_i(x-1, t-1) & \text{if } 0 < x < 255 \end{array} \right\} \tag{3.4}$$

The function hd_i is thus periodic with the period of p_i for all values of $t \geq 0$, and while x is constant. This can be easily confirmed with simple observations. After p_i cycles, the state of the shift-register is exactly as it was originally, and thus hd_i is periodic for all $t \geq 0$, in any of the bytes of the shift-register.

On the other hand the total power consumption of the i^{th} shift-register at the t^{th} cycle, is equal to the summation of the power consumption of all of its bytes. In other words, the function $g_i(t)$ defined as:

$$g_i(t) = hd_i(0,t) + hd_i(1,t) + ... + hd_i(255,t) \tag{3.5}$$

is proportional to the total power consumption of the i^{th} shift-register at the t^{th} cycle. Since $hd_i(x,t)$ is periodic for all $t \geq 0$, this new function is also periodic, and its period is exactly the same as of the period hd_i. If $hd_i(0,t)$, $hd_i(1,t)$, ..., $hd_i(p_i-1,t)$ were only to be considered, the function would actually be constant, since the first p_i registers form a loop. In a loop of registers, no matter how many times it is shifted, values and their order do not change, and thus the summation of HDs would remain constant. However this is not true for the next values, namely $hd_i(p_i,t)$, $hd_i(p_{i+1},t)$, ..., $hd_i(255,t)$, and those are actually the values that form the changing and periodic part of the $g_i(t)$. These two parts, the periodic and the non-periodic part of $g_i(t)$, are shown in figure 3.4. This figure shows a few consecutive cycles of shift operation in a shift-register, where for simplicity, the length of the shift-register is assumed to be 5, and p_i is assumed to be 3. In summary $g_i(t)$ can be divided in two sections as:

$$g_i(t) = g'_i(t) + C_i \tag{3.6}$$

in which g'_i denotes the periodic part, and C_i is the constant part.

Knowing that the power consumption of each shift-register is periodic, a new attack scenario can now be designed. Assume that $P(t)$ corresponds to the total power, that is consumed in all of the bytes of all shift-registers. $P(t)$ is thus proportional to summation of $g_i(t)$s that is:

$$P(t) \approx \sum g'_i(t) + \sum C_i \tag{3.7}$$

Now assume $P(t)$ is sampled for a number of $2m_{10}p_{10}$ consecutive cycles, where p_{10} is the length of the longest sub-key, and m_{10} is some coefficient which would be discussed later. For all of the sampling points in the range of $0 \leq t < 2m_{10}p_{10}$, a new function then is defined as:

$$P_{10}(t) = P(t + m_{10}p_{10}) - P(t) \tag{3.8}$$

For this new function it can be said that:

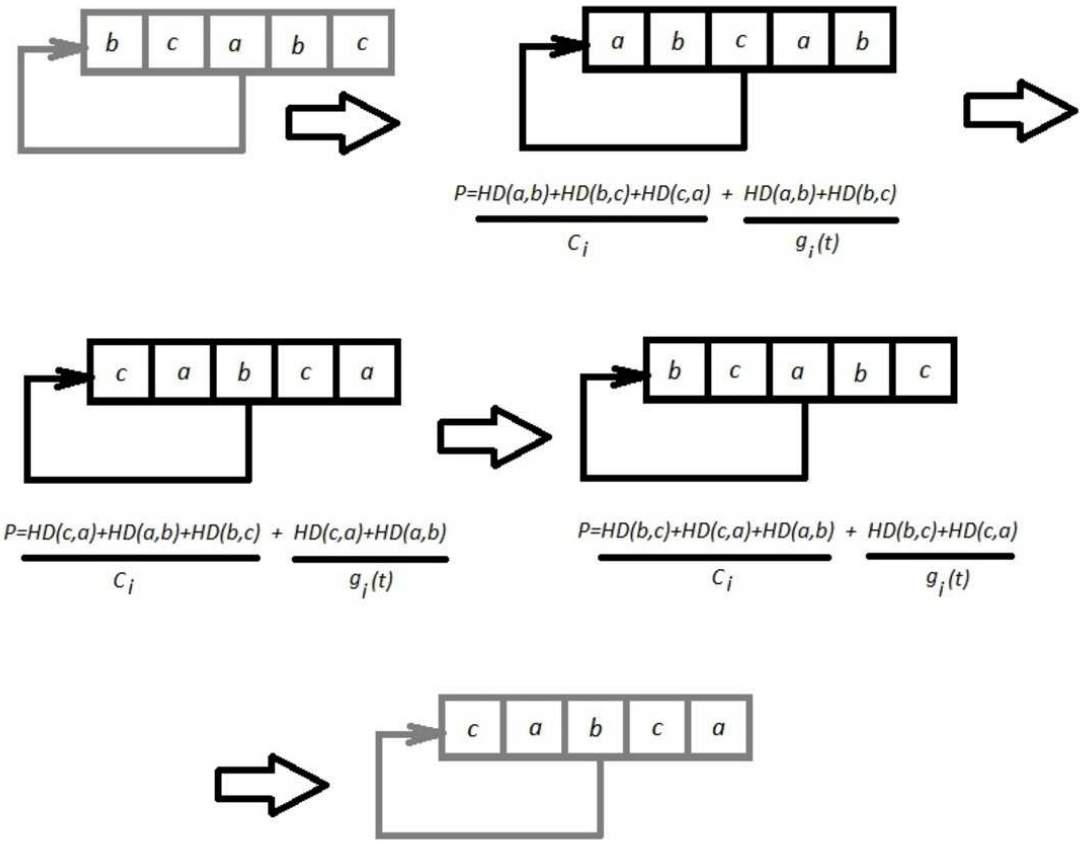


Figure 3.4: HDs for a few consecutive cycles of shift operations in a shift-register

$$\begin{aligned} P_{10}(t) &\approx \sum g'_i(t + m_{10}p_{10}) + \sum C_i - \sum g'_i(t) - \sum C_i \\ &= g'_{10}(t + m_{10}p_{10}) - g'_{10}(t) + \sum_{i=0}^{9} g'_i(t + m_{10}p_{10}) - \sum_{i=0}^{9} g'_i(t) \\ &= 0 + \sum_{i=0}^{9} (g'_i(t + m_{10}p_{10}) - g'_i(t)) \end{aligned} \tag{3.9}$$

The equation above shows that the result does not include any effect of the 10th shift-register. The result contains some other parts as $g''_i(t+p_i) = g'_i(t+m_{10}p_{10}) - g'_i(t)$ though, and what is noticeable about these parts is that, although the function hd_i in each part is manipulated, the result is still periodic with the period of p_i. In other words:

$$\begin{aligned} g_i''(t+p_i) &= g_i'(t+p_i+m_{10}p_{10}) - g_i'(t+p_i) \\ &= g_i'(t+m_{10}p_{10}) - g_i'(t) \\ &= g_i''(t) \end{aligned} \tag{3.10}$$

Thus $P_{10}(t)$, similar to $P(t)$, is a summation of a number of periodic functions, with the difference that the effect of 10th shift-register has been removed. The same idea can now be applied to P_{10} to remove the effect of the 9th shift-register in it. This is done by subtracting the first m_9p_9 samples, from the second m_9p_9 samples, in the remaining $m_{10}p_{10}$ samples ($P_9(t) = P_{10}(t+m_9p_9) - P_{10}(t)$). The only requirement here, is $2m_9p_9$ samples actually do exist in the $m_{10}p_{10}$ cycles from previous round of subtraction. In other words:

$$2m_9p_9 \leq m_{10}p_{10} \tag{3.11}$$

By performing this process repeatedly, it gets to a point that only the effect of the first shift-register remains in the obtained function, namely $P_2(t)$. This function would be a linear transform of $g_1'(t)$. Since the values of function $P_2(t)$ are known, a system of linear equations, with p_1 unknowns, and p_1 equations, can be formed to derive the values of $g_1'(t)$ for $0 \leq t < p_1$.

Now, by knowing the values of $g_1'(t)$, and $P_3(t)$ which was previously computed during the subtractions, another system of linear equations can be formed to deduce the values of $g_2'(t)$. This reverse process can be repeated up to the point that values of $g_{10}'(t)$ are found. Since and as mentioned, $g_i'(t)$ itself, is a linear function of the HDs of the i^{th} shift-register, the computed values in the system of equations can be used now to obtain the HDs of shift-registers.

There is one requirement though, as equation 3.11 denotes, which should hold for all of the m_i values and it is as follows:

$$
\begin{aligned}
2m_9p_9 &\leq m_{10}p_{10} \\
2m_8p_8 &\leq m_9p_9 \\
2m_7p_7 &\leq m_8p_8 \\
&... \\
2m_1p_1 &\leq m_3p_3 \\
&\text{and} \\
p_1 &\leq m_2p_2
\end{aligned}
\tag{3.12}
$$

These inequalities show the minimum required amount of samples for applying the attack scenario. By choosing the appropriate number of samples, the attacker ensures that the sampled data include enough cycles for all of the consecutive splits and subtractions. In the worst case, where the largest prime numbers are chosen as the length of sub-keys $(251, 241, 239, 233, 211, 199, 197, 193, 191$ and $181)$, with simple calculations, starting from m_2, and rounding up the results, m_{10} would be 219. Therefore, the total number of cycles for performing the attack scenario in this case is about $110,000$ cycles. In a slow encryption system with the clock frequency of $1MHz$, this means only 0.1 second of run time which is a totally reasonable time for performing an attack. The number of samples required can even be lower, and as low as 3162 samples, for the case that smallest prime numbers are chosen as the length of the sub-keys. In this case the sampling time is in the range of several milliseconds.

It should be noted that in choosing m_is the attacker should be careful to not accidentally choose some value for a m_i which is a multiplication of the remaining prime numbers (p_1, ..., p_{i-1}). If that happens, the result of the subtraction of that stage not only excludes the effect of i^{th} shift-register, but it also excludes the effect of some other shift-register with a shorter length too. Thus in the reverse process, when systems of linear equations are built, for that shift-register for which its effect was removed accidentally, the number of equations would not be enough to determine the values.

There is an easy solution to this problem however, and that consists of choosing $m_i + 1$ or $m_i + 2$ rather than some value for m_i which is a multiple of smaller prime

numbers. This way the result is not a multiple of that smaller prime number, and the problem can be avoided.

The scenario proposed here is capable of finding HDs of bytes of sub-keys. However some questions still remain to be answered. Questions such as how, in the first place, the power consumption of the shift-registers can be separated from the total power consumption? Or how knowing the HDs of bytes of sub-keys, would help in finding the actual values of them? What are the measurement requirements of such an attack scenario, and etc. These questions would be addressed in the next section.

3.5 Requirements, Performance and Improvements to the proposed Attack Scenario

The first question unanswered in the previous section is how to separate the power consumption of shift-registers from total power consumption. This problem was addressed before in section 3.3 too when Template Attacks were being discussed, and it was mentioned that performing any Template Attack has this as a requirement. In section 3.2 it was mentioned that Whitenoise implementation provides an input pin as shift-enable, and also an internal control register for mode of the operation of circuit. For Whitenoise, no matter if it is implemented in FPGA as is the case of this project, or it is implemented as an ASIC and provided as a chip, the mode register and shift-enable pin seem to be a necessary part of implementation. If the user is going to be able to reprogram the shift-registers with different sub-keys, then there should be a way to keep the encryption process idle while the sub-keys are being programmed. Not having this feature means the implementation includes only one constant set of sub-keys, and this is a tight restriction on an algorithm such as Whitenoise, which in theory has a lot more flexibility. But this same feature makes it possible to separate power consumption of shift-registers from the consumed power of rest of the circuit.

For any Device Under Attack, if the attacker has access to the device, she can switch the mode to program mode, and, without any further programming, allow sub-keys to shift and toggle to obtain their power consumption. She even does not need to have access to the IOs, and monitoring the power consumption in programming mode is enough to obtain the samples of power corresponding to shift-registers.

If she, on the other hand, wants to perform a Template Attack, which as mentioned

requires the samples of consumed power in the FIFOs and pipeline registers, she can again capture the trace of shift-registers alone, then reset the circuit, capture the trace of the whole circuit, and subtract the results. The remaining part of the power consumption is what corresponds to the other blocks (pipeline registers and FIFO), and is what she desires to determine.

The double mode of functionality in Whitenoise seems like a considerable weakness after all, and requires designer's considerations. As said, simply removing the reprogramming feature does not seem to be a good solution, so the implementation itself requires improvements.

The next question which is perhaps the most important one is what to do with the HDs after they have been determined? In Differential Power Analysis Attacks the attacker doesn't face this problem. What she does in a DPA is for each guess of the key, and various inputs, she computes the output values of some function as the target of attack. She then uses a model to transform the obtained results to hypothetical model, which correlates with the power consumption. So although for example HD model is used, but the model is just used to do the transformation of obtained results of the target function, to a better mean of comparison. Each guess of the key has its own hypothetical model. So a matching hypothetical model determines the correct key and not the HD of it. Therefore in DPAs, HD is not a problem and the value of the key is what is found eventually. Template Attacks also have a way to go around this problem, which eventually reveals the value of the key rather than the HD of it (The topic is not discussed here anymore to not confuse the reader, but for further details see [22]).

In the proposed attack scenario, however, the situation is obviously different. Here the final obtained values are HDs, which are not part of some hypothetical model. By having the HDs alone, the search space will be reduced. But since the total search space is enormous, this new knowledge would only reduce its size to a still huge space, not searchable by brute force methods.

To solve this problem let's first see what the unknown parameters are, and what is obtained about them. The number of sub-keys in their worst case is 2136 bytes(largest prime numbers). Therefore, there are about 2000*8 unknown variables, as the bits of all bytes of all the sub-keys. So far by knowing HDs, the attacker has 2000 equations about these values. After all, each HD of a pair of registers can be written as a equation, as follows:

$$HD(A,B) = A_1 + B_1 - 2A_1.B_1 + A_2 + B_2 - 2A_2.B_2 + ... + A_8 + B_8 - 2A_8.B_8 \quad (3.13)$$

So the attacker needs about 14,000 more equations. And there is in fact a way to obtain the necessary remaining equations. The attacker can reset the circuit so the intermediate registers would be initially set to 0. She switches the device to run mode, and she toggles the clock for only one cycle. The observed power at the edge of the clock denotes the power consumption regarding shift-registers, plus the power consumption of the first register in the pipeline which now holds the value of a super-key. The attacker already knows the HDs of bytes of shift-registers, so she can exclude the effect of them, and extract the power consumption of the first register of pipeline. Since this register was including 0 after resetting the circuit, the HD of it is simply its Hamming Weight (HW), which is the number of bits that are equal to one. The HW of a register can be written as:

$$HW(A) = A_1 + A_2 + ... + A_8 \quad (3.14)$$

On the other hand, the value of the super-key is calculated as:

$$S = A^1 \ XOR \ A^2 \ XOR \ A^3 ... XOR \ A^{10} \quad (3.15)$$

in which A^i are the last byte of the sub-keys, creating the super-key. This equation can be rewritten as:

$$\begin{aligned}
S_1 &= A_1^1 + ... + A_1^{10} - 2A_1^1.A_1^2 - ... - 2A_1^9.A_1^{10} + 4A_1^1.A_1^2.A_1^3 + ... - 1024A_1^1.A_1^2.A_1^3....A_1^{10} \\
S_2 &= A_2^1 + ... + A_2^{10} - 2A_2^1.A_2^2 - ... - 2A_2^9.A_2^{10} + 4A_2^1.A_2^2.A_2^3 + ... - 1024A_2^1.A_2^2.A_2^3....A_2^{10} \\
&... \\
S_8 &= A_8^1 + ... + A_8^{10} - 2A_8^1.A_8^2 - ... - 2A_8^9.A_8^{10} + 4A_8^1.A_8^2.A_8^3 + ... - 1024A_8^1.A_8^2.A_8^3....A_8^{10}
\end{aligned} \quad (3.16)$$

In which A_i^j denotes the i^{th} bit of the output byte A^j. These new equations change the form of the function to an algebraic description. There might be easier or better ways to do this too, but this one is the most straight forward method. By using equations 3.14 and 3.16, a new equation is obtained which can be added to the

system of equations. This process can be done as many times as desired and all the required equations can thus be obtained. The number of necessary tests to apply is about 14,000 (corresponding to 14,000 left unknown variables), which comparing to some template attacks that require 100,000 tests, is reasonable.

About the complexity level of system of equations itself, first it should be noted that the unknown variables are either 1 or 0, thus solving the system is a lot easier. Also it should be mentioned that the system of equations is polynomial. Alone, any equation obtained by combining 3.14 and 3.16 (and by performing one test on the implementation), is an equation of degree 10. Any such equation when added to the system of equations, increments the degree of the whole system by 1, but still the result remains polynomial. In the worst case (depending on the unknowns values), the system is a polynomial system of degree 14,000. Since the system is polynomial, the required computation time is also polynomial, which in terms of computer science theory, makes it feasible and fast to perform.

Now let's see how increasing the number of sub-keys, or increasing the size of each sub-key, would affect the computation time. For example assume that the number of sub-keys is doubled and there are 20 sub-keys instead of 10 of them. If the length of the new added sub-keys is still between 2 and 255, this means in worst case and for the 20 largest prime numbers smaller than 255, the number of unknown variables increases to about 3500 bytes (about 28,000 bits). This means the number of extra required tests, instead of 14,000, is about $28,000 - 3,500 = 24,500$ (total number of unknown variables minus the number of equations derived from first stage of attack and computing Hamming Distances). Thus the degree of the polynomial system is doubled, which does not much affect the complexity.

Another option is to increase the size of each sub-key. This can be done in two ways: increasing the length of sub-keys, or increasing the size of each byte of each sub-key, to a word consisting of a few bytes. For both of these two cases however, the result is the same as the case when the number of sub-keys was increased, and the polynomial degree of the system of equations, at worst case, is doubled. As a result of this discussion, increasing the number of sub-keys, or the size of them, does not much help in making the implementation more resistant, and more consideration is required to improve the implementation.

The next question is about the lengths of the sub-keys. What wasn't mentioned in the proposed scenario and was implicitly assumed, is that the lengths of the sub-keys are known to the attacker. The lengths of sub-keys are actually part of the block that

generates the bytes of sub-keys, and as mentioned before that section of the code is not available in the implementation being discussed.

So the question is how the attacker can obtain these lengths? The answer to this question is actually simple. Knowing the fact that the total power consumption of shift-registers is an addition of some periodic functions, the attacker can perform a Fourier transform on the sampled results. This transformed function would reveal the harmonics and major frequencies involved in the creation of the power trace, namely the periodic shift-registers. The shortest possible prime number is 2 and the largest is 251 with about 50 other possibilities for prime numbers between.

In a slow device with a clock of $1MHz$ the spectral components will have a frequency that ranges from $4kHz$ to $2MHz$. Within 1 second of sampling time from such a device, 1,000,000 cycles would be obtained, which provide a frequency resolution of $1Hz$, and maximum measurable frequency of 2MHz. This is more than enough to detect spikes regarding the harmonics and thus identifying the length of sub-keys. But still if the attacker wants, she can also apply one extra step, and perform linear filtering around the frequencies corresponding to the possible prime numbers, to obtain better results. There are only about 50 possible prime numbers between 2 and 251 after all, and applying this process would not be difficult.

The next question is about requirements in the measurement equipment. There are two major requirements which will be discussed. These requirements are not as strictly necessary in common attacks, as they are in the proposed attack scenario. The reason is common attacks such as DPAs or Template Attacks, a vector of hypothetical model, or a template which is again a vector obtained from averaging the samples, is the mean of comparison, to confirm if a guess is correct or not. In the proposed scenario however, every sample is measured independently, and used directly in the computations, and this suggests need of having a setup with special requirements.

The first of those requirements is the precision. The proposed scenario requires measurements, as small as $HD = 1$, in the consumed power. As will be seen in next chapter about the board built in this project, this requirement is not easily achievable. In the designed board presented in the next chapter, it would be seen that in the best case, activity of hundreds of registers can be measured, and measuring beyond this margin requires special equipment. Therefore, based on the application, the designers may or may not, want to consider the proposed scenario as a threat. If the implementation is used for common purpose applications, the proposed scenario might not be an issue, since the attacker most likely will not go that far to break an

implementation. But if its application is of greater importance, then there might be more serious efforts against the implementation, and attacker may have access to an advanced equipment too.

There is another requirement that the equipment should have. It should be *linear* and *stationary*. What for is meant by linear measurement setup is, the consumed power of two pieces of circuit combined, should be equal to the addition of their consumed power alone. If this condition doesn't hold, then there is no way to correlate the observed results to HDs, and the proposed attack becomes useless. Measurement setup should also be stationary, meaning for a same measurement in different times the result should be the same. In the next chapter these two concepts would be investigated in the designed board, and it would be seen that they do in fact exist. This means that such expectations from a setup are not unrealistic, and achievable.

3.6 Conclusions

As a short summary of what is discussed and the final conclusions, it was mentioned that Whitenoise seems resistant to common attacks. The proposed scenario along with its improvement can break the implementation and find the sub-keys, but since it requires a large number of computations and tests, and it also requires advanced measurement capabilities, Whitenoise seems a fairly strong stream-cipher. The proposal and its improvement can merely be seen as a theoretical proof that the implementation is vulnerable, and as mentioned, how seriously they are going to be taken is dependant to the application of Whitenoise, and the designer. The more serious the application is, the more likely that the attacker is willing to use high precision equipment.

As noted, the main weakness rises from separable portions of power consumption, which is a result of the two functionality modes the device has. The reset pin, which again as many other stream-ciphers is an inseparable part of the implementation, is another source of the problem, since it was used in the improvement of the proposed scenario to obtain more equations. That the design is highly parallel, or the sub-keys or even their length are unknown, would not help the designer in hiding the secret information. More consideration is required to resolve this weakness of implementation.

Chapter 4

Measurement Setup

In this chapter another part of this project, which was building a measurement setup, will be discussed. First the requirements for such a setup to sample traces of power is studied, and after that the sampling board, which is the main of part the setup, would be reviewed. Results of actual experiments would then be presented, and at the end they would be discussed as the conclusion of experiments.

4.1 Power Sampling Requirements

The easiest way for sampling the power of a digital device, is to put some resistor between the ground of the supply and the ground of the device, and measure the voltage across the resistor. The consumed power in the device is then the total consumed power in the supply minus the power, consumed in the resistor, and is proportional to the voltage drop across the resistor and is computed as: $(Vcc.V(t) - V(t)^2)/R$. However there are a few problems in applying this method in practice.

The first problem is that using a resistor between the ground of the device and the global ground would affect the functionality of the circuit. For example in a FPGA which has a power supply of $1.2v$, and the current can go as high as $500mA$, a resistor of 1Ω would cause a voltage drop of $.5v$ across the resistor, causing the power supply to be appeared on the device as $.7v$. This voltage drop would cause the FPGA to be turned off and thus the resistor would interfere with normal functionality of circuit. Therefore the resistor that is going to be used should be of a small value, namely smaller than $200m\Omega$.

This causes the 2nd problem and that is for such a small resistor the small changes

of voltage drop across the resistor are difficult to detect. For a $200m\Omega$ resistor, and a nominal current of $500mA$, only $.1v$ voltage drop would be seen across the resistor which does not interfere with the functionality of circuit. But this voltage drop is the maximum voltage drop of the circuit, and the effect of small logic blocks would be small and hard to detect.

For avoiding this later problem some sort of amplification is necessary. And because the functionality of the circuit shouldn't be distorted, the amplifier should not have impact on the resistance value. For this reason OPAMPs[1] are chosen here. The amplification also needs to be done in high frequency so small changes in the power can be captured, and thus high frequency requirements should also be considered. These include using different kinds of capacitors, ferrite beads and other things which would be discussed. In this project, it was decided to build a PCB[2] which includes the sampling and amplification parts, along with the digital section where FPGA is embedded. The board along with FPGA programming tools, a oscilloscope for capturing the samples, and a three-end power supply, are what make the measurement setup. This setup is shown in appendix A. In the following subsection the design of the board is reviewed in more detail.

4.2 High Frequency Power Sampling Board

As mentioned before, for this part it was decided to use a high performance FPGA. The FPGA used was from Spartan 6 family of Xilinx (XC6SLX150T-3FGG676). For this category of FPGAs, as many other BGA components, having a multilayer board is necessary, to have proper routing between the small pads. Also a multilayer board can provide ground or supply layers, which besides the fact that they make routing easier, they also provides better and wider connections, reduce the parasitic inductance, and act as shields to electromagnetic emissions of environment too.

For the BGA itself after a couple of trials in building the PCB, it was decided to use a BGA socket, so the FPGA can be detached from the board in case the PCB is faulty, or the FPGA is burned. There are many other requirements about designing the digital part of the PCB, and they were all considered. Some of these requirements include: DRCs[3] about pad sizes, traces and VIAs provided by Xilinx

[1] Operational Amplifiers
[2] Printed Circuit Board
[3] Design Rule Checks

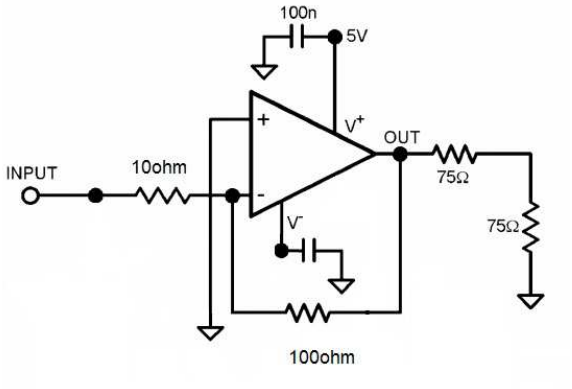


Figure 4.1: Inverting Amplifier

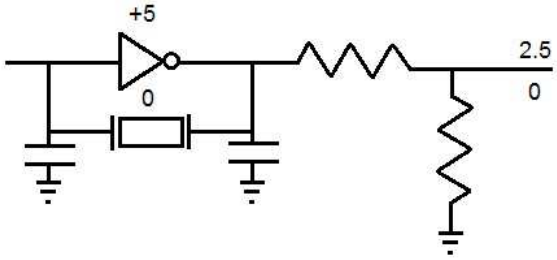


Figure 4.2: Circuit of the Oscillator

[32, 30, 31], recommendations about DRCs provided by the company that does the fabrication process, rules to follow in placing the coupling capacitors, and etc. Besides these all, the proper pin assignment of the FPGA is also important, so that it can be programmed and function correctly. In this design JTAG standard was used to program the FPGA. This standard provides a fast way of programming the FPGA using the parallel port of the PC and a JTAG cable, and it doesn't require any extra device such as PROM or a microprocessor for the programming process.

For designing the analogue part as said OPAMPs were used. A sample circuit to have differential amplification using OPAMPs is shown in figure 4.1. The input (FPGA Ground) is amplified differentially regarding the Global GND, so besides being amplified, the common noise is reduced too. Capacitors of different values are also used in the plus and minus supply, to have the supplies steady and not affecting the functionality of the amplifier.

Another part which plays an important role is the oscillator. The oscillator is build using logical CMOS invert gates and a quartz crystal. Its schematic is provided in figure 4.2. In the early trials of designing the PCB, the oscillator showed to have

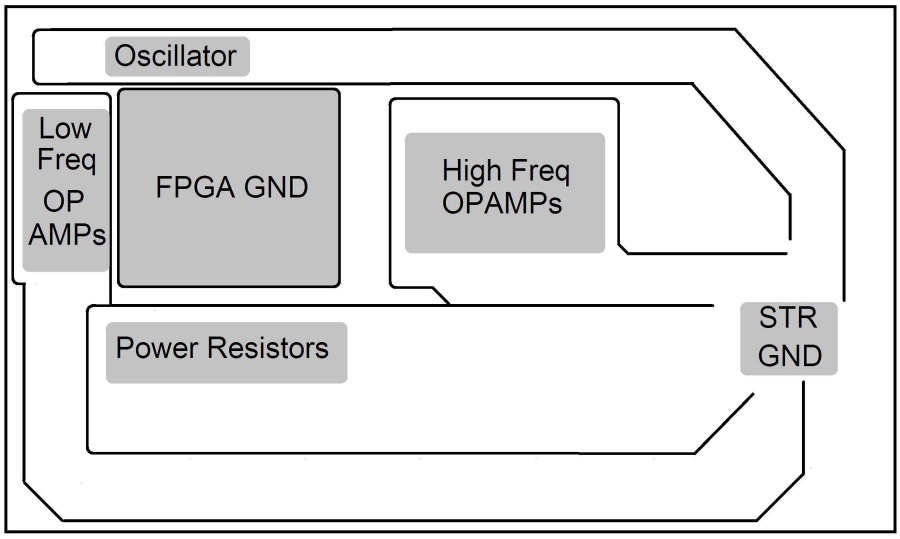


Figure 4.3: Split GND Layer and Star GND

significant effect on the +5 power supply which is shared with the OPAMPs, and it was causing ripples at the edges of the clock. Thus it was decided to use ferrite beads for power supplies to have more stable supplies.

Another important part of the design is what is embedded as the *Star Ground*. What it means to have a star ground in principle, is to have all the grounds from different parts of the circuit split, and meet, at the same point on the board. So while the effect of each one is split, the number of loops formed from supplies to the board is reduced to minimum, and therefore parasitic inductance is minimized. Appendix B shows the different layers of the PCB, including the ground layer. The slip sections which meet at one single point are visible in that layer. This is also shown in a simpler form in figure 4.3. The single point that the different sections end up in, is where the ground wires from the supplies would be soldered to.

There is one other phenomenon related to inductance of the circuit, specially in high frequencies. In a circuit the electrons always take the shortest path between two connected pins. In most cases this is a straight line between the two ends, but if the straight line is not available, and for example some split line prohibits a straight connection, then the shortest possible path close to that would be chosen instead. In this case, whatever the pattern of connection from supply to the pin is, it is best to have the same pattern on the ground layer too, so the surface of the created loop is minimized to vertical axis, and the parasitic inductance is reduced to the minimum

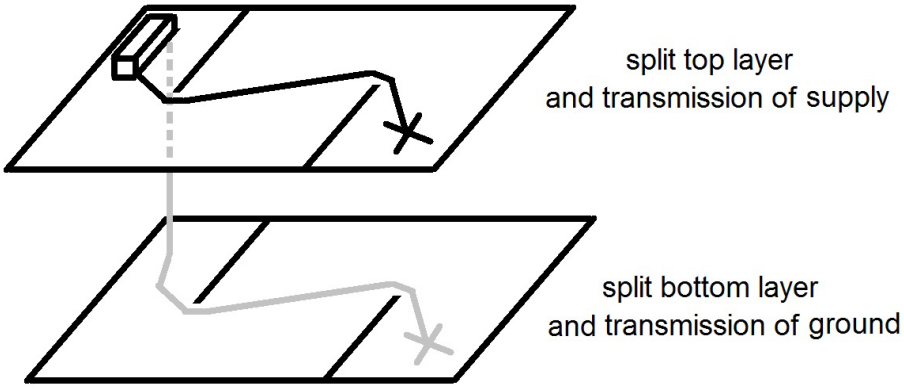


Figure 4.4: Reducing the Inductance

as well. Figure 4.4 illustrates this fact and as figure shows, in the ground layer (or any other related layer), there is the mirror of the pattern that exists in the supply layer to minimize the inductance. This phenomenon is considered in the designed PCB for all of the components, and as you can see, split lines in different layers have almost the same pattern.

Designing a board for such a FPGA and the high frequency components needs many other careful considerations and it takes a lot from designing the PCB of the board, to implement a HDL code on the FPGA and have it running. The board went through 3 trials before the 4th and final version was designed and fabricated. The FPGA and many other components were intentionally chosen among the advanced ones so the board will be multi-purpose and usable for future projects too. Tests and experiments then were done on the board, which will be discussed in the next section.

4.3 Results and Conclusions on Measurements

A number of tests are applied to discover the properties of the board and the Spartan 6 FPGA. In early examinations, the amplification section appeared to be troublesome, and that part of circuit had to be rebuilt and tested separately from the board, to understand and solve the issues. Therefore by the time that the main tests were being applied, the fourth version of board, which included the correct amplification circuit, was still under fabrication. Nevertheless the results which are presented here would be consistent with the case that the amplifier is used. Also at the end of this chapter

a brief discussion, for the case that amplifier exists, is presented.

The tests which are applied on the board can be categorized as follows and will be discussed one by one:

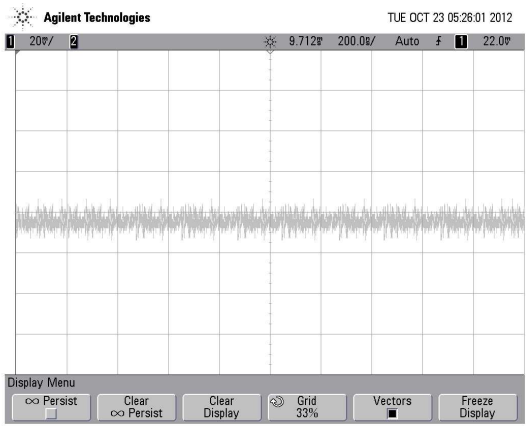
1. Tests to determine the percession in detecting toggling registers.
2. Tests to determine the pattern of consumed power in combinational logic comparing to sequential logic.
3. Tests to confirm the linearity of power measurements.
4. Tests to measure the effect of toggling outputs of FPGA.
5. Tests to identify the effect of global interconnect of the FPGA in consumed power.

First, to measure the power of the toggling registers, shift-RAM blocks of the FPGA were used. These cores are specific blocks with short connection between, which can provide FIFO structure. The FIFOs can be as deep as about 1K bits (1088 bits to be exact), and as wide as 256 bits. The input of FIFOs are fed with all zeroes in one cycle, and all ones in the next, so at each clock cycle maximum number of changes happen in the state of registers. Figure 4.5 shows the resulting power consumption for different number of FIFOs.

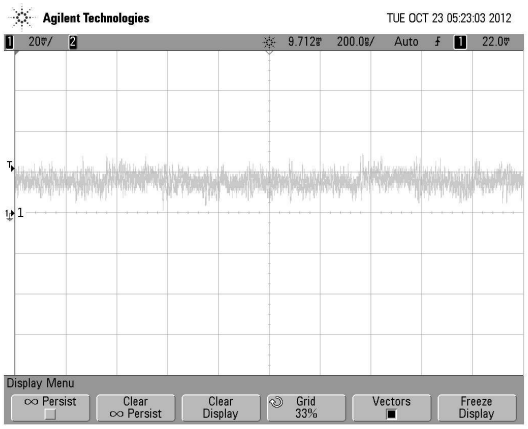
The figure shows a clear improvement in the peak of the signal when the number of FIFOs is increased. Also and as expected from what was discussed in 2.3, the figure shows a lower peak at the negative edge of the clock for each clock cycle. Figure suggests that about $32k$ bits is the limit, below which detection becomes difficult. $16k$ FIFO (not shown here) is still measurable but below that, distinguishing the signal from noise is not possible without amplification. Therefore $16k$ bits seems to be the precision of measurements.

Next, is to detect the power consumption of combinational logic, and to compare it with the result of sequential blocks of FIFOs. It took a while in this part to come up with an effective combinational logic which takes large amount of logic, toggles on every clock cycle, and doesn't contain much sequential logic. Only for such a logic the effect can be claimed to relate to the combinational logic and not the registers. The result was a circuit as presented in figure 4.6.

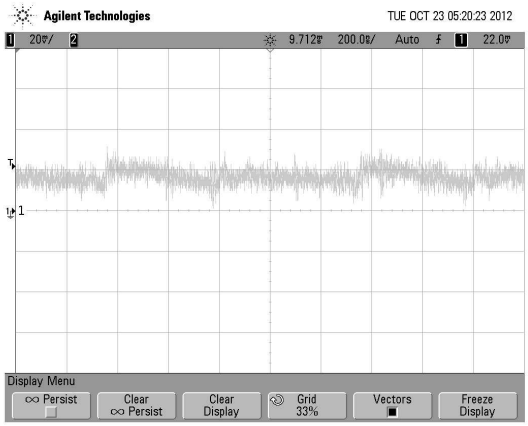
As can be seen from the figure, some input is first fed to a register, namely of 256bit width. The upper half and lower half are then multiplied, and the result



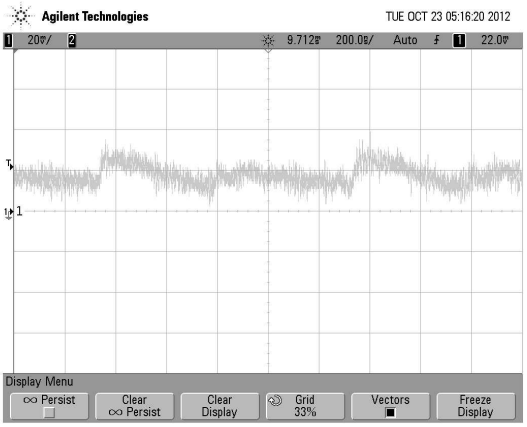
one 32-bit full depth FIFO



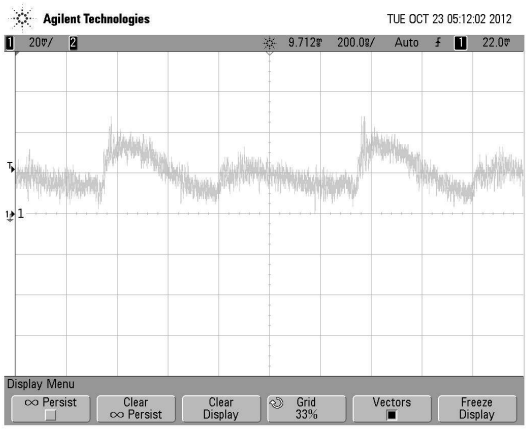
one 64-bit full depth FIFO



one 128-bit full depth FIFO



two 128-bit full depth FIFO



four 128-bit full depth FIFO

Figure 4.5: Power consumption of FIFOs

is again a 256 bit number which is split by half and multiplied again. This process happens a few times in series and the result is fed back to the register. The multipliers are implemented as separate components in design so the compiler does not merge them in the optimization process. Since each multiplier is a large combinational block consisting of so many adders and shifters, it is expected to see a high power consumption. The number of registers in this design is just 256 bits which is way lower than detectable range ($16k$), so the measured power should only be related to the combinational logic and connections, and not the sequential blocks.

In theory the result of the continuous multiplications might converge to a certain value (maybe all zeroes or all ones), and thus in the code it is included that if this occurs, the initial input will be fed to the registers again. Although some random inputs were chosen in the experiment, and they were not checked for convergence before actually applying the test, but amazingly the results confirmed that convergence in fact did happen. Figure 4.7 shows the result for a number of cycles, and it is obvious from the waveform how the result of multiplication is converging to a fixed value, reducing the amount of consumed power.

Figure 4.8 shows the result when output of the scope is zoomed in. Because of various peaks during each period of computation, the result appears as a shaded region. What is interesting in this result though, is that nine lines show more intensity than the other values in this merged pattern of output. Although this can't be claimed as a certain fact, but it is very likely that this nine lines are representing nine Hamming Weight values corresponding to some 8 bit bus lines, or 8 bit multiplication units, used as the cores in implementation of the circuit.

As another note the signal shows a more curvy pattern, with some delay from edge of the clock, rather than what is seen for the FIFOs, which again is expected. It was said in previous chapters that the power consumption of registers are expected

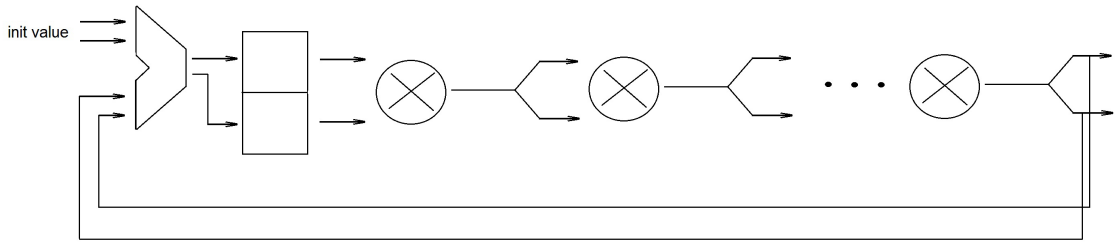


Figure 4.6: Circuit of series of multiplications

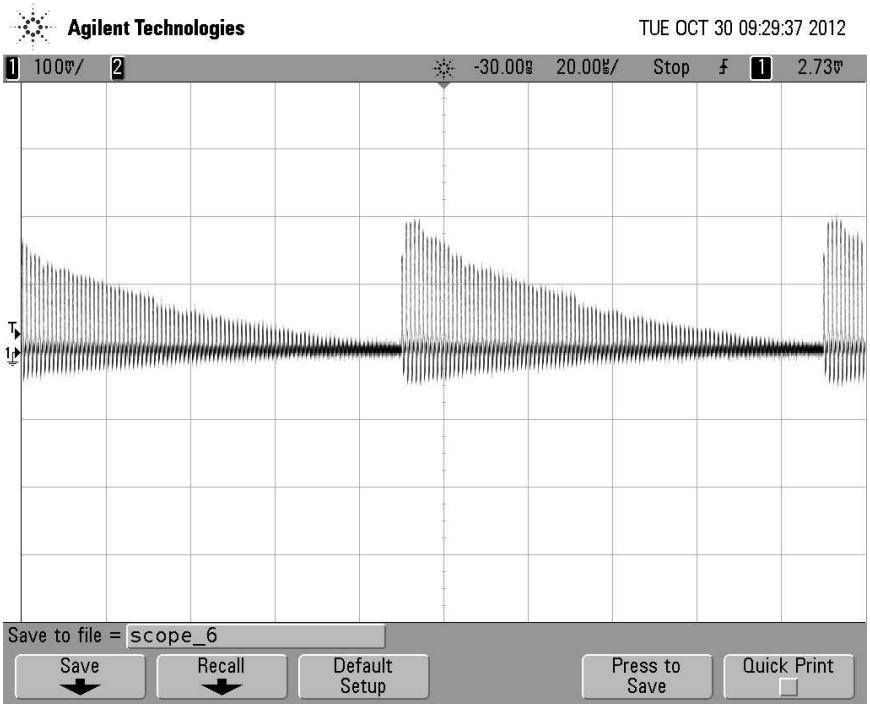


Figure 4.7: Converging output of multiplication circuit

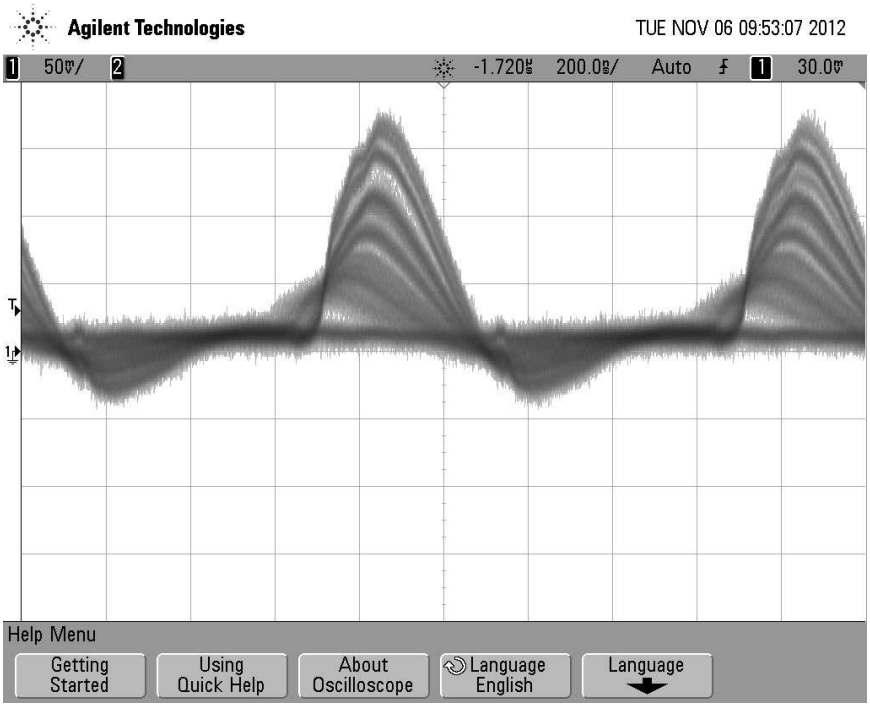


Figure 4.8: One clock cycle in multiplication circuit

to be seen as a peak close to edge of the clock, while the rest of the circuit has a delayed effect, from the edge of the clock, in the power consumption.

As a final note, the amplitude is obviously larger than the peaks of consumed power in the circuit of FIFOs (the scale of the scope is $50mv$ and $100mv$ in figures 4.7 and 4.8, comparing to the $20mv$ scale in figure 4.5), which is again not unexpected. A multiplier of $128 * 128$ bits, if not optimized, is implemented using 128 adders of the same width. In this design a few of such multipliers are used, so even if the multiplier is optimized, still there would be hundreds of adders creating a large combinational circuit.

The next test is about the linearity of the consumed power. This test would verify if the power consumption is linear, meaning that if two blocks of logic are implemented together, their effect would be added too. This test confirms that there is no dominant factor in power consumption, and also the consumed power is proportional to the amount of logic used in a linear way. As said at the end of previous chapter, this is very important property, and if it does not exist, performing an attack would be very difficult, if not impossible.

Figure 4.9 shows the results of power consumption, in the same scale, for the FIFOs, multiplier alone, and the case where these two circuits are merged. It is visible from the figure that linearity does in fact exist. The minimum value of the multiplication circuit, and that of the merged circuit, show this observation better since the diagram is shifted up in the second case, exactly as the amount and shape of the power consumption of FIFOs. Figure 4.10 Also confirms this observation, by comparing the diagram of the multiplication circuit with the merged circuit, when the diagram is zoomed out. It is obvious from the figure how the maximum and minimum value in the later case is increased by the same amount (the peak of consumed power in FIFOs).

The next test is about global interconnect. It is important to know if global interconnect has a significant role in the consumed power. Previous publications [24] mention that global interconnect typically consumes more power than the logic blocks (up to 10 times more). But it is not mentioned what the proportion of global interconnect to the logic cells were in the used circuits in their tests. Thus one wonders if the discussed diagrams in 4.5, and 4.7 to 4.10, were the effect of global interconnect, rather than registers or logic cells. If those are in fact caused by a dominant effect of the global interconnect, then the impact of global interconnect on the design is strong enough to mask the effect of other blocks. So any encryption design can be masked

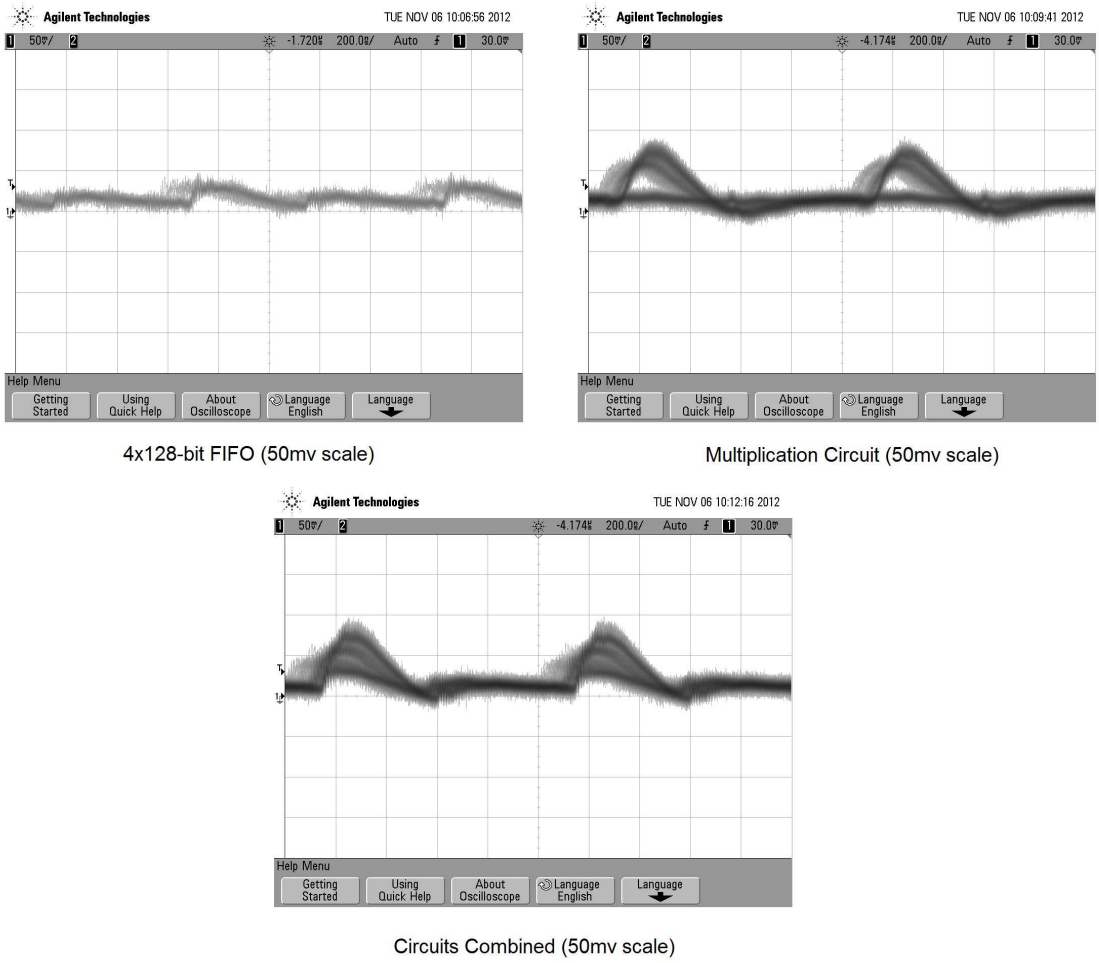


Figure 4.9: Power Consumption of two circuits alone and merged

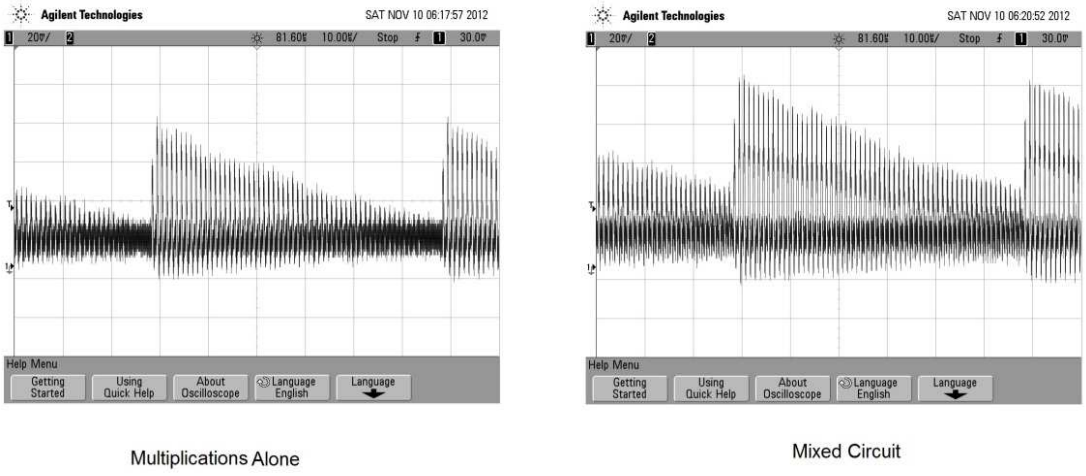


Figure 4.10: Consumed Power of Multiplication Circuit vs. Merged Circuit

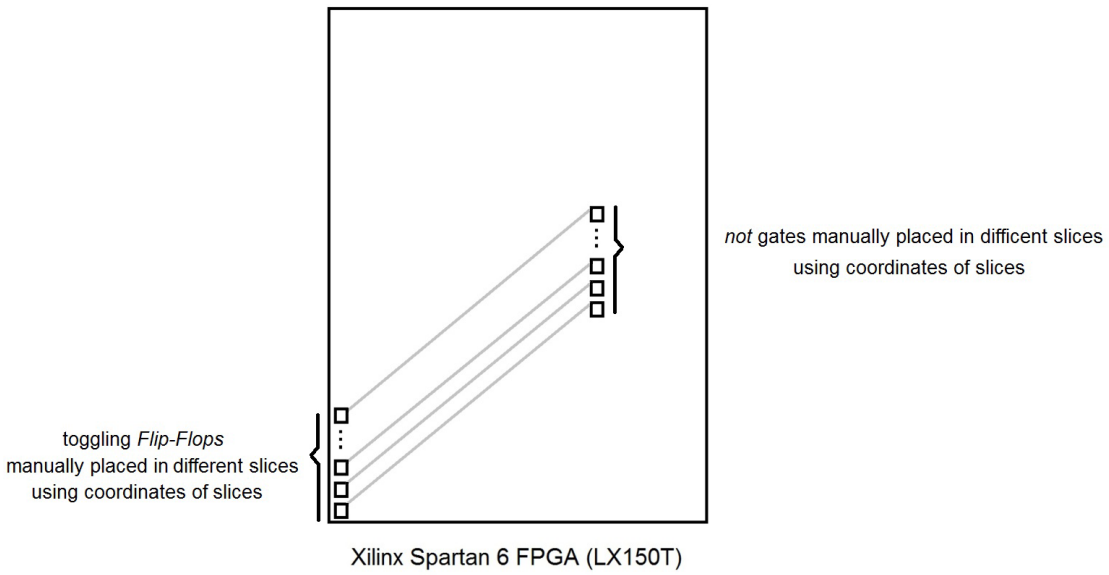


Figure 4.11: Manual placement of components in FPGA

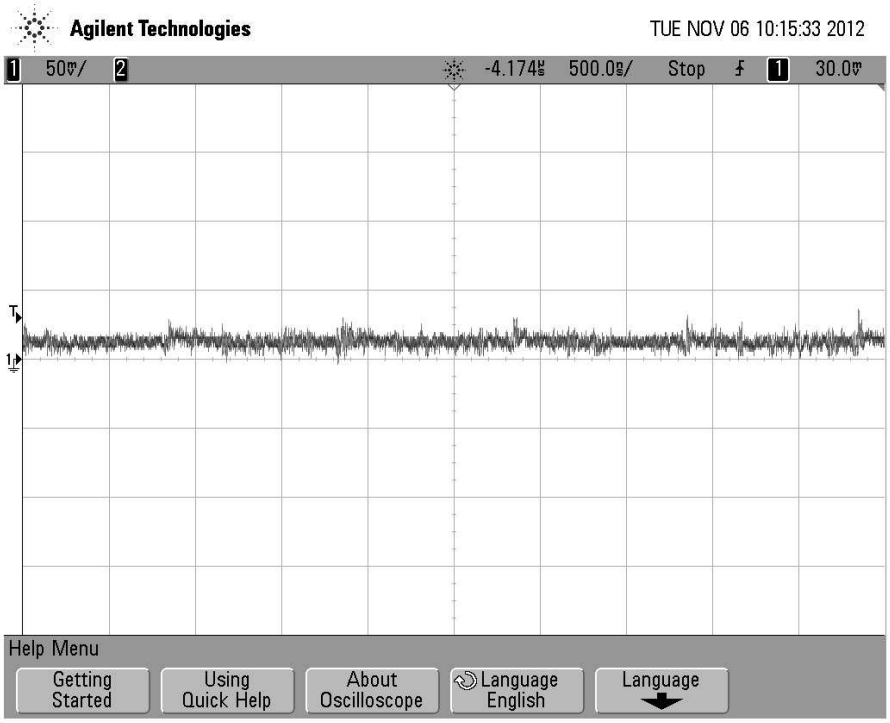


Figure 4.12: Toggling outputs

by a proper re-routing of the design, and the whole attacking concept would become meaningless.

Figure 4.11 shows a simple representation of placement and net-list of the circuit that was used for this test. 256 registers keep toggling at each cycle of the clock. Using HDL attributes, they are manually placed at the left bottom corner of the FPGA, and connected to some mid-right point of it where some simple operation such as inversion is performed. Manual routing inside of the FPGA is very difficult but since there is a long path between the logic blocks, it is expected that compiler uses global interconnect for connecting them. Also since the number of registers and amount of combinational blocks is small, what is seen would be related to global interconnect only. After testing this case the result was a flat line for the power, meaning that 256 lines of global interconnect do not have a detectable effect in power consumption.

As a final test the effect of the output ports were examined. In other words it was desired to see what happens if output pins of the FPGA, without having loads, are all toggling. The intent of this test was to see if the output logic has a different amount of effect, and causes a distinguishable pattern on the consumed power. Figure 4.12 shows the result when 256 pins of FPGA are toggling in each clock cycle at the same time. Since no extra logic is implemented in this circuit, and changes in outputs almost happen right after the edge of the clock, the edges in the power diagram are sharp and right at the edge of the clock too. The amplitude is detectable, but not significant. It seems output ports, if not driving any load, do not have a major impact on the consumed power.

4.4 Conclusions of Results

The power sampling board functions well and seems effective in sampling power. Some expected phenomenon such as different patterns for combinational and sequential logic were tested and confirmed. Some important tests, such as tests for confirming linearity and effect of global interconnect, were also applied. Also the precision in measuring the consumed power of FIFOs was obtained. Although the device has some desired characteristics such as linearity of consumed power, and no negative dominant factor, but the precision seems larger than what an attack, such as the one proposed for Whitenoise, requires. Using amplifier should improve this result, maybe to hundreds of registers instead of $16k$ bits, but this is still not enough. However as it was noted before, this requirement in attacking Whitenoise is not common among

common attacks, and it is up to the designers, and based on the application, that if the proposed scenario is going to be taken as a serious threat or not.

Chapter 5

Summary of Conclusions and Possible Future Works

In this final chapter what was discussed in the two previous chapters are summarized. As for the chapter 3, Whitenoise algorithm and implementation was studied. The following are the contributions of discussions of this chapter.

1. Whitenoise encryption algorithm was introduced.
2. The implementation of the algorithm was reversed engineerined, and provided as a block diagram.
3. DPA attacks againts Whitenoise were studied and rejected.
4. Different parts of the implementation against Template Attacks were studied. Number of samples available, behavior of noise, and requirements for applying such attacks were observed.
5. It was shown that Template Attacks are not applicable, or at least very weak, against the implementation.
6. A new scenario to obtain the Hamming Distances of sub-keys was proposed.
7. The number of samples necessary to apply the scenario was calculated. It was shown sampling time would be less than a second, even in a slow encryption process.

8. An improvement to that scenario was presented, which is able to find the length of the sub-keys, and also able to find the value of the every byte of sub-keys out of the HDs. Therefore it is capable of completely breaking the implementation.

9. It was concluded that considering the fact that Whitenoise is resistant to common attacks, and the proposed scenario requires very accurate equipment, large amount of calculations and a lot of tests to apply and obtain samples, the implementation is fairly strong.

10. The main weakness of Whitenoise, both in applying the proposed scenario, and also a possible Template Attack, was shown to be the separable portion of power consumption related to shift-registers.

In chapter 4, the power sampling board was reviewed. The following summarizes the contributions of this chapter:

1. Requirements for power consumption measurement were mentioned.

2. Some concepts in analouge design, such as the schematic of the amplifier, and the components necessary, were studied.

3. Many other concepts necessary to consider in developing the digital section and the board overall were presented. These include: star grounding, inductance reduction, embedding a BGA component, and etc.

4. The resolution of board and device in detecting toggling registers was identified.

5. Some combinational circuit was suggested and tested to clearly examine the behavior of power consumption regarding combinational blocks. Different pattern of consumed power for combinational circuit, comparing to the pattern of sequential logic, was observed.

6. The linear property of power consumption diagrams, which is an important requirement for applying attacks, was tested and confimed.

7. The effect of global interconnect, by manually placement of slices in FPGA, was tested. It was shown that global interconnect does not have a dominant negative effect.

8. The effect of toggling ouputs was tested. It was shown that although their effect is detectable, but it is not dominant as long as the outputs are not deriving loads.

As the final conclusions and possible future works, as mentioned before, the author highly recommends paying more attention to the separable portion of power consumption in the Whitenoise implementation. If the designer can come up with a new method to go around this problem, it would be a noticeable improvement. Although other techniques such as masking and noise addition can also be applied. But if this weakness is resolved, Whitenoise seems strong enough by itself.

About the power sampling board, applying more measurements for more pattern recognition, or using the amplifier to observe the improvements, are some suggestions for future work. Also as another suggestion, using the board to perform an actual attack on a known breakable implementation, or observing the effect of masking techniques on the power consumption, can be thought as other possible future works.

Bibliography

[1] Dakshi Agrawal, Bruce Archambeault, Josyula R. Rao, and Pankaj Rohatgi. The EM side-channel(s). pages 29–45. Springer-Verlag, 2003.

[2] Dakshi Agrawal, Josyula R. Rao, and Pankaj Rohatgi. Multi-channel attacks. pages 2–16. Springer-Verlag, 2003.

[3] Ross Anderson and Markus Kuhn. Low cost attacks on tamper resistant devices. pages 125–136. Springer-Verlag, 1997.

[4] Dan Boneh, Richard A. DeMillo, and Richard J. Lipton. On the importance of checking cryptographic protocols for faults. pages 37–51. Springer-Verlag, 1997.

[5] Stephan Boren and Andre Brisson. Software specifications for tinnitus utilizing whitenoise. Technical report, Whitenoise Lab, 2003.

[6] Stephen Laurence Boren and Andre Jacques Brisson. Dynamic distributed key system and method for identity management, authentication servers, data security and preventing man-in-the-middle attacks, 01 2009.

[7] Suresh Chari, Josyula R. Rao, and Pankaj Rohatgi. Template attacks. pages 13–28. Springer-Verlag, 2002.

[8] Nicolas Courtois and Willi Meier. Algebraic attacks on stream ciphers with linear feedback. *Eurocrypt ePrint Archive*, 2003.

[9] Paul N. Fahn and Peter K. Pearson. Ipa: A new class of power attacks. pages 173–186. Springer-Verlag, 1999.

[10] Eby G. Friedman. *Clock distribution networks in VLSI circuits and systems.* Institute of Electrical and Electronics Engineers, 1995.

[11] Jonathan J. Hoch and Adi Shamir. Fault analysis of stream ciphers. page 240253. Springer-Verlag, 2004.

[12] Whitenoise Laboratories (Canada) Inc. www.wnlabs.com.

[13] Paul C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. pages 104–113. Springer-Verlag, 1996.

[14] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. pages 388–397. Springer-Verlag, 1999.

[15] Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *Power analysis attacks - revealing the secrets of smart cards*. Springer-Verlag, 2007.

[16] Thomas S. Messerges, Ezzat A. Dabbish, Robert H. Sloan, and Senior Member. Examining smart-card security under the threat of power analysis attacks. *IEEE Transactions on Computers*, 51:541–552, 2002.

[17] Victor P. Nelson, H. Troy Nagle, Bill D. Carroll, and David Irwin. *Digital Logic Circuit Analysis and Design*. Prentice Hall, 1995.

[18] Siddika Berna Ors, Elisabeth Oswald, and Bart Preneel. Power-analysis attacks on an FPGA, first experimental results. pages 35–50. Springer-Verlag, 2003.

[19] Federal Information Processing Standards Publication. FIPS 46-3: The official document describing the DES standard, 1999.

[20] Federal Information Processing Standards Publication. FIPS 197: The official AES standard, 2001.

[21] Jean-Jacques Quisquater and David Samyde. Electromagnetic analysis (ema): Measures and counter-measures for smart cards. page 200210. Springer-Verlag, 2001.

[22] Christian Rechberger and Elisabeth Oswald. Practical template attacks. pages 443–457. Springer-Verlag, 2004.

[23] University of Victoria ReCoEng: Laboratory of Reconfigurable Computing Engineering. recoeng.ece.uvic.ca.

[24] Li Shang, Alireza S. Kaviani, and Kusuma Bathala. Dynamic power consumption in Virtex-II fpga family. pages 157–164. ACM, 2002.

[25] Sergei P. Skorobogatov and Ross J. Anderson. Optical fault induction attacks. pages 2–12. Springer-Verlag, 2002.

[26] Marc Stevens. Fast collision attack on md5. Technical report, Cryptology ePrint Archive, 2006.

[27] Junko Takahashi, Toshinori Fukunaga, and Kazuo Sakiyama. Fault analysis on stream cipher MUGI. pages 420–434. Springer-Verlag, 2010.

[28] I. Traore and M.L. Yanguo. Evaluation of whitenoise cryptosystem. Technical report, ECE Department of University of Victoria, Canada, 2003.

[29] David Wagner. A security evaluation of Whitenoise. *IACR Cryptology ePrint Archive*, page 218, 2003.

[30] Xilinx. Spartan-6 family overview. www.xilinx.com/support/documentation/.

[31] Xilinx. Spartan-6 FPGA packaging and pinouts specifications. www.xilinx.com/support/documentation/.

[32] Xilinx. Spartan-6 FPGA PCB design and pin planning guide. www.xilinx.com/support/documentation/.

[33] Xilinx. Xilinx inc. www.xilinx.com.

Appendix A

Power Sampling Setup

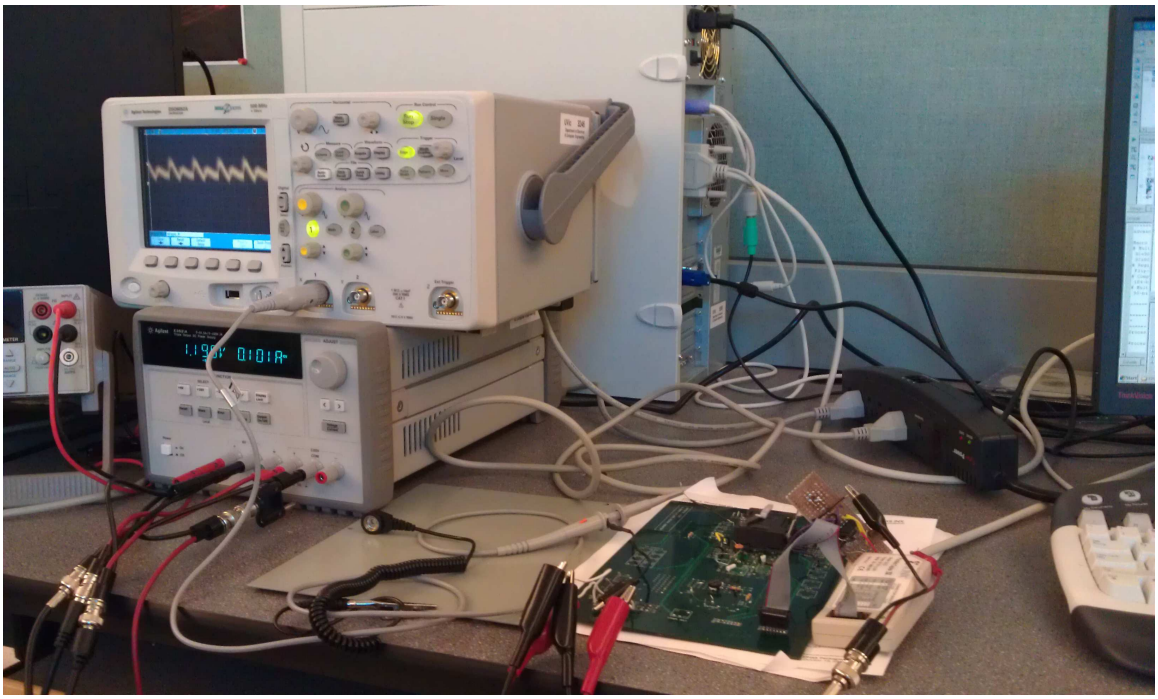


Figure A.1: Power Sampling Setup

Appendix B

PCB Layers (Final Version)

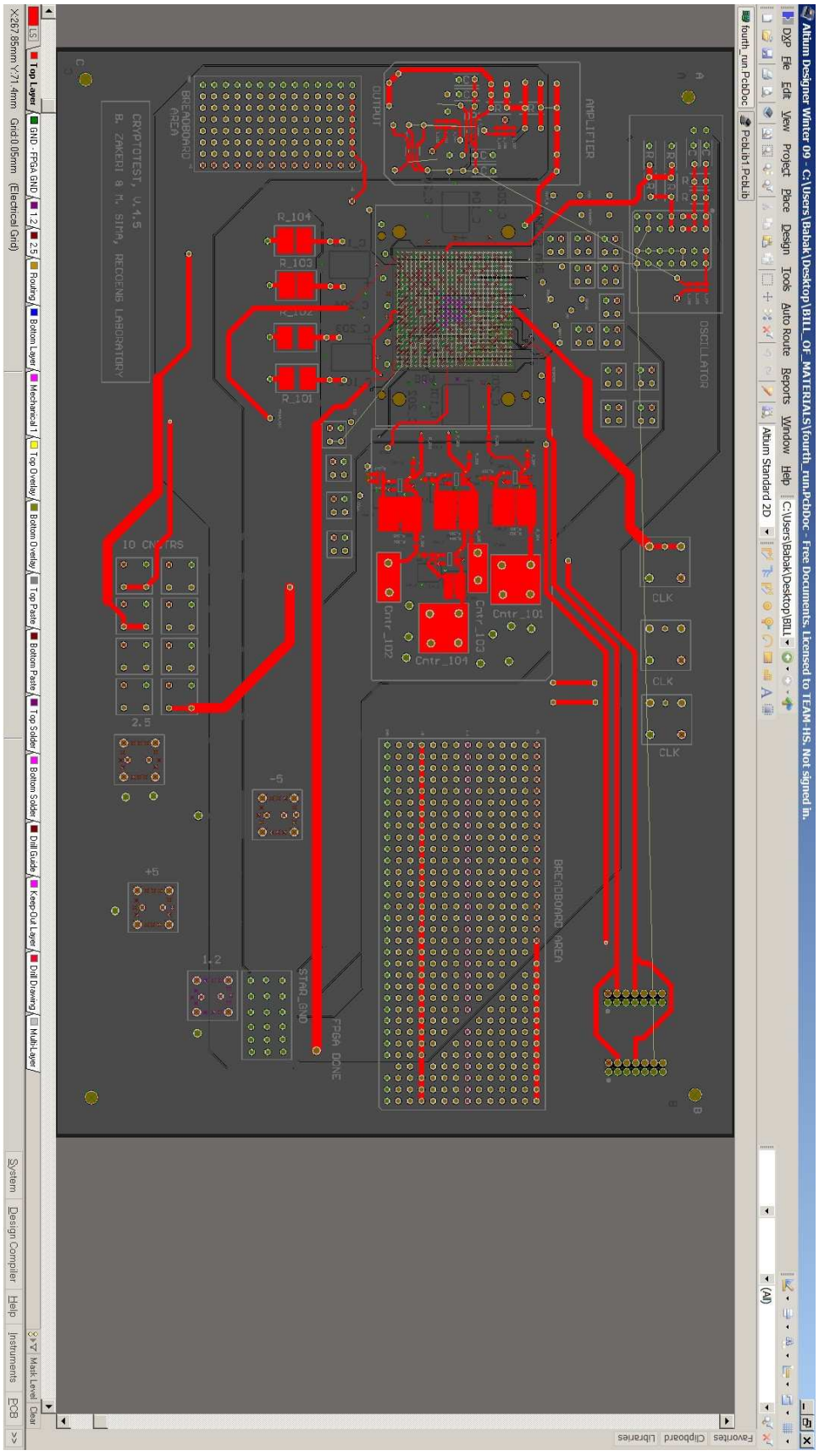


Figure B.1: Top Layer

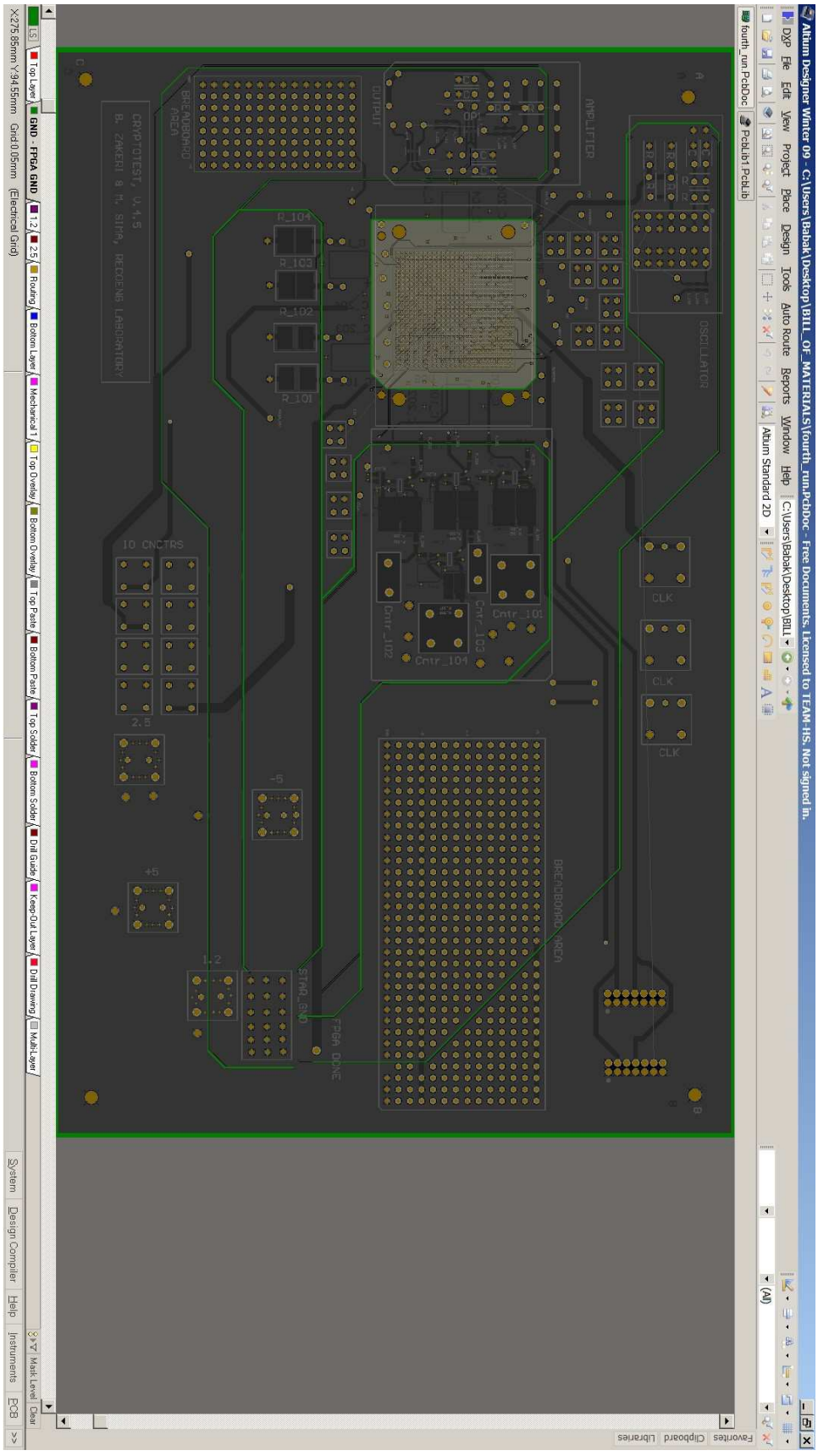


Figure B.2: Ground Layer

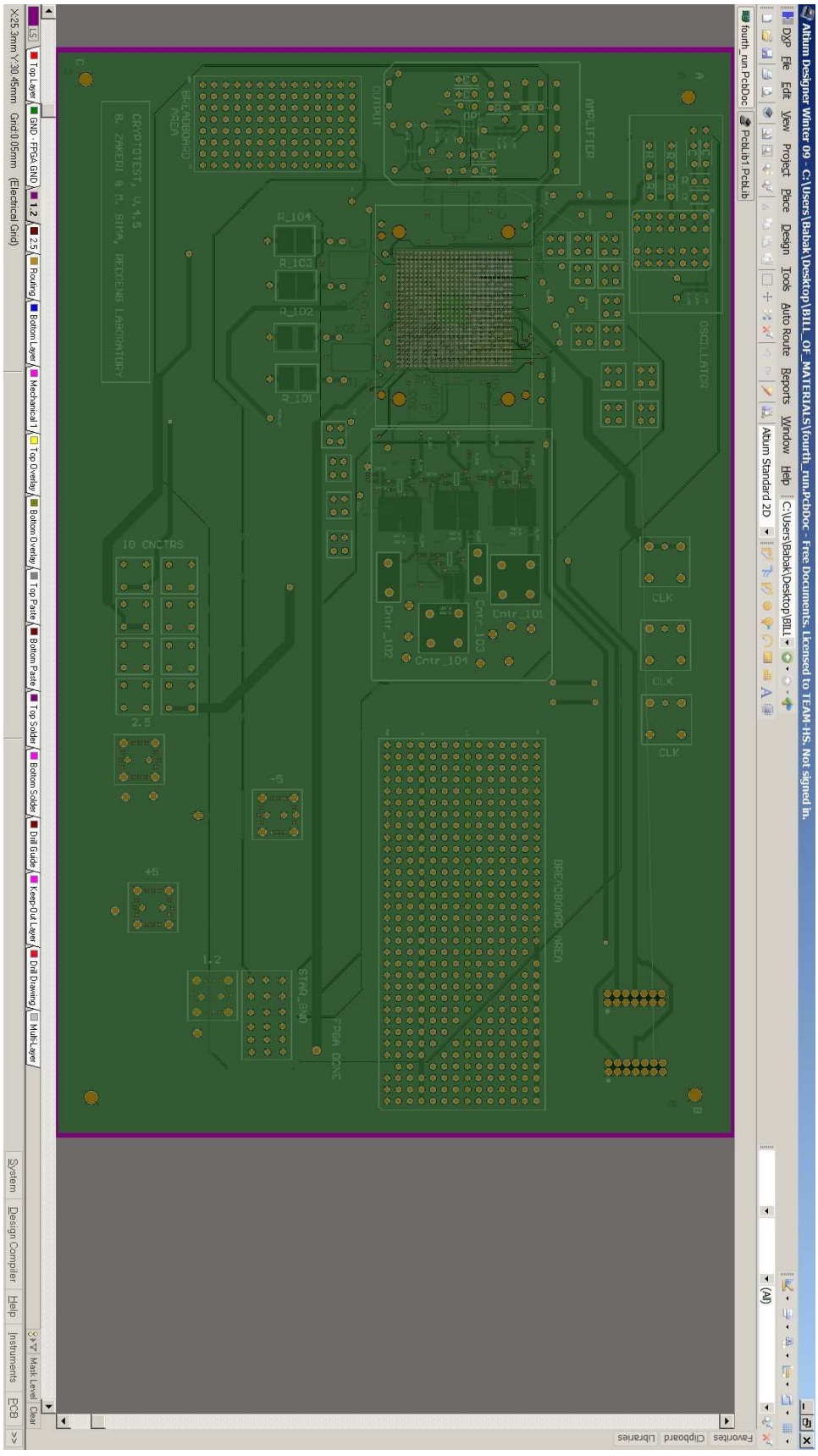


Figure B.3: 1.2v Supply Layer

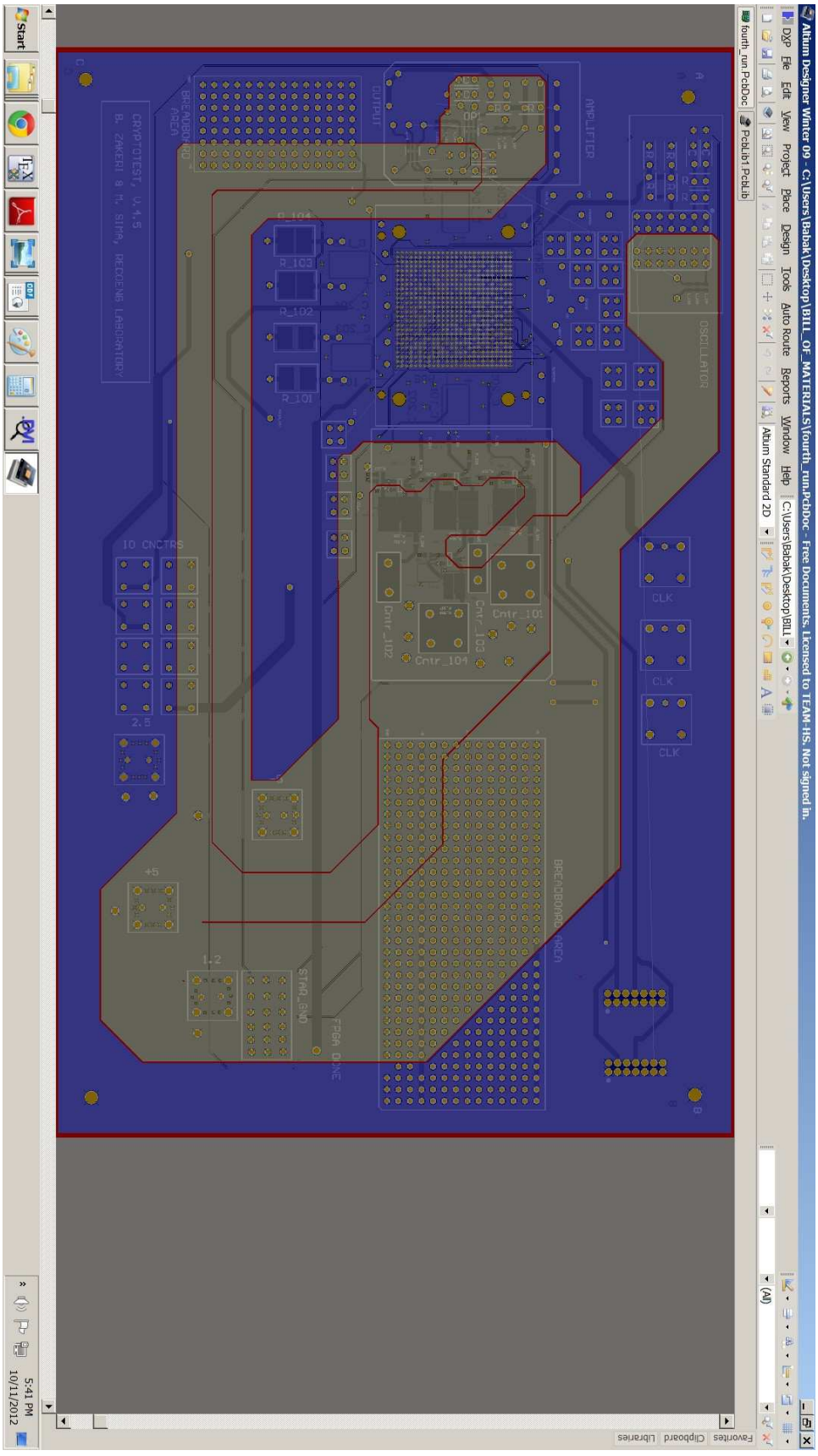


Figure B.4: 2.5v, +5 and -5 Supply Layer

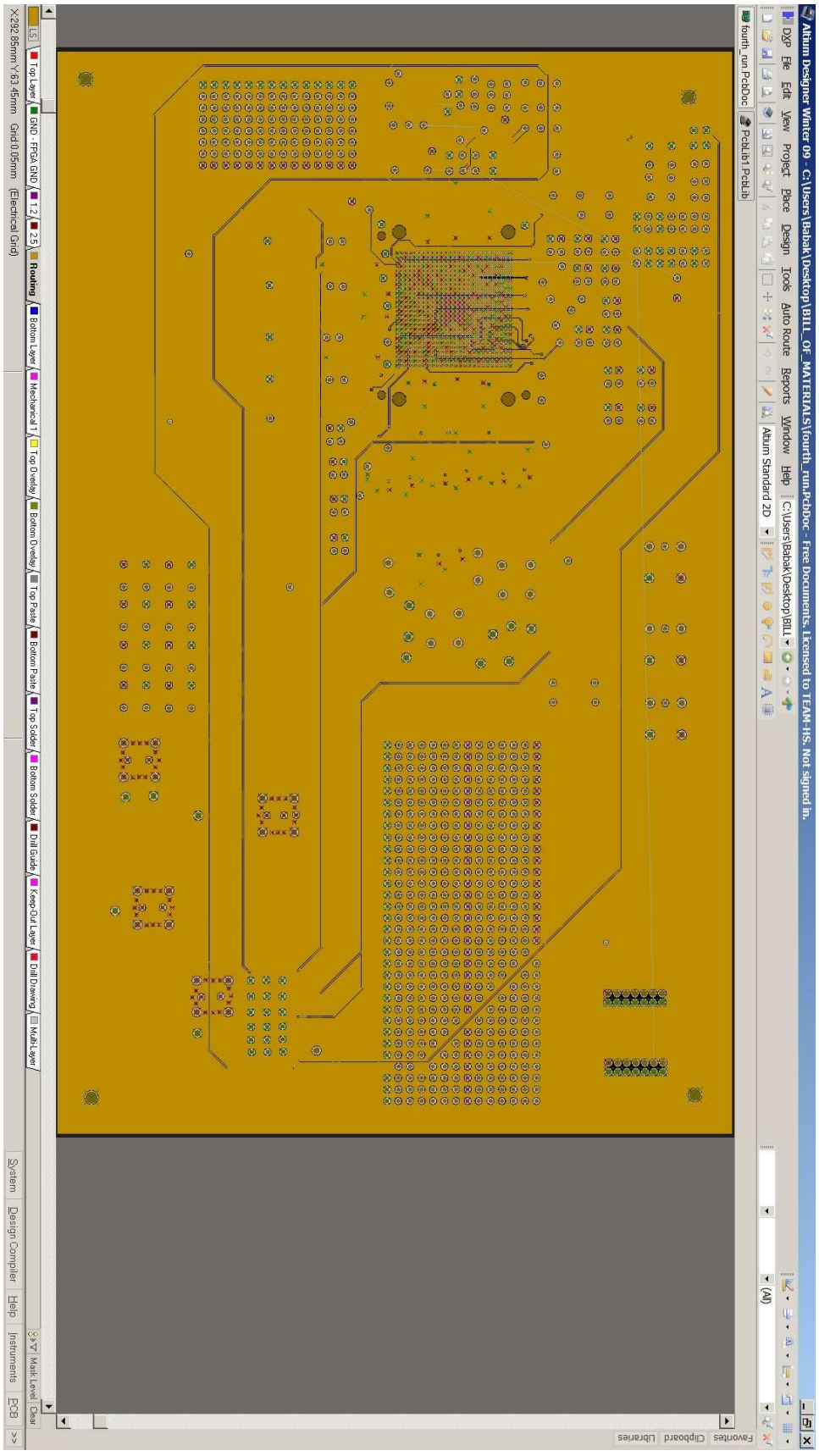


Figure B.5: Routing Layer

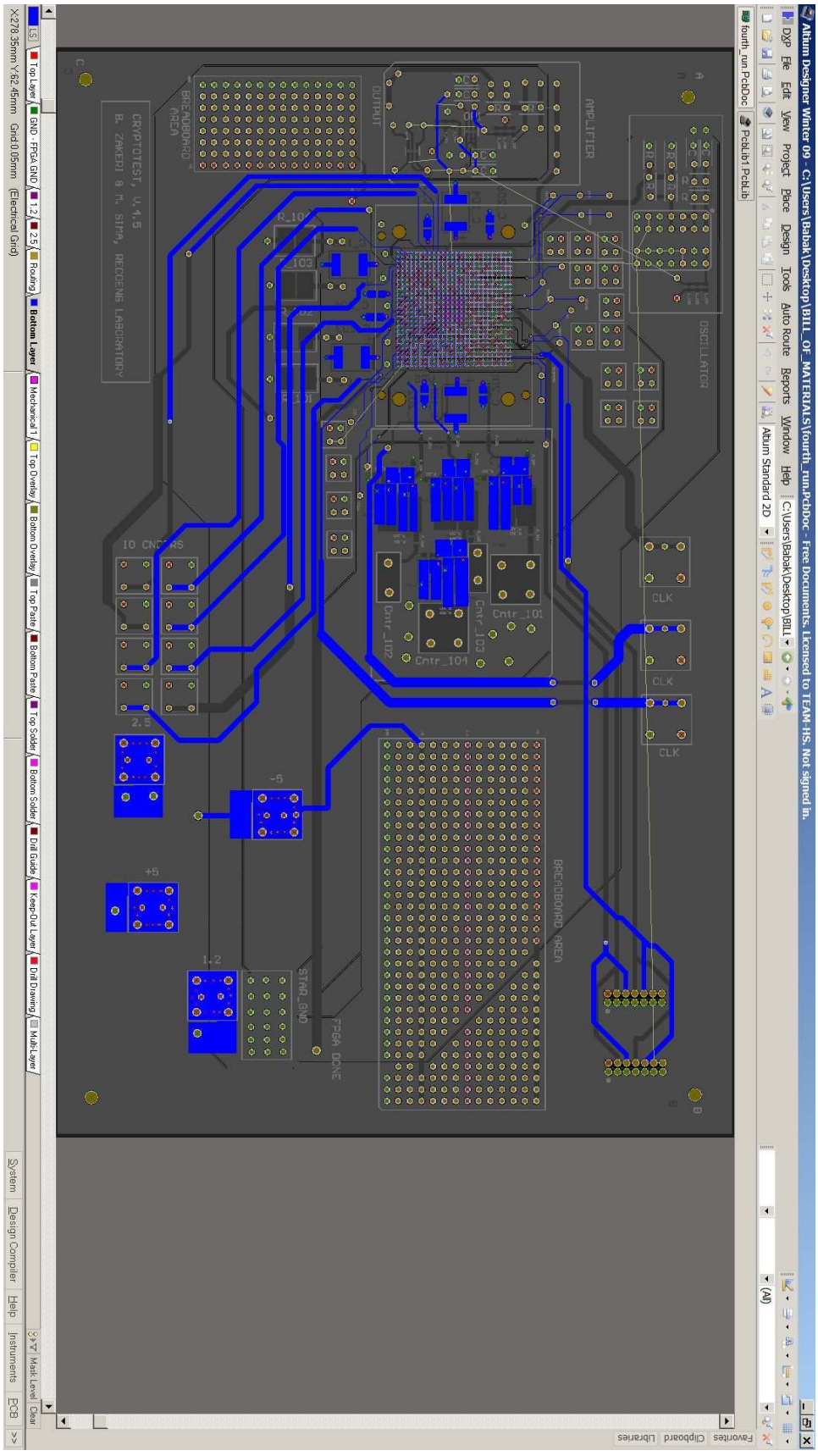


Figure B.6: Bottom Layer

Appendix C

Additional Information

The following brief information mentions where and how to access the resources of this project for whoever wants to continue this work and also gives some caution notes. The board and setup are in the ReCoEng[23] lab of ECE department of University of Victoria. The Xilinx tool for programming the FPGA is a licensed version of ISE 11 available in the lab and is updated to 11.5. The tools available in the department don't provide support for the Spartan 6 FPGA family and thus this version is used instead. The tool can not be updated anymore since the license has expired but it can be still used without updating as copyright terms indicate. Originally three licenses of software were provided by Xilinx and one or two licenses might still be left so the software might be installable on another machine.

The code for WN[12] implementation, along with all other documents, articles and files related to the project are available on the other system used by the author of this report. The PCB design software, Altium Designer, is also available on this machine as a trial version. The fabricated last version of the board (the 4th version) has few minor issues which are fixed in the PCB project file for future. These include unconnected VIAs in the Star Ground area and a few unconnected FPGA GND VIAs in the FPGA region.

Whitenoise code may need modifications before implementing on FPGA. Some of these modifications are already applied. These include changing the asynchronous Flip-Flops to synchronous ones as Spartan 6 architecture requires, changing the multiplexer from a core to some HDL code since Spartan 6 doesn't support this core, and a few other minor changes to fix the length of the shift-registers. Also some sample data has been provided for S-Box, and initial values of the Shift-Registers and are stored as COE files in the coregen directory of the project.

If the project is going to be rebuilt in ISE, note thsat specific libraries should be created in that ISE project and the related files put in them instead of the work directory. These libraries are included in the HDL codes of the project and based on where they are included one can rebuilt the project file.

Some notes on synthesis and other problems that occurred and were solved along the process were logged and available in the documents. These better be reviewed before writing any new HDL code to implement on FPGA since they can save much time and confusion when using the ISE tool of Xilinx. For troublesome pieces of code the synthesis takes much time or it may run out of memory so it's better to consider these notes first.

It is strictly recommended to use ESD Wrist Strap while using the board since there happened accidents before, ending up in burning the FPGA. Also power supply should be set with limits for the currents. A $200mA$ should be enough for any supply but sometimes the supply takes more current for larger circuits so be aware of this if your circuit is not functioning correctly. About the supplies, turn the output off while connecting the wires and do the same before disconnecting them. Be cautious about using supplies since wrong direction of plugging in the supply wires would burn the FPGA.

The holes provided for stands in the 4 corners of the board are connected to ground layer of the board. So if metal screws are used and the board is placed inside of the shield box, note that the shield box is then ground too. In general be careful about short circuits between supply wires since by being a little careless an accident might happen resulting in burning the FPGA.

Appendix D

Experiments on the Final Version of Board

This appendix provides the results of the experiments, applied later on the fourth version of the board. As mentioned, at the time of preparing this thesis the fourth version of the board was still under the fabrication process and thus not usable for performing tests. The three tests applied in fourth version include the following:

1. Tests to confirm the consistency of linearity now that the amplifier is added.
2. Tests to measure the new percision considering the existance of amplification.
3. And obtaining the harmonics of a periodic signal, which is the addition of a few periodic signals with smaller periods.

About the first test, the linearity was confirmed. As experiments showed, besides the large signals which enter the saturation region of amplification, very small distortion was observed on the main signal. In the fourth version of the board, the input and feedback resistors, indicated in figure 4.1, were chosen in a way that the amplifier provides a 100 times amplification. Since the supply voltages of OPAMPs are +5 and -5, any value less than 50mv in the FPGA GND, is observable on the output of the OPAMP without entering the saturation region.

About the second test, the precision easily improves to below 1,000 Flip-Flops, comparing to the board without amplification, where this value was about $16k$ Flip-Flops. There are two problems that currently prevent a better detection. The first problem is that, for detecting smaller values, a stronger amplifier is necessary. Increasing the amplification ratio in the current amplifier is not possible, since by doing

so, although the smaller signals will be detectable, but the larger signals would enter the saturation region of OPAMP, and thus not detectable any more. The second problem is that for smaller signals, they are masked with the ripples caused by the oscillator circuit. The ferrite beads added to the fourth version have improved the performance, and the amount of noise is decreased. However it is not completely removed and it is still preventing more precise measurements. Since the board can be improved regarding both of these issues, it is reasonable to believe that the precision can still be improved, maybe to a hundred, or even a few tens of Flip-Flops.

Having the amplifier added to the board, it was possible to perform more tests. The one that is described here relates to part of the proposed scenario to attack Whitenoise. It was mentioned in section 3.5, that as a requirement for attacking Whitenoise, the length of the sub-keys, which themselves are dependant on the key seeds, should be known. It was also mentioned that for finding the length of the sub-keys, attacker can perform a Fourier Transform, and by observing the harmonics, she can find the lengths.

A similar thing was done in this experiment. A few circuits, each of which causing a periodic pattern in the power consumption, were merged together to see if the building harmonics are detectable in the power consumption of the merged circuit. Since the board is not capable of measuring changes for small number of Flip-Flops, the circuits used here were not exactly what were mentioned in 3.4 and 3.5. Nevertheless the goal was providing periodic patterns on the power consumption which was satisfied. Figures D.1 through D.3 show the power consumption of these circuit when implemented separately. The repeating periods in these circuits are 3 clock cycles, 5 clock cycles and 7 clock cycles accordingly.

In the left hand side of each figure you see the power consumption, compared to the clock of the system, and the repeating patterns are quite clear in the figures. The right hand side of the figures on the other hand, shows the result, when the FFT[1] function of the scope is applied on the results. The clock frequency of the circuit is $1MHz$. Thus the periodic signals have the frequency of $333kHz$, $200kHz$ and $143kHz$. These signals, and their harmonics, are quite visible in figures D.1 to D.3.

Figure D.4 shows the power consumption when these circuits are merged into one circuit. The first few harmonics that are highlighted in the figure, happen at $143kHz$, $200kHz$, $286kHz$, $333kHz$ and $400kHz$, which accordingly, correspond to first harmonic of $7 * clk$ signal, first harmonic of $5 * clk$ signal, second harmonic of

[1]Fast Fourier Transform

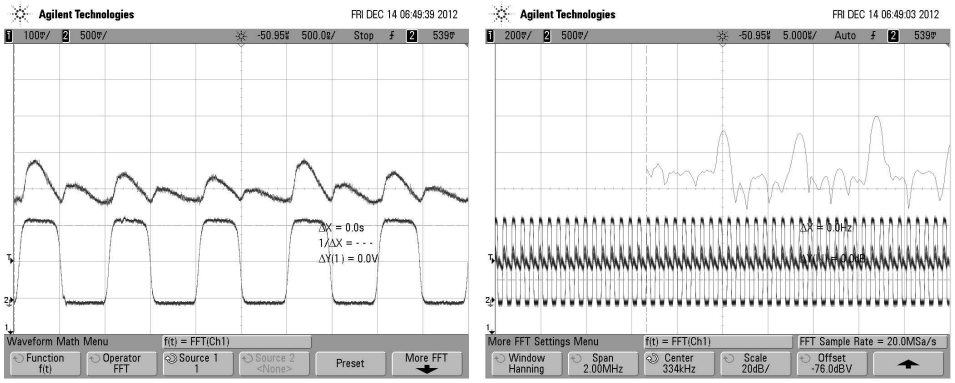


Figure D.1: Power Consumption for a signal with repeating period of $3 * clk$

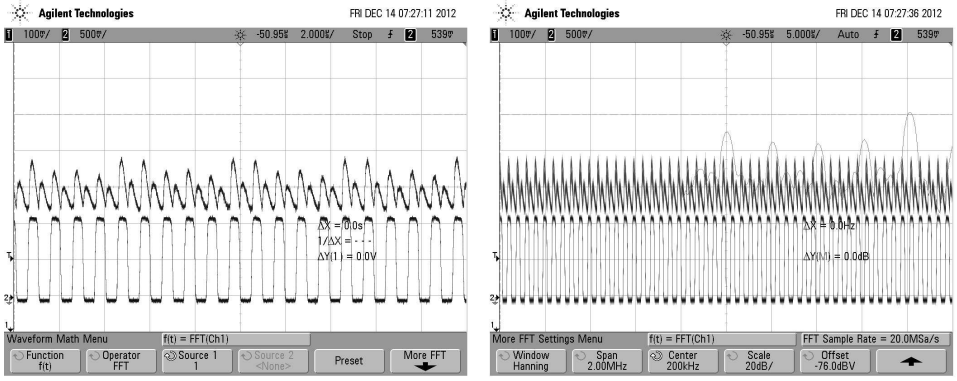


Figure D.2: Power Consumption for a signal with repeating period of $5 * clk$

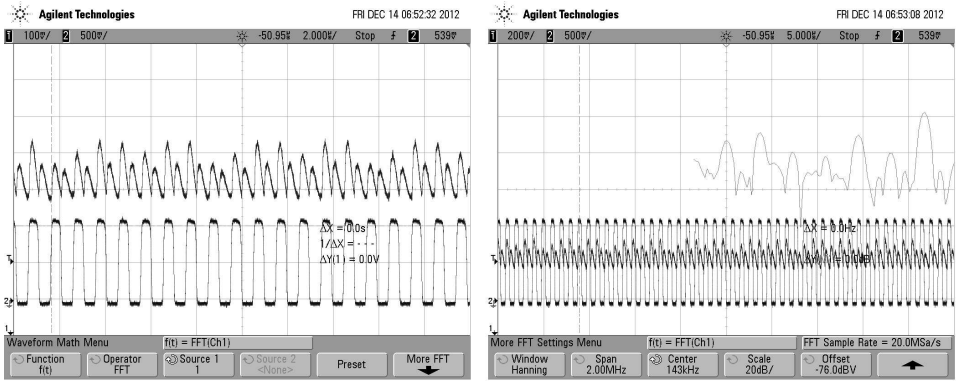


Figure D.3: Power Consumption for a signal with repeating period of $7 * clk$

$7 * clk$ signal, first harmonic of $3 * clk$ signal and second harmonic of $5 * clk$ signal. This result is an amazing evidence, that the repeating periods are in fact obtainable by measurements.

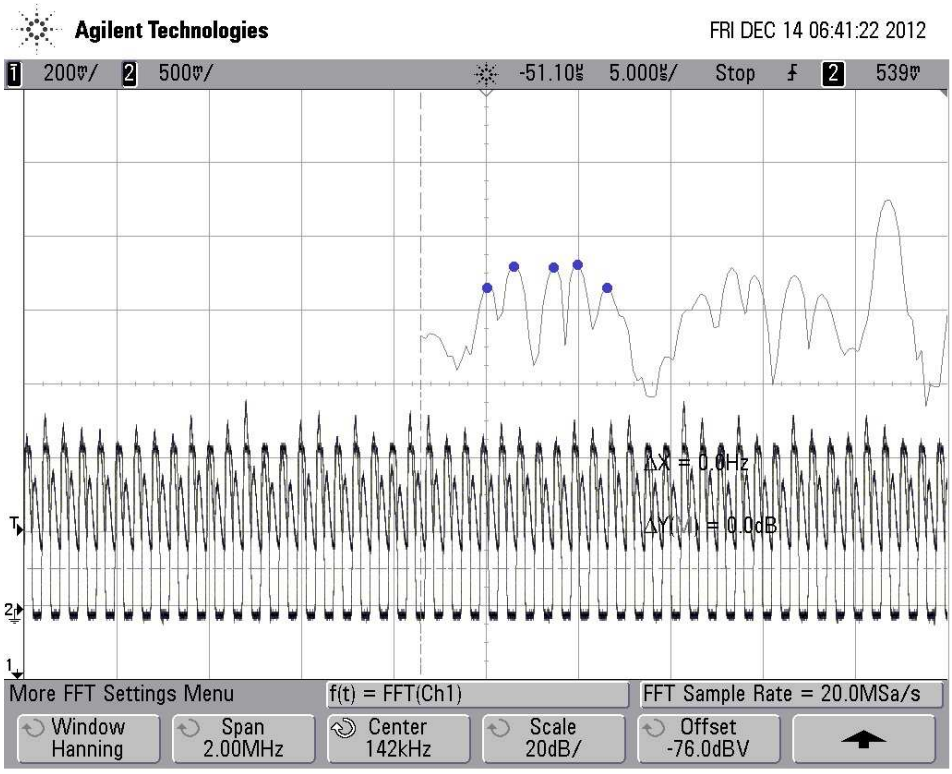


Figure D.4: Power Consumption for the mixed signal