9-2015

A Secure Neuromemristive Primitive to Mitigate Correlation Power Analysis on SHA-3 Hash Function

James B. Thesing
jbt1751@rit.edu

Follow this and additional works at: http://scholarworks.rit.edu/theses

Recommended Citation

This Thesis is brought to you for free and open access by the Thesis/Dissertation Collections at RIT Scholar Works. It has been accepted for inclusion in Theses by an authorized administrator of RIT Scholar Works. For more information, please contact ritscholarworks@rit.edu.
A Secure Neuromemristive Primitive to Mitigate Correlation Power Analysis on SHA-3 Hash Function

by

James B. Thesing

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of Master of Science in Computer Engineering

Supervised by

Associate Professor Dr. Dhireesha Kudithipudi
Department of Computer Engineering
Kate Gleason College of Engineering
Rochester Institute of Technology
Rochester, New York
September 2015

Approved by:

Dr. Dhireesha Kudithipudi, Associate Professor
Thesis Adviser, Department of Computer Engineering

Dr. Marcin Łukowiak, Associate Professor
Committee Member, Department of Computer Engineering

Dr. Stanislaw Radziszowski, Professor
Committee Member, Department of Computer Science
Dedication

To my family, for their support and encouragement to get me to this point. To my wife, Natasha, for giving me more support than she could realize.
I would like to thank my thesis adviser Dr. Dhireesha Kudithipudi for her guidance and advice during my thesis work. I would like to thank Dr. Marcin Łukowiak for serving on my committee and countless hours spent discussing power analysis techniques and security issues. Also, I want to thank Dr. Stanisław Radziszowski for serving on my committee and giving his time and guidance on my final thesis document.

I would also like to thank the members of the RIT Nano lab whose support, discussions and presentations led to much of the success in this work: James Mnatzaganian, Dan Christiani, Cory Merkel, Levs Dolgovs, Lennard Streat, Qutaiba Saleh and Abdullah Zyarah. They are a great team to work with.

Finally, a special thank you to Xuan Tran. Our collaborative efforts were the basis of figuring out the design flow and CPA attack methodology that led to both of our successful works.
Abstract
A Secure Neuromemristive Primitive to Mitigate Correlation Power Analysis on SHA-3 Hash Function
James B. Thesing
Supervising Professor: Dr. Dhireesha Kudithipudi

Passing messages to soldiers on the battle field, conducting online banking, and downloading files on the internet are very different applications that all share one thing in common, concerns over security of the data being processed. Data security depends on the cryptographic systems that take into account both the algorithmic weakness and the weaknesses of the hardware devices they are implemented on. The current dominant hardware design medium is complementary metal-oxide-semi-conductor (CMOS). CMOS has been shown to leak more power as the technology node size decreases. The leaked power has a strong correlation with the bits being manipulated inside a device. These power leakages have brought on a class of power analysis that is able to extract secret information being processed in the algorithm with far less computational power than brute force guessing. Recently, many hardware designs have been proposed which have shown resistance against different forms of power analysis by changing hardware layouts; however, these designs are realized in the same technology, CMOS, that causes the side channel attack problem.

There are many emerging technologies that are becoming more practical to implement in conjunction with CMOS. Of these, neuromemristive systems have two characteristics that can be exploited to prevent side channel attacks: low power operation and stochastic behavior. Attacks were conducted on both CMOS and neuromemristive based mitigations of the SHA-3 algorithm. In this thesis, digital side channel attack mitigations are created to exploit dual-rail logic. A secure neuromemristive primitive is designed using neural logic blocks that, to the best of our knowledge, have not been considered by others in mitigation of power analysis. Also, an in-depth analysis of power attacks on linear functions compared to typical non-linear attack points is conducted. Metrics such as number of power traces used for the Correlation Power Analysis (CPA), correlation coefficients, confidence ratios, power consumption, and transistor count were used to compare circuit performance. Success rate of guessing a key during SHA-3 operations, while configured as a MAC, was used as a system benchmark. It was found that CMOS is effective in countermeasures when masking linear functions, with the ability to use current standard cells, in ASIC design. If reconfigurable circuits are considered, the neuromemristive circuit had the overall best mitigation strength with almost complete decoupling of input data to power dissipated; moreover, this design offered low power operation and small form factor compared to the original circuit.
# Contents

Dedication ................................................................. ii

Acknowledgments ......................................................... iii

Abstract ........................................................................ iv

1 Motivation ................................................................. 1

2 Background ............................................................... 3
   2.1 Side Channel Attacks (SCA) ........................................ 3
   2.2 Cryptographic Hash Functions ................................. 3
   2.3 Message Authentication Codes (MACs) ...................... 5
   2.4 Merkle-Damgard Vs. Sponge Hash Construction .......... 7
   2.5 SHA-3 ................................................................. 9
      2.5.1 State XOR .................................................... 11
      2.5.2 Theta (θ) ..................................................... 12
      2.5.3 Rho (ρ) ....................................................... 12
      2.5.4 Pi (π) ......................................................... 14
      2.5.5 Chi (χ) ....................................................... 14
      2.5.6 Iota (ι) ....................................................... 16
   2.6 Neuromorphic Computing ........................................ 17
   2.7 Memristor Devices ................................................ 18

3 Supporting Work ....................................................... 20
   3.1 SCA with Power Analysis ........................................ 20
      3.1.1 Simple Power Analysis (SPA) ............................ 20
      3.1.2 Differential Power Analysis (DPA) ..................... 23
      3.1.3 Correlation Power Analysis (CPA) ..................... 26
   3.2 Current Mitigating Designs ....................................... 27
      3.2.1 Dual-rail with Precharge Logic ......................... 27
      3.2.2 Software level ............................................. 29
3.3 Neural Logic Block (NLB) .................................................. 29

4 Hardware Circuits .......................................................... 33
4.1 Digital Implementations ...................................................... 33
  4.1.1 High-Speed Core - Base Circuit .................................. 33
  4.1.2 Mitigation One - Dual Core Design ............................ 34
  4.1.3 Mitigation Two - $\theta$ Plane Masking ......................... 34
  4.1.4 Mitigation Three - NLB $\theta$ Plane Masking ............... 36
4.2 Analog Implementations .................................................. 38
  4.2.1 Mitigation Four - NLB $\theta$ Plane Masking - default .... 38
  4.2.2 Mitigation Five - NLB $\theta$ Plane Masking - Dual Implementation .. 42
4.3 Summary ................................................................. 42

5 Design Flow ............................................................... 43
5.1 Digital Simulations ....................................................... 43
5.2 Analog Simulations ....................................................... 46
5.3 Script API ................................................................. 47
  5.3.1 generate.py ......................................................... 47
  5.3.2 runSimulation.py .................................................. 47
  5.3.3 mtnlbControl.py ................................................... 48
  5.3.4 initialize.m, createPowerTrace.m, attack_cpa.m, thesisPlots.m ... 48
  5.3.5 Folder Structure .................................................... 50

6 Results ................................................................. 51
6.1 Initial SHA-3 Attack ...................................................... 51
  6.1.1 Performance Metrics ............................................... 56
  6.1.2 Benchmarks .......................................................... 60
6.2 Mitigation One — Dual-Core Design ............................... 62
6.3 Mitigation Two — XNOR $\theta$ plane masking .................... 70
6.4 Mitigation Three — MTNLB $\theta$ plane masking - digital .... 77
6.5 Mitigation Four — MTNLB $\theta$ plane masking - analog ...... 84
6.6 Mitigation Five — MTNLB $\theta$ plane masking - analog dual-core ... 88
6.7 Reconfigurable Circuits ............................................... 93
6.8 Summary ................................................................. 94

7 Conclusions And Future Work ........................................... 96
7.1 Conclusions ............................................................. 96
7.2 Future Work ............................................................ 98
Bibliography ................................................................. 99
List of Tables

2.1 \( \rho \) offsets ....................................................... 14
2.2 \( \iota \) round constants ........................................... 16

6.1 Resource utilization for SHA-3 initial implementation. .......... 58
6.2 Power consumption for high-speed SHA3 implementation. ........ 60
6.3 Time required to brute force search a key space. ............... 60
6.4 Resource utilization for the SHA-3 dual-core implementation. .... 69
6.5 Power consumption for the SHA-3 dual-core implementation. .... 69
6.6 Resource utilization for the SHA-3 XNOR implementation. ....... 77
6.7 Power consumption for the SHA-3 XNOR implementation ........ 77
6.8 Resource utilization for a single digital MTNLB. ................ 82
6.9 Resource utilization for the SHA-3 digital MTNLB implementation. 83
6.10 Power consumption for the SHA-3 digital MTNLB implementation. 84
6.11 Resource utilization for the SHA-3 analog MTNLB implementation. 88
6.12 Power consumption for the SHA-3 analog MTNLB implementation. 88
6.13 Resource utilization for the SHA-3 analog dual-MTNLB implementation. 93
6.14 Power consumption for the SHA-3 analog dual-MTNLB implementation. 93
6.15 Collected results for all circuits .................................. 95
## List of Figures

2.1 Side Channel Attack methodology ........................................... 4  
2.2 Standard hash function operation ........................................... 5  
2.3 Standard Message Authentication Code (MAC) Operation ............... 7  
2.4 Merkle-Damgard Hash Construction ........................................... 8  
2.5 Sponge Construction as can be seen in SHA-3 ........................... 9  
2.6 Keccak state terminology used by the Keccak authors .................. 11  
2.7 The filled SHA-3 state for one input vector in MAC configuration .... 11  
2.8 $\theta$ permutation of Keccak state ........................................... 13  
2.9 $\rho$ permutation of Keccak state [7] ...................................... 13  
2.10 $\pi$ permutation of Keccak state [7] ...................................... 15  
2.11 $\chi$ permutation of Keccak state ........................................... 15  
2.12 A simple neron model ......................................................... 17  
2.13 I-V characteristic of the piecewise linear memristor model [24] .... 19  
3.1 Distribution of power traces in an AES hardware implementation [10] . 21  
3.2 Power trace from an implementation of RSA [10] .......................... 22  
3.3 Power trace for one SHA-3 permutation .................................... 22  
3.4 Differntial power analysis attack model .................................... 24  
3.5 Differntial power analysis on AES key showing correct key byte guess. . 24  
3.6 An example of a non-linear and linear function’s impact on output bits. . . 25  
3.7 Wave Dynamic Differential Logic implementation of an AND gate .... 28  
3.8 Separability of AND and XOR logic functions ............................ 30  
3.9 Plot of activation functions for AND, OR, XOR, NAND, NOR and XNOR functions [24]. ......................................................... 31  
3.10 Energy delay product for the proposed NLB designs .................... 31  
4.1 Block diagram of the basic SHA-3 high-speed core implementation .... 34  
4.2 Block diagram of the SHA-3 dual-core mitigation ........................... 35  
4.3 Block diagram of the SHA-3 XNOR $\theta$ plane mitigation ................ 36  
4.4 Block diagram of the SHA-3 NLB $\theta$ plane mitigation .................. 37  
4.5 RTL Diagram of the Multi-Threshold Neuron Logic Block ................ 38
4.6 Block level diagram of the analog MTNLB. .......................... 39
4.7 Subcircuit diagram for analog MTNLB ............................... 39
4.8 Block diagram for the SHA-3 analog dual-MTNLB implementation. ... 41

5.1 Block diagram of the Synopsys design flow. ......................... 43
5.2 Flow of correlation attack in Matlab. .............................. 45
5.3 Block diagram of the Cadence design flow. ......................... 46
5.4 Folder structure used in this work for both simulation and attack. ... 49

6.1 A single SHA-3 power trace. ........................................... 52
6.2 50 power traces for the SHA-3 initial core implementation. ....... 52
6.3 Correlation coefficients for 6 key guesses of the SHA-3 initial implementation. ......................................................... 54
6.4 3D plot of the SHA-3 initial implementation correlation matrix. ... 55
6.5 Confidence ratios for the SHA-3 initial implementation. .......... 57
6.6 Confidence ratios of all 40 key bytes of the SHA-3 high-speed core. ... 59
6.7 Success rates for the SHA-3 initial implementation. ............... 61
6.8 Correlation coefficients for 6 key guesses of the SHA-3 dual-core implementation. ......................................................... 63
6.9 3D plot of the SHA-3 dual-core correlation matrix. ............... 64
6.10 Confidence ratios for the SHA-3 dual-core implementation. ....... 65
6.11 Confidence ratios of all 40 key bytes of the SHA-3 dual-core implementation. ......................................................... 67
6.12 Success rates for the SHA-3 dual-core implementation. .......... 68
6.13 Correlation coefficients for 6 key guesses of the SHA-3 XNOR implementation. ......................................................... 71
6.14 3D plot of the SHA-3 XNOR correlation matrix. .................. 72
6.15 Confidence ratios for the SHA-3 XNOR implementation. .......... 73
6.16 Confidence ratios of all 40 key bytes of the SHA-3 XNOR implementation. ......................................................... 75
6.17 Success rates for the SHA-3 XNOR implementation. ............. 76
6.18 Correlation coefficients for 6 key guesses of the SHA-3 digital MTNLB implementation. ................................................... 78
6.19 3D plot of the SHA-3 digital MTNLB correlation matrix. ........ 79
6.20 Confidence ratios for the SHA-3 digital MTNLB implementation. ... 80
6.21 Confidence ratios of all 40 key bytes of the SHA-3 digital MTNLB implementation. ......................................................... 81
6.22 Success rates for the SHA-3 digital MTNLB implementation. ... 83
6.23 Correlation coefficients for 6 key guesses of the SHA-3 analog MTNLB implementation. ......................................................... 85
6.24 3D plot for the SHA-3 analog MTNLB correlation matrix. .......... 85
6.25 Confidence ratios of the SHA-3 analog MTNLB implementation. .... 86
6.26 Confidence ratios of all 40 key bytes of the SHA-3 analog MTNLB implementation. .......................................................... 87
6.27 Success rates for the SHA-3 analog MTNLB implementation. ......... 87
6.28 Correlation coefficients for 6 key guesses of the SHA-3 analog dual-MTNLB implementation. ......................................................... 89
6.29 3D plot of the SHA-3 analog dual-MTNLB correlation matrix. ....... 90
6.30 Confidence ratios for the SHA-3 analog dual-MTNLB implementation. .... 91
6.31 Confidence ratios of all 40 key bytes of the SHA-3 analog dual-MTNLB implementation. .......................................................... 91
6.32 Success rates for the SHA-3 analog dual-MTNLB implementation. ...... 92
Chapter 1

Motivation

Messages received from soldiers in the field could call in artillery strikes, call for emergency medical evacuations, and signal an area safe for others. Cryptographers have done research in ensuring cryptographic algorithms are computationally safe; yet, information leaked from physical implementations is used to compromise security without dealing with the mathematical strength or weakness of an algorithm, leading to destruction on the battlefield.

These leakages have brought on a class of analysis that extracts secret information with far less computational power than brute force guessing. This has the implication of breaking any current, or future, cryptography algorithms if weaknesses in hardware implementations are not addressed. The problem has been known for decades with no full-proof solution. The first corrective measures involved writing software to mask what operations were currently being performed in the cryptographic algorithm. This masking, through use of the same hardware, but different software implementations, has been shown effective at preventing a simple set of the power analysis attacks. This masking at the software level shows almost no resilience to more modern and elegant power analysis strategies. Recently, many hardware designs have been proposed that show resistance against different forms of power analysis by using different hardware layouts. These designs drastically reduce the effectiveness of power analysis by changing the power profile of the hardware implementation; generally, by the use of dual-rail logic, to increases the number of power traces to mount a successful attack. Decreasing attack feasibility makes a cryptographic algorithm more resistant to side channel attacks; however, as devices become more efficient and the
level of computations increase, the same problem will show itself again because the solutions are all based on a platform, CMOS, that is causing the issue. Dual-rail logic also comes at the cost of higher area and power overheads.

Memristors are a basic electrical two terminal component where internal resistance is based on the history of the previous current that has flowed through it. Memristors are simple devices made out of thin film allowing high scalability to lower technology nodes. Their ‘memory’, or state, component makes them good candidates in neuromorphic designs that generally require some sort of memory component. Neuromorphic designs are those that attempt to recreate, or are inspired by, the biological processes in our brains to perform computations that are very expensive on traditional computing platforms. These have been broken down into neurons, the element that receives inputs and creates outputs, and synapses, the elements in connecting neurons together. The memristor works as the synapse by holding a value representing a weight for a single connection. This weight describes how connected the two neurons are to each other. To determine these weights, training is done on the circuit until it gives desirable results using some sort of cost reduction algorithm.

Neural Logic Blocks (NLBs) use these memristive devices in a network that is analogous to reconfigurable blocks in a Field Programmable Gate Array (FPGA). The network is able to be configured in such a way to allow different logic functions to be trained based on selection of synaptic weights. These blocks could be implemented in an FPGA instead of current configurable logic blocks (CLBs) to allow the properties of low power consumption with stochastic operation for secure implementations of any cryptographic algorithm. Another benefit comes with the compact design of the NLBs, as they use less transistors than current look-up table (LUT) based FPGAs.

The document is broken down as the following: Chapter 2 will focus on background information required to understand the work done, Chapter 3 will cover the supporting work used, Chapter 4 will discuss the designed circuits, Chapter 5 will show the design flow used for power analysis, Chapter 6 will document all results from the work, and Chapter 7 will form conclusions over the entire thesis.
Chapter 2

Background

This section will cover a description of SCA, cryptographic hash functions, applications of hash functions, a specific hash function called SHA-3 and the basic concepts of neuromorphic design that will be used for mitigations.

2.1 Side Channel Attacks (SCA)

SCAs are a category of attacks, in the cryptographic domain, that take measurable data from hardware implementations of algorithms and draw correlation to the algorithm’s input. This data can be time [20], power [10], radiation [11] or any phenomenon that is measurable. Fig. 2.1 shows the flow of mounting a SCA. Private data is given to a cryptographic algorithm that provides some level of security by using mathematical operations. The attacker collects leakage data using sensors, oscilloscopes, timers, etc. The attacker then uses various techniques to try and figure out what secret data was processed. These techniques will be discussed later as they apply to power analysis. SCAs come at the risk of making secure algorithms insecure with, in some cases, minimal energy and time. SCA on cryptographic algorithms will be studied as applied to cryptographic hash functions.

2.2 Cryptographic Hash Functions

Cryptographic hash functions are a stricter subset of standard hash functions that are similar to fast data processing functions. A hash function takes a message of any length as an input and then outputs a set number of bits as seen in Fig. 2.2. Where cryptographic hash functions differ is in the three properties that must be satisfied to ensure the security of the
Figure 2.1: Side Channel Attack methodology
output. These properties are [17]:

- Pre-image resistance: given $H(M_i)$, find $M_i$
- Second pre-image resistance: given $M_1$ find $M_2$ such that $H(M_1) = H(M_2)$
- Collision resistance: find $M_1$ and $M_2$ such that $H(M_1) = H(M_2)$

These three properties should be hard to break for an algorithm to be considered a secure cryptographic hash function. Hard to break is defined by how many combinations of the output bits, $n$, need to be tested before one of the conditions above is found. Pre-image and second pre-image resistance should be $2^n$ while collision resistance is $2^{n/2}$. With $n$ chosen to be of large enough, brute force searches become unfeasible on current computing platforms. Cryptographic hash functions are used in many applications such as: unique message signatures, authenticated encryption, password databases, etc.

### 2.3 Message Authentication Codes (MACs)

Secure hash functions have many uses when two parties wish to know data has not been tampered with. This is commonly seen on the internet when a file download has a checksum listed next to it. The checksum is obtained by running the entire file through a hash function and generating a message digest. When the user downloads the file, they run the file through the same hash function and compare their obtained digest with the listed checksum. If the
hash function is truly hard to invert, by following the previously defined properties, the user should be confident the file has not been tampered with.

Another use case for secure hashing is in password databases. If a database stored usernames linked to passwords, an attacker would only need to steal the database to have access to all of the user accounts. To mitigate this, secure databases will only store the hash of a password linked to the usernames. When a user enters his correct password, software will hash the entered password and compare it to the value in the database. If the hacker obtains the data in the database, they will only have hashes of the correct password. As long as the pre-image condition described holds, the attacker would have to brute force search $2^n$ messages to find a message that hashes to the same value for access to a user account. This also means that there exists multiple passwords that can be used to log into a user account, but the property of second pre-image resistance requires finding this to be hard, i.e. $2^n$.

The use case that will be explored in this work will be hash functions used for MAC implementations. MACs are used to determine if the message sent from one party came from that party. Referring to Fig. 2.3, assume the scenario where Bob wants to send Alice a message, where Bob will be a real user and Alice represents a web server. The first step involves Bob and Alice both agreeing on a secret key to use by way of public key cryptography. Public key cryptography is beyond the scope of this work so it will be assumed that the keys have been properly established and only Bob and Alice know them. When Bob wants to send a request to Alice, Bob will first send his message through a hash function to generate a digest/tag. Bob will then send both the message and the tag to Alice. Upon receipt, Alice will take Bob’s message and put it through the same hash function to generate a digest/tag. Alice will then compare the generated tag to Bob’s sent tag. If the tags match Alice can assume that the message came from Bob [17]. In this case, no encryption is being done. The key is being used as the secret information that will provide a unique signature that could only be performed by messages originating from Bob or Alice. This can be further applied to authenticated encryption where SHA-3, the Keccak permutation, is used as a base [3], [4].
2.4 Merkle-Damgard Vs. Sponge Hash Construction

Hash functions are typically built on algorithms that have a set input and a set output. Since a hash function is desired to have input of variable size, a step is required to ensure the input message can be fully processed by the algorithm. There are two dominant theories for construction: Merkle-Damgard and sponge. Merkle-Damgard uses an idea of chaining inputs shown in Fig. 2.4. The message is broken into $N$ blocks based on a multiple of the defined hash function input and then padded so all inputs to the hash functions are full. For the first message block, the block is sent to a hash function with some initialization vector. The output is sent to the next hash function that also takes the next message block. The multiple inputs to the hash function are XOR’d together to provide a single input to the hash function. This continues until every message block has been processed in the hash function. The final hash function output will be the digest.

This works well for solving the variable input problem, but structural weakness in this construction have been found which reduces the strength of the pre-image, second pre-image, and collision resistant properties. These weakness are:
• Length Extension Attack

• Multi-collision attacks

• Herding attacks

Length extension attacks show up when any Merkle-Damgard based MAC construction uses $H(\text{secretKey}||p(message))$. The attacker only needs to generate

$$H(\text{secretKey}||p(message)||p(\text{controlledMessage}))$$  \hspace{1cm} (2.1)

The attacker never learns the secret key, and does not have access to the original message. The hash digest gives the attacker the last state of the hash function that can be used to generate a new message. The attacker can then compute a new digest from the previous state with his appended message. This will result in a message and digest that will look to be valid from another user/server.

Multi-collision and herding attacks follow a similar strategy. The attacker tries to find a collision on one of the hash outputs. Once this collision is known, the attacker can build an infinite number of messages that all hash to the same value. This is dangerous if one thinks about how messages are sent over the internet with standard header information. An attacker just needs to find a collision at the end of the header and can then generate any type of message. There would be no way for the other party to distinguish that the original message has been tampered with. [17]

Sponge construction involves the concepts of absorbing and of squeezing to solve the
structural problems found in Merkle-Damgard construction [5]. Fig. 2.5 shows the structure of a sponge function when applied to the Keccak hash permutation. The message is split into chunks similar to the Merkle-Damgard construction. These chunks are absorbed in the same manner by XORing the previous state and the current message block and running it through the hash permutation, $f$. The difference is the portion of the state, $c$, that is internal to the algorithm as is not the case in the Merkle-Damgard approach. It is impossible to know the final state of the hash construction as only a portion of the state, $r$, is used as output. Sponge construction also brings the benefit of variable output as the construction can continuously be squeezed to generate more output bits. These properties solve the weaknesses of the Merkle-Damgard construction mentioned above.

This work investigates mitigations of SCA with power analysis on the target implementation of SHA-3. SHA-3 is a new standard hash function, making it a good target to explore mitigations on before weaknesses are exploited.

### 2.5 SHA-3

The National Institute of Standards and Technology (NIST) is a government funded agency that has the responsibility of establishing and keeping standards in many application domains, which include cryptography. Of their standardized algorithms, they have released a family of hash functions called secure hash algorithm (SHA). In 2007, NIST held a competition looking for the next SHA algorithm, SHA-3. This came as SHA-0 had been broken
and SHA-1 had theoretical breaks. SHA-2 had no known theoretical breaks, but had a similar construction to SHA-1. This fueled a desire for a new SHA algorithm. SHA-3 was to be diversified from the previous algorithms so no theoretical attacks developed for SHA-2 could be easily modified for SHA-3. Keccak was chosen as the winner of the SHA-3 competition on Oct 27, 2012. In 2014 NIST published a draft federal information processing standards (FIPS) publication outlining the standard implementation for SHA-3.

Fig. 2.5 shows the construction of Keccak. Keccak uses a sponge construction where every hash operation is denoted by an \( f \) in the figure. Keccak is then defined by specifying the values of \( r \) and \( c \). \( r \) refers to the rate of the algorithm and is the speed that messages can be consumed and digests generated. \( c \) refers to the internal capacity and is never propagated as an output, but controls the level of security of the algorithm. The authors of Keccak prove that the strength of Keccak is equal to \( 2^{c/2} \) [7]. One Keccak hash permutations is broken into \( 12 + 2\ell \) rounds where \( \ell \) is found from 2.2 and \( w \) is the size of a word in the implementation.

\[
w = 2^\ell
\]  

For simplicity in comparison, the suggested Keccak implementation in FIPS 202 [15] will be used which has \( r = 1024 \), \( c = 576 \), and \( w = 64 \). This means that the Keccak permutation will have 24 rounds for any block of input that needs to be absorbed\(^1\). The SCA mitigations discussed in this work are for securing the Keccak permutation to provide security to SHA-3 and any other algorithm that uses the Keccak permutation as a base.

Keccak focuses on manipulation of an internal state at the bit level. The Keccak authors use a set of terminology when talking about the Keccak permutation that can be seen in Fig. 2.6. A Keccak state is \( 5 \times 5 \times w \) bits, or \( 5 \times 5 \times 64 \) in this case. The state is filled starting from \((0,0,0)\) along the z-axis until full, then along the x-axis, and finally the y-axis. Once \( r \) bits have been filled, the remaining state is initialized to ‘0’ to represent \( c \). Similar to chaining algorithms, there must be a padding function that makes sure the input data is a multiple of \( r \). The padding function used in SHA-3 is a \( 10 \times 1 \) structure where the bits ‘1’,\(^1\)

---

\(^1\)Keccak will be referred to as SHA-3 throughout this work when the full algorithm is being addressed. Keccak will be referred to as Keccak when the specific permutation is being discussed.
1 to \( r-1 \) ‘0’s, and a final ‘1’ bit are appended to the message to ensure it is a multiple of \( r \).

Fig. 2.7 shows how the state will be filled for experiments in this work. The bottom plane will contain a 320-bit key, followed by two more planes (640-bit) and one lane (64-bit) of random input. The key will be, in hex format: \{00,01,02,03,...,39\}.

The Keccak permutation then can be describe in 6 distinct operations: \textit{State XOR}, \textit{theta(\( \theta \))}, \textit{rho(\( \rho \))}, \textit{pi(\( \pi \))}, \textit{chi(\( \chi \))}, and \textit{iota(\( \iota \))}.

### 2.5.1 State XOR

The State XOR involves filling a Keccak state and then performing the bit-wise XOR operation between the current state and the previous state. In the case of the first state, the previous state is a Keccak state with all ‘0’ bit entries. This provides the absorption property of the sponge construction. Power analysis at this state is not a primary target as the initial state and the input are known. This would be a good attack point if a random initialization was used for the state, instead of the FIPS recommended all zeros state, to identify the random initialization.
2.5.2 Theta (θ)

θ provides column diffusion by calculating each column’s parity and using an XOR function to combine two neighbor columns. This is defined by:

\[ S[x][y][z] = s[x][y][z] \oplus (\oplus_{i=0}^{4} (s[x-1][i][z]) \oplus (\oplus_{i=0}^{4} (s[x+1][i][z-1])) \] (2.3)

To reduce hardware complexity of having an 11-bit XOR operation for every output bit, θ is typically implemented in two phases: the θ plane creation and the θ\_out phases. The θ plane creation follows:

\[ \theta_{plane}(x, z) = \oplus_{y=0}^{4} (s[x][y][z]) \] (2.4)

The θ plane is created with 5-bit XOR operations. The θ\_out phase can be implemented with 3-bit XOR operations as follows:

\[ \theta_{out}(x, y, z) = s[x][y][z] \oplus \theta_{plane}(s[x-1][i][z]) \] \[ \oplus \theta_{plane}(s[x+1][i][z-1]) \] (2.5)

Fig. 2.8 show a visual representation of the state operation. The output at a single bit in the state is the current bit and the 10 bits of two neighboring columns. In hardware, this is typically done in two stages. 320 5-bit XORs convert the columns into a 5x64 plane. 1600 3-bit XORs take an initial bit and 2-bits from the θ plane to create the output of θ. This is a good SCA attack point as there is a mixing of known and unknown bits in the creation of the θ plane and θ\_out.

2.5.3 Rho (ρ)

ρ provides inter-slice diffusion inside the state. This is done by rotating the lanes by an offset depending on the lane’s x and y coordinates. The mathematical formula for this
operation is:

\[ S[x][y][z] = s[x][y][z - (t + 1)(t + 2)/2], \]

where \( t = \begin{cases} 
(0 \ 1 \ 2 \ 3)^t \cdot (1 \ 0) = (x \ y) & 0 \leq t < 24 \\
-1 & x = y = 0
\end{cases} \) \hspace{1cm} (2.6)

Fig. 2.9 shows a visual representation of the \( \rho \) permutation. Table 2.1 shows the offsets that are used for each of the lane rotations that follow 2.6. The permutation in hardware can be done by remapping using wires, not provide useful power data for a SCA.
2.5.4  **Pi (π)**  

π provides long term diffusion by permuting the lanes to different locations in the state. π is mathematically defined as a multiplication of the current state by a matrix in GF(5) as follows:

\[
S[x,y] = s[x,y] \times \begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix}^T
\]  

(2.7)

The entire lane, values along z-axis, moves to the new (x, y) coordinate. Also, the permutation defines the center state as (x, y) = (0, 0), where as during state filling, the bottom left bit has (x = y = 0). Fig. 2.10 shows a visual representation of π. The permutation is broken down into 6 subfigures where the movement happens about the center (0, 0) point. Since π is just a remapping of bits, this is a bad SCA target for the same reason as mentioned for ρ.

2.5.5  **Chi (χ)**  

χ provides the only non-linear function of Keccak, important so Keccak is not vulnerable to linear cryptographic analysis. χ can be thought of as 5-bit, 5w parallel S-boxes operating on the rows of the Keccak state. It is mathematically defined as:

\[
S[x,y] = s[x,y] \oplus \text{NOT}(s[x+1],y)s[x+2,y])
\]  

(2.8)

Fig. 2.11 shows a visual representation of the χ permutation involving one XOR, one AND, and one NOT gate per state bit. This non-linear operation was chosen by the authors for its low gate count. This is a great candidate as a target for SCAs as each 5-bit row can
Figure 2.10: $\pi$ permutation of Keccak state [7]

Figure 2.11: $\chi$ permutation of Keccak state
Table 2.2: $\iota$ round constants

| RC[ 0] | 0x0000000000000001 | RC[12] | 0x000000008000808B |
| RC[ 1] | 0x0000000000008082 | RC[13] | 0x000000008000808B |
| RC[ 2] | 0x8000000000000001 | RC[14] | 0x8000000000008089 |
| RC[ 3] | 0x0000000000008009 | RC[15] | 0x800000000000008B |
| RC[ 4] | 0x0000000000000001 | RC[16] | 0x8000000000000080 |
| RC[ 5] | 0x000000000000000A | RC[17] | 0x8000000000000080 |
| RC[ 6] | 0x0000000000000000 | RC[18] | 0x0000000000000000 |
| RC[ 7] | 0x0000000000000000 | RC[19] | 0x0000000000000000 |
| RC[ 8] | 0x0000000000000000 | RC[20] | 0x0000000000000000 |
| RC[ 9] | 0x0000000000000000 | RC[21] | 0x0000000000000000 |
| RC[10] | 0x0000000000000000 | RC[22] | 0x0000000000000000 |
| RC[11] | 0x0000000000000000 | RC[23] | 0x0000000000000000 |

be thought of as a 5-bit S-Box, the typical attack point in AES. [27] shows when keys get larger than one plane, $\chi$ is a required attack point for finding the full key. Also, they show using a larger key gives the best security, with keys that are multiples of $r$ being the most effective. In this work, $\chi$ will not be looked at as a SCA analysis point as it is not needed for keys that reside only in the first Keccak plane. If mitigations work for keys residing in a single plane, they are inherently valid for the more complicated cases.

2.5.6 Iota ($\iota$)

$\iota$ is the last permutation of the Keccak hash function. A constant value is XOR’d with the first lane $(0, 0)$. This constant value is different for every round making each Keccak round different. The round constant comes from a 64-bit linear feedback shift register (LFSR). This mitigates the chance of the Keccak state entering a cycle. A cycle, in the context of a cryptographic algorithm, is when a previous round output is generated by a future round. The consequence is that the rounds in the middle have no impact on the algorithm, greatly reducing security. $\chi$ and $\theta$ ensure the value from $\iota$ is mixed into the state the following round. Table 2.2 shows the round constants for the 24 rounds of Keccak. $\iota$ has 64 unknown values and 64 known values. This could be used as a SCA analysis point, but only for the first 64-bits of the key. If the key is longer then 64-bits, which is recommended, another part of Keccak would also need to be attacked.
For SCA mitigations, many groups have looked at using different digital CMOS strategies to mitigate attacks with no clear umbrella solution. The neuromorphic computing paradigm implements logic much different than traditional AND, OR, etc gates by using inspiration and abstractions from the human brain. Analog circuits, that are more susceptible to noise, could provide enough randomness to help decorrelate leakage data in a circuit to the input being processed.

2.6 Neuromorphic Computing

Neuromorphic computing is a field that has groups who wish to mimic how the brain operates or build systems that take inspiration from abstractions of the brain. Our current Von Neumann computing architectures are very efficient and reliable when it comes to typical number crunching; however, the hard to solve problems, that make up the computer vision and machine intelligence domains, are not made easier from number crunching. Even though a computer can compute many Fourier transforms a second, a task that we could never hope to achieve, it can be near impossible for a computer to distinguish a cat from a dog [18].

This work does not attempt to implement neuromorphic computing components in hopes of making a system that can solve problems similar to the brain. It will attempt
to implement neuromorphic computing components to take advantage of some of their inherent properties. The brain with its approximate 20 billion neurons has single neurons connected to 10,000 other neurons. These connections are called synapses, and it is believed that the connections, or lack of connections, is the result of learning and forgetting. Signals propagate through the brain when these neurons are stimulated from the amount of input they receive. The important properties from the brain for this implementation are its low power, approximately 20 W for the entire brain, and its stochastic nature of operations, thanks to the synaptic connections. Fig. 2.12 shows the model that will be used, a single neuron with 5 synaptic connections. An activation function, described in section 3.3, will determine when the neuron should be active. Equation 2.9 describes this neuron model where $X$ is the input, $W$ is the corresponding weight and $Y$ is the output after applying some activation function. Synaptic weights will be used to mimic the concept of plasticity in the brain where connections to the neuron can be strengthened or weakened depending on what a desired output should be. This can be implemented in memory units in digital design, but is not easily scalable if network size becomes too large. Memristor devices, in the analog domain, can be used to provide for many adjustable states in simple two-terminal devices.

$$Y = activation\left(\sum_{n=1}^{5} X_n \times W_n\right)$$  \hspace{1cm} (2.9)

### 2.7 Memristor Devices

Memristor devices link charge and magnetic flux [28]. They are thin film, two-terminal, passive circuit devices that are known for their pinched hysteresis loop shown in Fig. 2.13. When the memristance of the device is held constant, it acts as a simple resistor following the relation:

$$I = \frac{V}{M}$$  \hspace{1cm} (2.10)

\footnote{For more detailed information, [16] goes more in-depth about computational neuromorphic computing as only a few basic properties are desired in this implementation.}
The output strength is configured by adjusting the property $M$ allowing for the concept of a synaptic weight to be realized with one device. This device allows for highly scalable architectures due to the simple two-terminal nature of the memristor. For digital design, this would require a memory unit for every synaptic weight bit of precision. The calculation of the synaptic weight and input is done by the differing amount of current that can flow through the memristor based on its configured resistance. The summation step, in the analog domain, is done by a shared memristor node. This leads to a massive resource savings as flip-flops to store weights, multipliers to calculate synapse responses, and adders to sum synaptic outputs, required in the digital domain, are not used.
Chapter 3

Supporting Work

3.1 SCA with Power Analysis

SCA can consist of analyzing power consumption [10], electromagnetic radiation [11], timing information [20], cache hits/misses, etc. For this work, only power analysis will be explored as it relates to simple power analysis (SPA), differential power analysis (DPA), and correlation power analysis (CPA). Power analysis is made possible because of the transistor properties in CMOS design. When a transistor transitions to a logic ‘0’ or to a logic ‘1’, it consumes power. The power has been shown to be consistent over multiple similar operations by an experiment done by Paul Kocher [10]. Fig. 3.1 shows the distribution between of power consumed for an AES S-box operation. The least significant bit (LSB) of the S-box output, an 8-bit value, was targeted. The power consumed for 16,000 traces was separated into two bins based on if the LSB was ‘1’ or if it was ‘0’. The figure shows that the mean distribution when the LSB was ‘1’ is different than that of when it was ‘0’. Power analysis tries to find patterns between this characteristic power consumption and the input being processed of a specific cryptographic algorithm.

3.1.1 Simple Power Analysis (SPA)

SPA attempts to decipher what is going on inside a hardware implementation based on how much power is consumed in a circuit from a single waveform. Fig 3.2 shows a power trace from an implementation of Rivest-Shamir-Adleman (RSA), one of the first public key encryption algorithms. In [10], it is shown that by looking for the peaks at each clock edge, the RSA key can be obtained by noting that a large peak corresponds to a ‘1’ and a small peak corresponds to a ‘0’. The peaks have to do with different mathematical operations
in RSA that are dependent on the current bit (‘1’ has more operations than ‘0’). This is a dangerous problem as a secret key can be found by human inspection in a matter of seconds with no required knowledge of the input or output to the system.

SPA can show features in a power trace to indicate when certain operations are happening inside a cryptographic system and how long they are taking. Once features can be determined in power traces, an attacker can use these features to help extract key information; however, SPA is generally not used to attack current cryptographic implementations as it is very easy to counter. A small amount of noise in the system may be enough to mask the signal to where it is not easily deciphered from inspection. Also, this noise can make attack automation hard when trying to determine if a certain feature is present or not. With a single trace, it is hard to reduce noise, which is where DPA comes in.

Fig. 3.3 shows a single SHA-3 power trace. From this power trace, it is possible to see the cycles that are used to load the data, the cycles are used for the permutation rounds, and the cycles are used to generate output. This can facilitate reducing the search space of a waveform for an attack; yet, the data of interest to determine the secret key resides in the variations of the waveform peaks. It can require thousands of traces and statistical analysis to find correlations that map algorithm input to power consumption that is not possible
Figure 3.2: Power trace from an implementation of RSA where the secret key can be deduced from a single trace. Red represents the parts of the waveform that leak the bit is ‘1’, green represents the parts that leak the bit is ‘0’ [10]

Figure 3.3: Power trace for one SHA-3 permutation including the input buffering, SHA-3 permutation and output generation
from a single trace.

### 3.1.2 Differential Power Analysis (DPA)

DPA was developed in [10] to provide a more robust way to find secret keys that is resistant to noise and can easily be automated. The following shows the mathematical formula for a DPA attack:

\[
\Delta D_j[t] = \frac{\sum_{i=1}^{m} S(M_i, K_j)P_i[t]}{\sum_{i=1}^{m} S(M_i, K_j)} - \frac{\sum_{i=1}^{m} (1 - S(M_i, K_j))P_i[t]}{\sum_{i=1}^{m} (1 - S(M_i, K_j))}
\]  

(3.1)

Fig 3.4 shows a visual model for the attack strategy using 3.1. Many input messages \( M \), broken into individual messages \( M_i \), are created and given to a cryptographic implementation. Power traces \( P \), broken into individual power traces \( P_i \), for each input vector are then collected. The secret key is broken up in sections depending specifically on what makes sense for the algorithm under attack; generally, it is the size of the nonlinear permutation to make calculations easier. For this example, assume the key, \( K \), is 128-bit and is broken into 8-bit guesses, \( K_j \). Each key section has 256 key guesses that range from 0 to 255. For each key guess, the attack will attempt to find the output \( I_{ij} \), where \( I_{ij} = S(M_i, K_j) \) and \( S(\ldots) \) is a selection function that determines if a bit under attack is a ‘1’ or ‘0’. Once the attacker has this, they can target one bit of \( I_{ij} \) and divide all power traces into two bins based on if the target bit is a ‘1’ or a ‘0’. From Fig 3.1, if the current guess is right, it should correctly split the power traces into groups that represent the power consumption in the circuit. If it is an incorrect guess, it should split the power traces randomly. Once the attacker has a bin for output ‘0’ and ‘1’, they average the bins and then take the difference of them. Because of the previous found result that the power consumption differs depending on the value of the bits, the correct differential trace should be the one that has the largest peak. Differential traces from wrong guesses should approach zeros with enough traces in the attack. This is because the sample mean of a random distribution, over enough samples, should converge to the distribution mean; thus making the difference of two different random samples of one distribution head toward zero. Fig. 3.5 shows a sample of power traces from an attack on AES.
Figure 3.4: Differential power analysis attack model

Figure 3.5: Differential power analysis on AES key showing guess 1,2,3,4,5, where guess 3 is the correct key byte [10]
Figure 3.6: An example of a non-linear’s and linear function’s impact on output bits for key guesses of 0x05 and 0x85. The top examples are for SHA-3 (linear) while the bottom examples are for AES (non-linear). Light red indicates the portion of the operation that is controlled by the attacker for generating outputs to the selection function (key guess). The dark red numbers represent the bits that can affect the dark red block, the target bit in DPA.

DPA is insensitive to noise because of differential traces. The more traces used, the more likely any random noise will be removed during the differential subtraction step. This attack works well when it is used on non-linear portions of a cryptographic algorithm because small changes in the input of the non-linear portions should have a large effect on the output. For algorithms where the best attack points are linear, problems with the DPA method can be seen.

An example showing a comparison of DPA between linear and non-linear functions can be seen in Fig. 3.6. This shows the linear SHA-3 operation of the θ plane creation versus the non-linear S-box function of AES, which has an XORing of a round key before the function. Both operations use an XOR function of some known data and some secret key data. A key of 0x05 and 0x85 are shown together to see the different effects on the output with the change of a single bit. On the linear SHA-3 permutation, changing a bit in a key
guess can only impact the same bit in the output. In the AES case, changing any bit in the input to the non-linear function can have an impact on any output bit. This is important as DPA targets certain bits of the output. If only the LSB is considered, the impact of the the plain text bits from the MSB to LSBI are thrown away when figuring out the bin to put the waveform in for the SHA-3 case. This throws out the differing power of the transistors in the circuit that do not directly effect the target bit, most bits in the design. As such, the non-linear function can be seen as a function that is providing some amount of information about all bits that enter it, while linear functions need some other methodology to get this information.

3.1.3 Correlation Power Analysis (CPA)

CPA was first described in [8]. It is based off the Pearson correlation coefficient that determines the relationship between how two random variables change. CPA works because it has been shown that there is a strong correlation between the transistors that have transitioned and power consumed than just the current state of the transistors (as in DPA). When the number of transistors set is used, this is called hamming weight. When the number of transistors that have changed value is used, this is called hamming distance. Hamming weight and hamming distance are referred to as power models of CMOS circuits, and allow for characterization of a circuit to be reduced to correlation between its power consumed and one of these models. CPA tries to find the correlation between power consumed and either the hamming weight or hamming distance of the input and output of the algorithm. The lower the magnitude of the correlation coefficient, the less likely a key guess is the correct key.

The Pearson correlation coefficient is shown in 3.2. The Pearson correlation coefficient is the covariance of the data, power traces and hamming distances/weights, divided by the variance of the two distributions.

\[ \rho = \frac{\text{cov}(x, y)}{\sigma_x \sigma_y} \]  \hspace{1cm} (3.2)

For linear functions, CPA solves the problem of DPA involving a lack of correlation between bits in a section of attack. CPA takes into account the state of all bits for each
attack block as the hamming weight or hamming distance is the metric that is compared to power data, instead of the state of an individual bit.

Since this work will focus primarily on mitigations to current implementations of CMOS designs in the cryptographic context, it is important to discuss the leading proposed solutions. The majority of proposed mitigations use the same CMOS digital design principles to reduce power leakages that are at the root cause of the problem. A neuromorphic design in the analog design domain will be proposed as a possible competing solution.

3.2 Current Mitigating Designs

3.2.1 Dual-rail with Precharge Logic

Dual rail with pre-charge logic is a family of mitigation designs that split transistor operation into a pre-charge and evaluation phase with the additional use of complementary logic. The pre-charge phase is meant to set the circuit to a known electrical state before any bit transitions happen, eliminating the use of the hamming distance power model. The complementary logic is meant to mask the power of the number of bits set in the cryptographic algorithm, eliminating the use of the hamming weight power model. The evaluation phase is the normal phase where a bit becomes a logic ‘1’ or logic ‘0’.

Wave Dynamic Differential Logic (WDDL) attempts to hide power consumption by replicating the inverse of the circuit so that every bit transition from ‘1’ to ‘0’, there also exists a ‘0’ to ‘1’. Fig. 3.7 shows an example of a WDDL AND gate. This was tested on an FPGA [9] and found effective against DPA. The authors suspected that more sensitive measurement equipment might be able to mount a successful SCA. Shortly after, a new attack that focuses on a condition called the Early Evaluation (EE) effect was developed that exploits a condition where two inputs to a logic gate arrive at different times [26]. Assume the case where the values of A and B are set to ‘1’ in Fig. 3.7. Initially, all inputs to the two gates are ‘0’, pre-charge, setting the output of both gates to logic ‘0’. During the evaluation phase, if A and B arrive at some $\triangle t$ time different from each other and $\bar{A}$ and $\bar{B}$ arrive at the same time, the OR gate would transition to its correct state of logic ‘0’ $\triangle t$ time before the AND gate. This defeats the purpose of the complimentary logic and is not
a simple problem to solve as it would require exact capacitance matching in VLSI layout.

Masked Dual-rail Pre-charge Logic (MDPL) attempts to avoid the need for perfect balancing of wires inside the design by masking every value and pre-charging cells [19]. This comes at the cost of increased power consumption and area overhead as much as a factor of 4. The design was shown to run slower than WDDL; yet, it does not have the noticeable attack vulnerabilities of WDDL and does not require capacitance balanced wires. This makes for an easier implementation, but at the cost of higher power consumption and much larger area.

Secure Triple Track Logic (STTL) solves the problem of EE by adding a synchronization signal to the pre-charge/evaluation signal to prevent evaluations until all signals are valid [23]. It requires a synchronization signal that is slower than the dual-rail logic delay providing a design that runs slower for cases that would not normally need synchronization. The benefit is that this logic style can be considered completely dual-rail.

Balanced Cell-based Differential Logic (BCDL) is the final design that will be discussed in this family of mitigations that improves on STTL by using a fast global synchronizing signal [14]. This design is still slower than MDL and WDDL but faster than STTL.

There are also many variations on the WDDL implementation that will not be discussed here. They can be seen to improve on the basic WDDL flaws, but not at a level that would justify hardware development costs for implementations.
3.2.2 Software level

[2] introduces a way to mask power consumption by using random code not correlated to the cryptographic operation being performed. Code is analyzed to determine blocks that contain cryptographic calculations and then random code is inserted into these blocks. This was shown to be effective against DPA but at the cost of 30% longer run times and 27% higher power consumption. Also, these designs run on processors that lose the benefit of fast gate-level algorithm designs.

All the presented mitigation designs make minor modifications to the current design paradigm of digital CMOS logic with varied success. It stands to reason that stepping out of the digital CMOS design context, that has the inherent properties allowing information leakage, may help to find circuits that have strengths against power analysis.

3.3 Neural Logic Block (NLB)

The architecture that will be explored in this work is the NLB. Specifically, the NLB of focus is the Multi-threshold NLB (MTNLB) proposed in [24]. An NLB is a component that uses the computational neuromorphic computing principles discussed in section 2.6. The NLB is a single layer component capable of learning any basic logic function: AND, NOT, OR, XOR, XNOR, NAND, NOR, and any n input functions. What is important to note, XOR/XNOR are examples of operations that are not linearly separable. An example of this linear separable concept is shown in Fig. 3.8 with a 2-input AND gate compared to a 2-input XOR gate. The AND gate has one decision boundary, when both inputs are high. The XOR gate was two decision boundaries: when both inputs are high and when both inputs are low. The AND gate could be represented using a single layer neuron that learns the function of the hyperplane; however, the XOR gate would need two neurons that learned the described hyperplanes and a second level that combined them to a single result. Since the NLB does not use simple monotonic activation functions, it can implement any n input function as long as the activation function has n + 1 boundaries. This component can be looked at as a trainable, analog, look up table (LUT) that is analogues to the LUTs
Figure 3.8: Separability of AND and XOR logic functions. The green circle with an X represents when
the output is desired to be a one while the red circle represents when the output should be zero.

used in FPGAs.

The non-monotonic activation function is shown in Fig. 3.9 with the left plots representing
training without a bias signal and the right plots with a bias signal. AND and OR gates only have one
decision boundary, where all inputs are logic ‘1’ for AND or when one input is logic ‘1’ for OR. For the XOR function, an \(n\)-bit XOR function requires \(n+1\) boundaries as the output toggles based on the number of inputs that are active. The same is true for NAND, NOR and XNOR except now it is desirable for the output to have a ‘1’ when all inputs are ‘0’. For a generic activation, the bottom plots show how scaling the values of the
thresholds can let one activation function map to many different logic functions.

Because of the random weight initialization with training, the amount of power consumed for any \(n\)-inputs will be inherently different, as there are multiple weight combinations that could generate the same desired output [24]. The goal is to mask the power dissipation by adaptive use of the MTNLB. The stochastic nature of the synapse weights in the MTNLB network, with bipolar weights, will provide variability in the power traces; moreover, there is a significant reduction in the power consumption with the MTNLB compared to standard lookup table or CLB logic, as shown in Fig. 3.10. This is important as smaller power fluctuations in a power trace will cause more traces to form correlations. Lastly, it is also shown that the MTNLB, when implemented in analog-digital mixed signal design, requires less transistor resources than an equivalent performing LUT design used
Figure 3.9: Plot of activation functions for AND, OR, XOR, NAND, NOR and XNOR functions. The top plots are the desired activation for each labeled gate. The bottom plots shown how the activation functions can be realized with one standard activation function. No bias is shown on the left and using a bias is shown on the right [24].

Figure 3.10: Energy delay product for the proposed NLB designs compared to that of a standard LUT [24]. The MTNLB is the only implementation style that will be used in this work, but power data is provided for the ANLB and RANLB.
in an FPGA. This means that MTNLBs could be used to replace, or compliment, LUTs in an FPGA to provide SCA mitigation to any implemented cryptographic algorithm. The characteristics that are hoped to be exploited in the MTNLB to produce a SCA resistant circuit are:

- Inherent device variability of the memristor (process variations)
- Ultra low power design (several orders of magnitude less)
- Stochastic power consumption based on the trained values of memristor (unique to each NLB)
- Less required resources (Less resources used to implement MTNLB compared to LUT).
Chapter 4

Hardware Circuits

The circuits in this work are split into two groups: digital and analog. The digital section circuits implement the full SHA-3 algorithm, while the analog circuits only implement the  plane due to the length of simulation. To provide accurate comparisons between the analog circuits that have only  implemented to the full digital implementations, all circuits discussed in section 4.1 will have another circuit with only  implemented. The digital  only circuits will not be discussed as they are a subcircuit of the presented circuits.

4.1 Digital Implementations

4.1.1 High-Speed Core - Base Circuit

For analysis of the SHA-3 algorithm, the SHA-3 high-speed core implementation provided by the SHA-3 authors was used [6]. This HDL model was modified when adding mitigations to provide a base for comparison of mitigations to no mitigation. Fig. 4.1 shows the block level diagram of the high-speed core. Starting from the left, the input into a round is either the previous state, in the case of squeezing, or the previous state XOR’d with a new block of input, in the case of absorbing. The block diagram shows a 5x5x5 cube for simplicity, but the actual state in all circuits will be 5x5x64. From here, the columns of the 1600-bit state are XOR’d to generate a 5x64 plane that is called the  plane. To generate the resulting  output, the bit at \((x, y, z)\) is XOR’d with two-bits in the  plane as discussed in section 2.5.2 to obtain \(\theta_{out}(x, y, z)\). The next phases of  and  are remappings. The  operation takes each input bit and applies a non-linear function comprised of 3 logic gates, AND, XOR, and NOT per bit. The final operation is a 64-bit XOR on the first lane of the state using LFSR generated round constants defined in Table 2.2.
Figure 4.1: Block diagram of the basic SHA-3 high-speed core implementation. The state is represented as a $5 \times 5 \times 5$ cube where as the actual implementation is $5 \times 5 \times 64$. The blue colored gates and state bits show an example of creating one $\theta(x, z)$ plane bit and also the $\iota$ step. The red gates and state bit show an example of how one $\theta_{out}(x, y, z)$ bit is created.

4.1.2 Mitigation One - Dual Core Design

The first circuit for SCA mitigation testing was to use two simultaneously instantiated SHA-3 high-speed cores. The first core would perform the SHA-3 hash algorithm while the second core would compute the SHA-3 hash with inverted input data. This design is inspired from the dual-rail logic family where every ‘0’ to ‘1’ bit transition has a complementary ‘1’ to ‘0’ transition. Since the $\theta$ plane creation uses a linear function, XOR, dual-rail logic can be implemented by using the inverse input data. This is not full dual-rail logic as this does not hold for the $\chi$ permutation. Since the attack is happening on the $\theta$ plane, the methodology is valid. Fig. 4.2 shows the block diagram for this mitigation circuit. Two SHA-3 cores are instantiated and one is fed the normal input data while the second receives the inverse data by using 1600 NOT gates, one for each bit.

4.1.3 Mitigation Two - $\theta$ Plane Masking

For a more compact digital implementation, mitigation two uses the inverse operation on the creation of the $\theta$ plane. This is done by modifying the base SHA-3 core and adding a
Figure 4.2: Block diagram of the SHA-3 dual-core mitigation. The state is represented as a $5 \times 5 \times 5$ cube where as the actual implementation is $5 \times 5 \times 64$. The blue colored gates and state bits show an example of creating one $\theta(x, z)$ plane bit and also the $\iota$ step. The red gates and state bit show an example of how one $\theta_{out}(x, y, z)$ bit is created.
Figure 4.3: Block diagram of the SHA-3 XNOR $\theta$ plane mitigation. The state is represented as a $5 \times 5 \times 5$ cube where as the actual implementation is $5 \times 5 \times 64$. The blue colored gates and state bits show an example of creating one $\theta(x, z)$ plane bit and also the $\iota$ step. The red gates and state bit show an example of how one $\theta_{out}(x, y, z)$ bit is created.

set of XNOR gates to calculate the inverse $\theta$ plane. This saves on $1600 \times 3$ gates that are used in the $\chi$ permutation and the one 64-bit gate used during $\iota$. Fig. 4.3 shows the block diagram of this circuit.

4.1.4 Mitigation Three - NLB $\theta$ Plane Masking

The third mitigation uses the concepts of NLBs discussed in section 3.3. The implementation of MTNLBs presented in previous work has been an analog implementation. This work will implement both an analog and digital MTNLB circuit to compare mitigation in different design domains. Power analysis success results from characteristic power consumption seen in CMOS circuits; thus, implementing the MTNLB in digital logic can show the possibility of decorrelating power consumption from input data while still using full digital CMOS logic.

Fig. 4.4 shows the high level block diagram of this mitigation. The 5-bit XOR gates in the $\theta$ plane creation step are replaced with 5-bit MTNLBs. Fig. 4.5 shows the RTL level description of one MTNLB. Weight values represent the synaptic connections between the
input and the MTNLB. These weights are trained outside of this RTL model using Matlab. This was done as the power consumption of the training phase is not as important as the power consumed during the cryptographic hash computation. The weights are represented using fixed point notation \((Q_m, Q_n)\) where \(Q_m\) is the number of integer bits and \(Q_n\) is the number of fractional bits. The \((Q_m, Q_n)\) used in this implementation is \((3, 4)\) for a total of 8-bits.

Normally the synaptic response between the input and the neuron is found by multiplying the synaptic weight with the corresponding input. This works well in software, but in hardware this would require a \((Q_m, Q_n) \times (Q_m, Q_n)\) multiplier for every input. In this design, there is a need for 320 5-bit MTNLBs resulting in \(320 \times 5\) multiplier units. To simplify this, an 8-bit, 2-1 MUX is used for every synaptic input and uses the XOR input, either ‘0’ or ‘1’, as its select line. If the input is low, it will pass on the value of ‘0’, while if the input is high, it will pass on the value of the synaptic weight. The outputs of these 8-bit, 2-1 MUXs will be summed using an adder tree of \((Q_m, Q_n)\) bit adders. For the neuron activation function, the non-linear function described in section 3.3 will be used. This will be implemented much like the analog version using comparators to determine what range the
synaptic sum falls into. In this case, there are 256 possible values as the fixed-point number is 8-bit. The 256 values are split into 6 sections fulfilling the $n + 1$ decision boundaries requirement to ensure a possible solution. The final output of the MTNLB will be either ‘0’ or ‘1’ dependent on an even boundary being active (active low in this implementation). This is equivalent to the activation function outlined in the MTNLB description.

### 4.2 Analog Implementations

Digital circuits are ineffective at implementing some of the basic features found in neuromorphic design, such as addition, weight storage and synaptic response calculations. The analog implementations were only implemented for the $\theta$ permutation due to simulation time. These circuit will be compared to the corresponding digital $\theta$ only circuit for fair comparisons.

#### 4.2.1 Mitigation Four - NLB $\theta$ Plane Masking - default

The MTNLB can be broken down into three main components, the global trainer, the range-select/local trainer and the activation function as shown in Fig. 4.6.

Fig. 4.7a shows the range-select component. It takes a voltage input and performs the input and synaptic weight multiplication by the current that flows through the memristor.
Figure 4.6: Block level diagram of the analog MTNLB. For one MTNLB there is a single global trainer, \( n \) local trainings, \( n \) memristors, one range select component and one activation function.

Figure 4.7: (a) shows the range select component. It consists of \( n \) local trainers (red), \( n \) memristors, \( n \) comparators (green) and \( \log_2(n) \) NAND gates (blue). (b) shows the local trainer circuit. 'Z' is high impedance. (c) shows the activation circuit. (d) shows the global trainer.
The currents are summed by a shared node instead of the need for adders as in the digital implementation. This summed current is sent to $n$ comparators where the number of comparators is equal to the number of inputs of the MTNLB. The comparators find the range, $m$, that the summed current falls into by having one of the active-low select lines be active. By design, only one $\overline{Sel}_m$ signal is active at a time.

Inside the range-select block are $n$ local trainers whose responsibility is adjusting individual synapse weights, memristors, based on control signals from the global trainer. Fig. 4.7b shows the circuit diagram for the local trainer. The signals $\overline{ClkSel}$, $\overline{m^+}$ and $\overline{m^-}$ come from the global trainer. When $\overline{ClkSel}$ is high, the values of $\overline{m^+}$ and $\overline{m^-}$ are valid for training. If the input, $X_i$, is low, the value of ‘0’ is passed through the MUX and the weight of the memristor is unchanged. If the value of $X_i$ is high, the values of $\overline{m^+}$ and $\overline{m^-}$ are applied to the memristor to change its internal state. $\overline{m^+}$ and $\overline{m^-}$ are set by the global trainer to correctly increment or decrement the memristor state. When $\overline{ClkSel}$ is low, there is no training to be done, and the voltage on the input of the memristor is $X_i$ while the output is summed together with the rest of the memristors.

The activation function takes, as inputs, the active-low select lines from the range select block, $\overline{sel}_i$ and the inputs to the MTNLB, $X_i$. Fig. 4.7c shows the circuit diagram for the activation function. The activation function is a generic function where the desired function is configured by changing the reference currents in the range-select block. To get the non-monotonic function, the even select lines from the range-select block are used as inputs to a NAND gate. This covers the function previously shown in Fig. 3.9 on the left side, but not the plots on the right. To get the alternate behavior of having an active output when all inputs are low, a D-flip-flop is used with the clock signal being the XNOR of the inputs. If all the inputs are low, a D-flip-flop will have the value of $Y_{exp}$ clocked to the output. If $Y_{exp}$ is ‘0’ while all inputs are low, the values of $Y$ and $Y_{comp}$ will be the normal expected outputs. If $Y_{exp}$ is ‘1’ while all the inputs are low, the bias signal will be set to ‘1’. This has the effect of training the circuit on the inverse of the desired function, and then outputting that inverse whenever a future input propagates through the circuit. This bias signal can only change if all the inputs to the NLB are set to ‘0’ and $Y_{exp}$ is low.
Figure 4.8: Block diagram for the SHA-3 analog Dual-MTNLB implementation. One box labeled ‘NLB’ is equivalent to the circuit described in Fig. 4.6. One MTNLB pair is shown while the full design of $\theta$ will have 320 MTNLB pairs. Red wires represent active signals while black represent non-active signals. The MTNLB pair will always have five of its 10-bit lines active for any given input in this configuration.

The final block is the global trainer shown in Fig. 4.7d. The global trainer generates training signals when the MTNLB output does not match the expected output. A special XOR is used for the $Y_{exp}$ and $Y$ signals that outputs the XOR of the two signals ($SEL$) and the complement of $Y$, ($NEG$). Two latches are used with a second clock to ensure the sampled training signals stay valid for one training cycle. If $NEG$ is high, $Y$ is less than $Y_{exp}$ which sets $\overline{m^+}$ and $\overline{m^-}$ to cause a negative drop across the memristor (increased resistance). If $NEG$ is low, $Y$ is greater than $Y_{exp}$ which sets $\overline{m^+}$ and $\overline{m^-}$ to cause a positive drop across the memristor (decreased resistance). When $SEL$ and the $R/W$ signal are high, training is desired and $ClkSel$ goes high. If either $SEL$ is low, correct output, or $\overline{R/W}$ is low, currently reading a value, the value of $ClkSel$ is set low to prevent modification of memristor states.
4.2.2 Mitigation Five - NLB $\theta$ Plane Masking - Dual Implementation

Mitigation five follows the same concept as the digital dual-core SHA-3 implementation. For every MTNLB trained to compute an XOR function, there will be a complementary MTNLB that will learn the XOR function, but with inverted inputs. There will be a total of ten memristor weights trained to the XOR function with five (as ten total inputs with complements) inputs always active for any given input combination. This can be considered the dual-rail logic version of the MTNLB and can be seen in Fig. 4.8 showing how five, and only five, inputs will always be active.

4.3 Summary

To compare the analog results for effectiveness to the digital mitigations, the digital mitigations were re-simulated in circuits that used a one round SHA-3 permutation with only $\theta$ instantiated. As will be seen in the results, this is valid for comparisons as the addition of $\pi$, $\rho$, $\chi$ and $\iota$ portions does not significantly affect the success rates of finding a key.

---

$^1$The amount of time the $\bar{R}/W$ signal is high affects how much the memristor weights change. This affects the training rule used. To learn more about this parameter in the training rule, a stochastic gradient descent, refer to [13].
Chapter 5

Design Flow

5.1 Digital Simulations

The design flow used to test, validate, and attack the SHA-3 circuit implementations is a modification of the design flow presented in [22]. Fig. 5.1 shows the design flow used for power analysis in Synopsys. Python generates input data to the SHA-3 hash function that is defined as a 320-bit key pre-pended to a 704-bit message.

The generated input is sent to Matlab to calculate a hamming weight matrix\(^1\). The hamming weight matrix has a hamming weight calculated for every input vector and key guess pair. The hamming weight for any input vector can be found by first finding the \(\theta\) plane as seen below:

\(^1\)The hamming weight and hamming distance mean the same thing for \(\theta_{plane}\) creation. The initial state is all 0, and the attack is looking at the change from this 0 state to the \(\theta_{plane}\) calculation.

Figure 5.1: Block diagram of the digital design flow from RTL to synthesized power data using Synopsys, Matlab and Python. Red blocks use Python, blue blocks use Matlab and green blocks use Synopsys.
\[
\theta_{plane}(x, z) = \bigoplus_{y=0}^{4} S(x, y, z)
\]

\[x \in \{0, 1, \ldots, 4\}, z \in \{0, 1, \ldots, 63\}\] (5.1)

Where all coordinates \(y = 0\) are the 320 key bit guesses of the SHA-3 MAC algorithm. From the \(\theta\) plane, all hamming weights can be calculated as follow:

\[
HW_{p,i} = \left( \sum_{z=0}^{7} \theta_{plane,p} \left( \left\lfloor \frac{i}{8} \right\rfloor, (8 \ast i) \mod 64 + z \right) \right)
\]

\[p \in \{1, 2, \ldots, V\}, i \in \{1, 2, \ldots, 40\}\] (5.2)

\(HW_{p,i}\) is the hamming weight for the input vector \(p\), of the key byte \(i\), and \(V\) is the number of input vectors. For these experiments, the key guess size is one byte (256 values over 40 bytes) and the number of power traces used is 500,000 for a 500,000 X 256 X 40 matrix\(^2\). This is only calculated one time and can be reused on any number of circuits as long as the input data remains the same.

For the power collection phase, an RTL diagram along with a technology node file, TSMC 130 nm and 45 nm in this case, is read into the Synopsys Design Compiler. The compiler maps the HDL model into ASIC components and routes the design according to provided constraints. The synthesized HDL file is fed into the VCS MX simulator with a testbench to generate a switching file that notes events for power analysis. This file is given to the Synopsys Prime Time power analyzer along with a file that will give any top level port constraints. The Synopsys Prime Time tool saves power consumed for each event in a power_out file. The power_out file contains an entry for every component in the design allowing the ability to gather power consumption from any sub-entity. Also, Prime Time will estimate the average power consumption of the circuit divided between the clocking network, combinational logic, circuit leakage, and dynamic power.

For these experiments, only the total power of the circuit is required, so a Python file

\(^2\)This formula is only valid for this one byte case, changing the number of bits under attack would affect the size of the array and how many bits are summed together
is used to strip all of the lower level component entries to obtain a file that is easier to read in Matlab\(^3\). The final step is to perform correlation analysis on the power data and the hamming weight matrix to determine the correct key guesses. Fig. 5.2 shows an example of the attack flow. Matlab’s Corr function is used to compute the correlation of each column in the power matrix to every column in the current byte of the hamming weight matrix. Each column pair will produce one value in the correlation matrix that represents the correlation of key guess \(x\), at power collection time \(y\), for the key byte \(z\).

In [22], the number of power traces used for a design was under 10,000 traces. For SHA-3, since a linear function was the attack point, 500,000 traces were needed. The hard drive size required to collect 500,000 traces would result in over 10 TB of data for the initial SHA-3 implementation, with no mitigations, running the same scripts. To reduce this, a Python script was created that runs all the Synopsys sections of Fig. 5.1 as well as ‘Parse Output File’ and ‘Verify Result’. This script can be given an iteration number to collect that number of power traces from Synopsys, parse them, and then append them to

\(^3\)The component listed with ID-2 is the power consumption for the entire design.
the previously computed traces allowing the ability to run a given simulation on multiple cores (10 in this case, of 50,000 traces each) while only generating 10 GB of data total.

5.2 Analog Simulations

The MTNLB components were developed in Verilog-AMS and simulated in Cadence using NCSIM. Fig. 5.3 shows the block level diagram for the Cadence design flow. Since analog simulation take a long time, a single MTNLB was simulated 320 times to generate power profiles for 320 unique MTNLB instantiations. The power profile was created by allowing the MTNLB to train for 40 training cycles and then collecting the power consumed for one clock cycle of each possible input. A training cycle consists of seeing every possible input and the expected output to train on. Every MTNLB was verified to output the correct values, the XOR function, for all possible input combination before power results for that MTNLB were used. Using Matlab, the same algorithm that generated the hamming weight information was modified to index into one of the 320 MTNLBs and get the power that would be expected to be consumed for that SHA-3 state’s set of inputs. This was repeated for the remaining 319 MTNLBs in the design to generate the total power for one power trace. The process was repeated for the remaining 499,999 traces to obtain 500,000 traces that represented the $\theta$ function. Once the power traces were created, the attack used the same code to generate the correlation coefficient matrix and output plots. The mtnlbControl.py file also randomizes the initial states of the memristor before every simulation.

The digital components were simulated in Synopsys with a 130 nm and 45 nm node
size, while the analog simulation used a 45 and 16 nm node size. This was done to show a wide range of device technologies for SCA. Power consumption comparisons will only be taken at the 45 nm node size to provide fair comparisons. This work will also show the 130 nm simulation closely matches the results another group obtained with a physical evaluation board that contained both 45 nm and 22 nm FPGA fabrics [12]. This provides evidence in the practical application of the results from the different circuit simulations.

5.3 Script API

This section will break down each script with an explanation on how to use it and its intention.

5.3.1 generate.py

generate.py calls KeccakInputGen(...), to generate random input vectors based on a given key. The header for this function is:

```python
def KeccakInputGen(self, iterations, numFiles, k, r=1024, c=576, n=1024, verbose=False):
    """Compute the Keccak[r,c,d] sponge function on message M
    iterations: the number of input vectors inside a file
    numFiles: the number of files of size 'iterations' to make
    r: bitrate in bits (default: 1024)
    c: capacity in bits (default: 576)
    n: length of output in bits (default: 1024)
    key: key to use in this (must be 320 bits)
    verbose: print the details of computations (default: False)
    """
```

This script formats data as expected by the runSimulation.py script and was used in this work to generate 10 files containing 50,000 test vectors each.

5.3.2 runSimulation.py

runSimulation.py controls running the Synopsys tools in a way that is resource management friendly. The header for this script is:

```python
def runSimulation(circuitName, synopsysIterations, totalIterations, inputDataName):
    """Run the Synopsys power design flow""
```
This script will create an input file the correct size as denoted by `synopsysIterations` for as many runs until `totalIterations` is reached. If simulation fails for any reason, the script can be rerun and progress will continue from the most recent failure point. If reused for another algorithm, the only change that would need to be made is altering how the algorithm generates the input file to the testbench to match how the target algorithm’s test bench expects input. `runSimulation.py` also uses helper functions to format the Synopsys power file to something that is easier to read in Matlab.

### 5.3.3 `mtnlbControl.py`

`mtnlbControl.py` handles the analog power profiling step in Cadence by using the run-
`MTNLBSimulation(...)` function:

```python
def runMTNLBSimulation(inputSelectFile, MTNLBoutput, finalOut, numberNLBs):
    """simulates NLB component
    inputSelectFile : file containing initial memristor states
    MTNLBoutput : file for Cadence to dump power output
    finalOut : file to stop all power outputs
    numberNLBs : # NLB’s to simulate
    ""
```

This function will profile as many NLBs as set by `numberNLBs` while verifying that the output matches the expected output. The results are 30 time samples long, with a 5 ns step size, per line. Each line is the output of the MTNLB for one input combination (0-31). All the results are appended together in an output file that is used by Matlab to create the analog power traces.

### 5.3.4 `initialize.m, createPowerTrace.m, attack_cpa.m, thesisPlots.m`

These Matlab scripts all facilitate the CPA attack. `initialize.m` will create a hamming weight matrix, if one does not exist, for the current generated input data. If a new cryptographic algorithm was attacked, this file would need to be altered to match the hamming weight
model of that algorithm. *attack_cpa.m* performs a CPA attack using Matlab’s built in correlation function, Corr(…). This script is generic to any algorithm as long as the user implements a way to create the hamming weight matrix for their algorithm. The *thesisPlots.m* script generates all the figures seen in the results chapter. It works on the result of the correlation matrix; thus, is generic to any algorithm.

The *createPowerTrace.m* script is specific to the MTNLB power profiling results generated from the *mtnlbControl.py* script. It takes the input to the XORs in the $\theta$ plane and identifies the power consumption for the corresponding MTNLB. These results are summed for all 320 MTNLBs in the $\theta$ plane giving the MTNLB power trace for $\theta$. 

Figure 5.4: Folder structure used in this work for both simulation and attack.
5.3.5 Folder Structure

The folder structure used can be seen in Fig. 5.4. Inside the root folder is the folder containing all the inputs for the designed circuits, a folder for each circuit implementation, and all the helper scripts that are common to all implementations. Inside of a circuit folder is a Matlab and Synopsys folder. The `matlab` folder holds the formatted output from Synopsys before it is read in by Matlab using the `attack_cpa.m` script. Inside the Synopsys folder is the folder structure used for Synopsys power simulations. `compile.log`, `simulation.log` and `command.log` store the output for circuit synthesis, circuit simulation, and entered commands respectively. `compile.log` will contain the routing information along with the final size of the design layout. `simulation.log` will record any output from the testbench during circuit simulation as well as the final average power data for the circuit. The `rtl` folder holds all of the HDL files for the current implementation. The `lib` folder contains technology node files that indicate to Synopsys the technology node to use for synthesis. Also in this folder is a .sdc file that can be used to give signal constraints to any signal that is part of the top level hierarchy, i.e. inputs and outputs. The `tb` folder contains the current circuit’s testbenches. The `scr` folder contains the scripts used by the Makefile to automate the full Synopsys suite of tools. There is one script for compiling, `dc_commands_circuit.tcl`, one for simulation, `simv_commands_circuit.tcl`, and one for power collection, `pt_commands_circuit.tcl`. Finally, the `test_vector` folder contains the current set of inputs to give to the design for power collection. The files in this folder are created by the `runSimulation.py` script in the top level folder.
Chapter 6

Results

The first section in this chapter will show analysis of power traces and the attack methodology as it applies to the initial SHA-3 circuit with no modifications. Metrics to rate performance and benchmarks to evaluate the success of an attack will also be discussed as they apply to the initial SHA-3 circuit. The following sections will show results for each hardware mitigation following the metrics and benchmarks introduced in the first section.

6.1 Initial SHA-3 Attack

Before an attack can be carried out, and mitigations applied, there needs to be an understanding on the power profile of the circuit under investigation. Fig. 6.1 shows a single SHA-3 power trace for both 130 nm and 45 nm technology nodes. From the single trace, all rounds can be identified by noting the spikes of the waveform. This is useful when many power traces are needed for an attack, as a small window can be used to perform SPA/DPA/CPA. The first round happens at the largest spike of power for the 130 nm case, while the first round for the 45 nm case happens at the first spike higher than the power consumption of the falling edge of the clock, both 320 ns. This knowledge can reduce an attack to a few target cycles; however, this work will perform CPA on all clock cycles to analyze behavior over the entire SHA-3 algorithm. Also of note, $\theta$, $\rho$, $\pi$, $\chi$ and $\iota$ can not be deciphered from a single SHA-3 power trace. The 45 nm implementation consumes less power (9 mW compared to 48 mW spikes), but has more prominent static power leakages. This can be seen during the data loading phase of Fig. 6.1b where the dynamic consumption is not above the static leakage.

Fig. 6.2 shows 50 power traces for both the 130 nm and 45 nm technology nodes of
Figure 6.1: A single SHA-3 power trace with the 130 nm implementation on the left and the 45 nm implementation on the right. The red section is the data loading section that moves 64-bits of input into the state per clock cycle. The green section is the 24 cycle SHA-3 algorithm. The purple section is when the finish flag is set to denote a SHA-3 computation has completed. The orange section is the data output generation, 64-bits per cycle. The last cycle is the reset cycle bringing the state back to initial conditions.

Figure 6.2: 50 traces for the SHA-3 initial circuit with the 130 nm implementation on the left and the 45 nm implementation on the right. The lighter blue sections show both the data loading and data receiving sections. The yellow section of the left plot shows the initial round while all rounds are yellow on the right plot. In both cases, the variance in peak power differs by approximately 1 mW.
the SHA-3 initial implementation. The power traces can be seen to be consistent between each other with less than 1 mW of variance. This variance is the attack point for CPA, correlating to the input data.

Fig. 6.3 shows the correlation of 6 different key guesses. The top plot of each subplot is the correlations with an attack of 5,000 power traces. The bottom plot of each subplot is the correlations with an attack of 500,000 power traces. The plots show the importance of using enough power traces for an attack. With few traces, correct correlations are close to the same level as the incorrect correlations. The key guesses are specifically picked to show characteristics in linear function CPA. The first key guess is 0, the correct guess. The next key guess is 255, the 8-bit inverse of 0. The other key guesses are in pairs with their inverse, the values are: (8, 247) (1 and 7 hamming distance away) and (33, 222) (2 and 6 hamming distance away). It can be seen, in Fig. 6.3a, that the key guess of 0 is shown with the highest correlation. The correct key guess of round 1 is the lowest correlated value in round 2. The inverted guess of 255 follows the opposite behavior of the non-inverted correct guess; this makes sense when referring to the previous discussion in section 3.1.1. The key guess of 0 has no active bits, or a hamming weight of 0. The key guess of 255 has every bit active, or a hamming weight of 8. Since the hamming distance of these two guesses are the farthest apart, their power consumption should be the farthest apart if the hamming weight power model is valid. For the key guess that is 1 hamming distance away, 8, from the correct guess, 0, the correlation value is slightly lower than the correct correlation value. This follows for guesses that are 2 and 3 hamming distance away from the correct guess with decreasing correlation peaks. A hamming distance of 4 will result in the lowest peak while 5, 6 and 7 all follow the inverse behavior of 1, 2 and 3.

The mentioned behavior follows for both 6.3c and 6.3d; however, Fig. 6.3b does not follow this trend. No clear peaks are distinguishable even after 500,000 traces and correlation values are a magnitude lower than the others. The argument for CPA was every active bit consumes some characteristic power (hamming weights or hamming distance), either increasing or decreasing, depending on the number of active bits at a given point in time.
Figure 6.3: Correlation coefficients for 6 key guesses on key byte 1 of the SHA-3 initial implementation. The correct key is 0 with inverse 255. Key guess 8 is a hamming distance of 1 from the correct guess with 247 as its inverse. Key guess 33 is a hamming distance of 2 from the correct guess with 222 as its inverse.
Figure 6.4: The correlation matrix for the SHA-3 initial implementation over two technology nodes for full and θ configurations. θ configurations are similar to each other while the full implementations are not.

Since the guess pairs presented in the figure are the inverses of each other, their correlation coefficients are also the inverse of each other. This observation is meaningless when looking at non-linear functions such as the SHA-3 χ function or the AES S-box; yet, this property will be exploited in mitigation techniques applied to linear functions such as θ.

Fig. 6.4 shows the correlation matrix of key byte one over all 256 possible key guesses for both 130 nm and 45 nm technology nodes. The correlation matrix is a three-dimensional matrix that has time samples on the x-axis, key guesses on the y-axis and correlation coefficients on the z-axis. The probable correct key guess is the one that has the maximum peak of the correlation matrix (or 3D plot). The magnitude of peaks compared to others gives some information about how confident the max correlation value represents the correct key guess. In practice, this plot is not useful to perform an attack, but it can be useful
to see how different mitigations are effecting the power profile over the entire SHA-3 algorithm. Fig. 6.4b is the only plot that differs among the subplots with behavior that gives no clear peaks while the other subplots clearly show the current guess of ‘0’. Since there are many mitigations presented in this work, a set of metrics will be defined to be used in performance comparisons.

### 6.1.1 Performance Metrics

Comparing the highest correlation value to the next highest correlation value creates a performance metric called a confidence ratio defined as follows:

\[
C_k = \frac{\max(\Delta_k)}{\max(\{\Delta_0, \ldots, \Delta_N\} - \Delta_k)}
\]  

(6.1)

The smallest value for the confidence ratio is 1, which would only happen if the correlation coefficient of the top two key guesses were identical. The higher the confidence ratio, the higher the spread between the found key guess and the next highest guess; thus, a high confidence value would mean a high probability that the key guess chosen is the actual key. Masking this information would show that the mitigation technique gives less confidence that the attack worked, or give false confidence that it did work. Fig 6.5 shows the confidence ratios for all 40 bytes of the initial SHA-3 implementation for three different sets of power traces. Confidence ratios should increase with the addition of more power traces as seen from 5,000 to 250,000 traces. From 250,000 to 500,000 traces there is little change, indicating that the correlation coefficients are becoming stable. Fig. 6.5b breaks from this pattern with a mostly level confidence ratios showing no large changes with more power traces. This is expected given the trend from Fig. 6.4b with no clear correlation peaks.

Fig. 6.6 shows the confidence ratios in another way with indications if a key byte is being guessed correctly. Higher bars represent higher confidence values while green bars represent correct indicated guesses with red bars representing incorrect guesses. It is interesting to note, confidence ratios on linear functions are much lower than compared to non-linear functions like S-boxes, the main reason why attacks on linear functions require many more traces. The 45 nm full implementation has low confidence values with most
Figure 6.5: Confidence ratios for all key bytes at three different numbers of power traces used for the SHA-3 high-speed core implementation.
Table 6.1: Resource utilization for SHA-3 initial implementation. Numbers in parenthesis are the number of transistors for that gate while the total numbers are for that subsection of SHA-3.

<table>
<thead>
<tr>
<th>SHA-3 Section</th>
<th>Transistor Equation</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>State XOR</td>
<td>$1024 \times (8)$</td>
<td>8192</td>
</tr>
<tr>
<td>$\theta$</td>
<td>$320 \times (10 + 2 \times 8) + 1600 \times (10)$</td>
<td>24320</td>
</tr>
<tr>
<td>$\rho$</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$\pi$</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$\chi$</td>
<td>$1600 \times (6) + 1600 \times (2) + 1600 \times (8)$</td>
<td>25600</td>
</tr>
<tr>
<td>$\iota$</td>
<td>$64 \times (8)$</td>
<td>512</td>
</tr>
<tr>
<td>State Flip-Flops</td>
<td>$1600 \times (12)$</td>
<td>19200</td>
</tr>
<tr>
<td>Total</td>
<td>$8192 + 24320 + 25600 + 19200$</td>
<td>77312</td>
</tr>
</tbody>
</table>

key guesses staying incorrect while the other implementations all approach 100% correctly guessed key bytes, a trend that follows previous plots. The 130 nm $\theta$ only configuration provides the highest confidence ratios among the 4 subplots. Even through the 45 nm implementation had low confidence and low correct guesses, the 45 nm $\theta$ only implementation had high confidence and 100% success rate.

For circuit design, there are a few common metrics that apply to any implementation that can be used as comparison tools. Average power consumption and transistor count will be explored for each mitigation type. While security is the ultimate goal, a system that can provide good security while performing well in area and power consumption can be useful in embedded application domains.

Only transistors used in the actual SHA-3 circuit will be considered and not any that are used to interface data with the core. The Synopsys library contains cells for: XOR2, 8 transistors, XOR3, 10 transistors, AND2, 6 transistors, NOT, 2 transistors, and a 1-bit full-adder, 10 transistors. Table 6.1 shows the calculated transistor count for the initial SHA-3 circuit. Table 6.2 shows the approximate power consumption as measured from Synopsys for both 130 nm and 45nm technology nodes as well as $\theta$ only results. The 45 nm implementation consumed approximately 3 times less power for the full case and 2 times less power for the $\theta$ only case.
Figure 6.6: Confidence ratios of all 40 key bytes of the SHA-3 high-speed core. Key guesses that are correct are indicated in green while those that are not are red. Average confidence for all 40 bytes in each trial is included in the plot titles.
Table 6.2: Power consumption for high-speed SHA3 implementation (full and $\theta$ only) as reported by Synopsys Prime Time analyzer. $\theta$ only implementations did not use clocked components.

<table>
<thead>
<tr>
<th>Power Group</th>
<th>130nm (mW)</th>
<th>45nm (mW)</th>
<th>130nm $\theta$ (mW)</th>
<th>45nm $\theta$ (mW)</th>
</tr>
</thead>
<tbody>
<tr>
<td>clock_network</td>
<td>8.655 (48.42%)</td>
<td>3.094 (51.67%)</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>register</td>
<td>3.422 (19.14%)</td>
<td>1.085 (18.12%)</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>combinational</td>
<td>5.797 (32.43%)</td>
<td>1.809 (30.22%)</td>
<td>0.517 (100%)</td>
<td>0.237 (100%)</td>
</tr>
<tr>
<td>total</td>
<td>17.874</td>
<td>5.988</td>
<td>0.517</td>
<td>0.237</td>
</tr>
</tbody>
</table>

6.1.2 Benchmarks

The final comparison will be seeing how well each system performs while mitigating CPA. Fig. 6.7 gives an example of how benchmarks will be presented as it applies to the SHA-3 initial implementation. The success rate of guessing each key is plotted with 4 curves. Each curve represents what group the correct guess is in: top 1, top 2, top 4 and top 8. The top 1 group provides the most ideal attack result as there is no further computations needed to obtain the key. If there is 100% success for top 2, it means that there are a possible $2^{40}$ keys that need to be checked. For top 4 and top 8 the possible keys to check are $2^{80}$ and $2^{120}$ respectively.

Table 6.3: Time to brute force search a key space on the TIANHE-2 supercomputer if a key search takes 1000 FLOPS

<table>
<thead>
<tr>
<th>Key Size</th>
<th>Time to crack</th>
</tr>
</thead>
<tbody>
<tr>
<td>40 bits</td>
<td>32.47 mS</td>
</tr>
<tr>
<td>80 bits</td>
<td>1132 years</td>
</tr>
<tr>
<td>120 bits</td>
<td>$1.245 \times 10^{15}$ years</td>
</tr>
</tbody>
</table>

For perspective on what those numbers mean for a brute force attack, take an example using a super computer to brute force a key space. The TIANHE-2 is currently the fastest super computer and can do calculations at 33.86 petaflops a second. For a conservative estimate, assume it takes 1000 FLOPS to check one SHA-3 hash. This means the number of hashes that can be computed per second is: $\frac{33.86 \times 10^{15}}{1000} = 33.86 \times 10^{12}$. Table 6.3 lists how long each of the above key sizes would take to brute force search with the TIANHE-2. Top 1 and top 2 can be seen as easily broken, while top 4 and top 8 are not computationally feasible, but still would be considered successful for a theoretical attack.
Figure 6.7: Success rates for guessing the correct 40 key bytes versus the number of power traces used for the SHA-3 initial implementation. Data points are generated in increments of 5,000 from 5,000 to 500,000. Curves indicate the top group the correct key guess is in based on CPA results.
The $\theta$ implementations both have higher success rates than their matching full implementation for both technology nodes. The 45 nm full implementation has lower success with only 30% accuracy after 500,000 power traces. This seems to indicate that smaller circuits, i.e. $\theta$ only, are easier to attack than larger circuits where all computations happen in one clock cycle; intuitively, this makes sense.

### 6.2 Mitigation One — Dual-Core Design

Mitigation one attempts to mask data by taking inspiration from dual-rail logic. The desire is to have the output, and its inverse, generated for every gate in the design. This method instantiates two SHA-3 cores and uses the regular data for one, and the inverse data for the other. This is not true dual-rail logic as the $\chi$ function inverse can not be computed this way, but the desire is to mask just the $\theta$ step makes this implementation valid.

Fig. 6.8 shows the plot of 6 key guesses for the first byte of the dual-core mitigation. The behavior between the guesses is the same as what was seen in the initial implementation with no mitigations. Again, the full 45 nm implementation provides lower correlation values with incorrect results as the top correlation peaks. The correlation coefficient for the correct guess is 0.015 greater than that without any mitigations when comparing the dual core 130 nm to the no mitigation 130 nm circuit.

Fig. 6.9 shows the 3D correlation matrix plot of the dual-core mitigation circuit for key byte one. Fig. 6.9a suggests the mitigation does nothing as the correct byte being clearly identifiable. The $\theta$ only implementations both show better mitigation results with Fig. 6.9c getting the wrong guess as the highest correlation peak and 6.9d having multiple similar peaks. Fig. 6.9b looks like random noise, something seen in the initial implementation for the full 45 nm case.

Fig. 6.10 shows the confidence ratio for all 40 key bytes and how they change with more power traces used. Both 45 nm implementations show small increase in confidence ratios for any additional power traces above 5,000. The 130 nm $\theta$ implementation shows mainly high increases from 5,000 to 250,000 traces with not much change between 250,000 and 500,000. With 130 nm full, key bytes 1-9 are the only bytes in the implementation to show
Figure 6.8: Correlation coefficients for 6 key guesses on key byte 1 of the SHA-3 dual-core implementation. The correct key is 0 with inverse 255. Key guess 8 is a hamming distance of 1 from the correct guess with 247 as its inverse. Key guess 33 is a hamming distance of 2 from the correct guess with 222 as its inverse.
Figure 6.9: The correlation matrix for the SHA-3 dual-core implementation over two technology nodes for full and $\theta$ configurations.
Figure 6.10: Confidence ratios for all key bytes at three different numbers of power traces used for the SHA-3 dual-core implementation.
big confidence increases with increasing power traces. Analyzing the hamming weights in this section, the keys have values between 0 to 3, which is on the lower side of possible hamming weights (0-8); however, keys 0x09, 0x10, 0x11 all have a hamming weight of 2, and do not show signs of higher confidence.

Fig. 6.11 shows all 40 key bytes with their confidence ratios and if the current byte is indicated in the top 1 group for success. The key bytes with highest confidence shown in the previous plot are all guessed correctly and also have the highest confidence ratio. Having the correct guesses be those with the highest confidence ratio can be bad, as the attacker could use a simple threshold function to determine 8 key bytes correctly, reducing the brute force attack to $2^{8*(40-8)} = 2^{256}$. This can not be completed with a brute force attack, but it does constitute a theoretical break of the algorithm.

Some analysis can be done to understand why this portion of the key is weaker for the 130 nm full implementation. In the SHA-3 state, the bottom plane holds a key, the second and third planes hold message bits, the fourth plane holds one lane of message bits while the others are zero, and the fifth plane is all zeros. The increased confidence could come from the additional lane of message data that the attacker can use to figure out the key bytes. The first 8 key bytes all fall under this extra lane, making the attack target 3 lanes of message data and one of secret key information. Every other key byte has 2 lanes of message data, giving less known bits to unknown bits. The trend did not follow for the 130 nm θ only circuit which does bring into question if this truly is the case; however, the only correct guesses in both 45 nm implementations was in these key bytes, 1-9.

Fig. 6.12 shows the benchmark results for the dual-core design. Other than the 8 key bytes that were shown to be guessed correctly, no other key bytes are identified, and the addition of more power traces seems to add nothing more to the attack. This is a good result as it means that increasing the amount of power traces does not bring reduction in security through SCAs. This does not follow for the 45 nm full implementation that shows continual increasing success rates. At 500,000 traces, the mitigation still works, but it is unclear how high success rates could be with more traces.
Figure 6.11: Confidence ratios of all 40 key bytes of the SHA-3 dual-core implementation. Key guesses that are correct are indicated in green while those that are not are red. Average confidence for all 40 bytes in each trial is included in the plot titles.
Figure 6.12: Success rates for guessing the correct 40 key bytes versus the number of power traces used for the SHA-3 dual-core implementation. Data points are generated in increments of 5,000 from 5,000 to 500,000. Curves indicate the top group the correct key guess is in based on CPA results.
Table 6.4: Resource utilization for the SHA-3 dual-core implementation. Numbers in parenthesis are the number of transistors for that gate while the total numbers are for that subsection of SHA-3.

<table>
<thead>
<tr>
<th>SHA-3 Section</th>
<th>Transistor Equation</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>State XOR</td>
<td>$2 \times (1024 \times 8)$</td>
<td>16384</td>
</tr>
<tr>
<td>$\theta$</td>
<td>$2 \times (320 \times (10 + 2 \times 8) + 1600 \times (10))$</td>
<td>48640</td>
</tr>
<tr>
<td>$\rho$</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$\pi$</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$\chi$</td>
<td>$2 \times (1600 \times (6) + 1600 \times (2) + 1600 \times (8))$</td>
<td>51200</td>
</tr>
<tr>
<td>$\iota$</td>
<td>$2 \times (64 \times 8)$</td>
<td>1024</td>
</tr>
<tr>
<td>State Flip-Flops</td>
<td>$2 \times (1600 \times (12))$</td>
<td>38400</td>
</tr>
<tr>
<td>Total</td>
<td>$16384 + 48640 + 51200 + 38400$</td>
<td>154624</td>
</tr>
</tbody>
</table>

Table 6.5: Power consumption for the SHA-3 dual-core implementation (full and $\theta$ only) as reported by Synopsys Prime Time analyzer. $\theta$ only implementations did not use clocked components.

<table>
<thead>
<tr>
<th>Power Group</th>
<th>130 (mW)</th>
<th>45 (mW)</th>
<th>130 $\theta$ (mW)</th>
<th>45 $\theta$ (mW)</th>
</tr>
</thead>
<tbody>
<tr>
<td>clock_network</td>
<td>8.655 (37.35%)</td>
<td>3.094 (42.92%)</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>register</td>
<td>4.283 (18.48%)</td>
<td>1.085 (15.05%)</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>combinational</td>
<td>10.20 (44.17%)</td>
<td>3.030 (42.04%)</td>
<td>4.635 (100%)</td>
<td>1.989 (100%)</td>
</tr>
<tr>
<td>total</td>
<td>23.138</td>
<td>7.209</td>
<td>4.635</td>
<td>1.989</td>
</tr>
</tbody>
</table>

Table 6.4 shows the calculated transistor count for the dual-core SHA-3 implementation. Table 6.5 shows the approximate power consumption as measured from Synopsys. The dual-core implementation uses $\approx 30\%$ more power while increasing transistor counts by a factor of 2 for the 130 nm case compared to the initial implementation. The 45 nm full implementation consumes less than 3 times the power, a better result than the initial WDDL logic implementation introduced in this work that saw area increases on the order of 4 (due to transistor re-sizing) and area increases on the order of 2. No re-sizing was considered here as the data is clocked, and thus valid, at a pre-defined time in the circuit. This prevents the concern about early evaluation while still being able to use standard CMOS blocks. Power consumption was more than 3 times higher when comparing 130 nm and 45 nm full circuit configurations.
6.3 Mitigation Two — XNOR $\theta$ plane masking

The next mitigation strategy attempts to get the benefits of the dual-core mitigation design, but with less resources. Instead of using two streams of data as in the dual-core case, this method uses an array of XNOR gates to compute the $\theta$ inverse while the $\theta$ plane, XOR gates, is created.

Fig. 6.13 shows the plot of 6 key guesses for the first key byte of the XNOR mitigation. The behavior follows the previously mentioned behavior for key byte guesses for all subplots with one exception, the second round. The second round provides correlation that is equal to the first round for a key guess, with the sign being the same (previous results shown a change of sign in the second round). From the no mitigation implementation, the XNOR mitigation decreased the correlation coefficient by 0.025 for 130 nm. The top correlation coefficient for the 45 nm full case was double the 130 nm case. For the $\theta$ implementations, more power traces can be seen to increase correlation coefficient of incorrect guesses toward the correlation coefficient of the correct guess. The overall correlation coefficient of the $\theta$ implementations can be seen to be significantly higher than the full circuit implementations.

Fig. 6.14 shows the 3D correlation plot for the XNOR mitigation giving a better idea of what was seen from the previous plot. When compared to the other mitigations, the lower correlation coefficients bring the correct peaks closer to the noise threshold, for 130 nm; yet, peaks at guess 0, the correct value for this key byte, can still be identified. This behavior is less prevalent when looking at the 45 nm full implementation that still has peaks much higher than the surrounding noise. The 130 nm $\theta$ only configuration provided peaks that were an order of magnitude higher than the full 130 nm configuration while the 45 nm $\theta$ only configuration was twice that of the 45 nm full implementation. This plot shows a behavior that was not seen in the initial and dual-core mitigations with the lower technology node being easier to attack.

Fig. 6.15 shows the confidence ratios for all key bytes with increasing number of used power traces. For most key bytes, the increased number of power traces does lead to higher
Figure 6.13: Correlation coefficients for 6 key guesses on key byte 1 of the SHA-3 XNOR implementation. The correct key is 0 with inverse 255. Key guess 8 is a hamming distance of 1 from the correct guess with 247 as its inverse. Key guess 33 is a hamming distance of 2 from the correct guess with 222 as its inverse.
Figure 6.14: The correlation matrix for the SHA-3 XNOR implementation over two technology nodes for full and $\theta$ configurations.
Figure 6.15: Confidence ratios for all key bytes at three different numbers of power traces used for the SHA-3 XNOR implementation.
confidence values. This is not the case for Fig. 6.15a which had relatively similar confidence ratios even with increasing power traces. Confidence ratios were higher in the 45 nm full implementation when compared to the 130 nm full implementation. The matching θ implementations shared this behavior except with higher total confidence ratios. After 250,000 traces, no configuration seems to benefit greatly from additional power traces.

Fig. 6.16 shows the confidence ratios with the correctly guessed correctly key bytes indicated. The 130 nm implementation has the lowest confidence values with the least number of correctly guessed key bytes. The 130 nm θ only implementation also has the behavior of low number of correct guesses. The 45 nm full implementation has a wide range of correlation values, but is able to get every key correct except for four. The 45 nm θ only implementation performs the worst with very high confidence ratios and 100% success on guessing the correct key byte.

Fig. 6.17 shows the benchmark results for the SHA-3 XNOR mitigation. The strongest configuration is the 130 nm circuit that has the top 8 group achieving a max success of 80%. The 130 nm θ only implementation’s top 8 group achieves a max accuracy of 90%, compelling evidence as a bound for the full implementation. After 50,000 traces, the 130 nm θ implementation does not increase in accuracy with additional traces. This behavior is desirable in a SCA mitigation. The 45 nm full circuit performs worse than the 130 nm full implementation with the top 2 group getting to 100% after 400,000 traces. The 45 nm θ implementation is the worst performing in this set of configurations with the entire key identified in 100,000 traces. This seems to indicate that the smaller technology node can be easier to attack, or atleast, harder to mitigate attacks on.

Table 6.6 shows the calculated transistor count for the SHA-3 XNOR implementation. Table 6.7 shows the approximate power consumption as measured from Synopsys. Compared to the initial implementation, there was a 0.6% increase in power for the 130 nm full mitigation and a 10.8% increase in transistor count. The 45 nm implementation reduced power consumption by a factor of 2.8 for the full implementation and a factor of 2.46 for the θ only implementations.
Figure 6.16: Confidence ratios of all 40 key bytes of the SHA-3XNOR implementation. Key guesses that are correct are indicated in green while those that are not are red. Average confidence for all 40 bytes in each trial is included in the plot titles.
Figure 6.17: Success rates for guessing the correct 40 key bytes versus the number of power traces used for the SHA-3 XNOR implementation. Data points are generated in increments of 5,000 from 5,000 to 500,000. Curves indicate the top group the correct key guess is in based on CPA results.
Table 6.6: Resource utilization for the SHA-3 XNOR implementation. Numbers in parenthesis are the number of transistors for that gate while the total numbers are for that subsection of SHA-3.

<table>
<thead>
<tr>
<th>SHA-3 Section</th>
<th>Transistor Equation</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>State XOR</td>
<td>1024 * (8)</td>
<td>8192</td>
</tr>
<tr>
<td>$\theta$</td>
<td>$2 \times (320 \times (10 + 2 \times 8)) + 1600 \times (10)$</td>
<td>32640</td>
</tr>
<tr>
<td>$\rho$</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$\pi$</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$\chi$</td>
<td>$1600 \times (6) + 1600 \times (2) + 1600 \times (8)$</td>
<td>25600</td>
</tr>
<tr>
<td>$i$</td>
<td>$64 \times (8)$</td>
<td>512</td>
</tr>
<tr>
<td>State Flip-Flops</td>
<td>$1600 \times (12)$</td>
<td>19200</td>
</tr>
<tr>
<td>Total</td>
<td>$8192 + 32640 + 25600 + 19200$</td>
<td>85632</td>
</tr>
</tbody>
</table>

Table 6.7: Power consumption for the SHA-3 XNOR implementation (full and $\theta$ only) as reported by Synopsys Prime Time analyzer. $\theta$ only implementations did not use clocked components.

<table>
<thead>
<tr>
<th>Power Group</th>
<th>130 nm (mW)</th>
<th>45 nm (mW)</th>
<th>130 nm $\theta$ (mW)</th>
<th>45 nm $\theta$ (mW)</th>
</tr>
</thead>
<tbody>
<tr>
<td>clock_network</td>
<td>8.655 (48.09%)</td>
<td>3.094 (48.45%)</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>register</td>
<td>3.414 (18.97%)</td>
<td>1.085 (16.99%)</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>combinational</td>
<td>5.927 (32.94%)</td>
<td>2.207 (34.56%)</td>
<td>2.491 (100%)</td>
<td>1.011 (100.00%)</td>
</tr>
<tr>
<td>total</td>
<td>17.996</td>
<td>6.386</td>
<td>2.491</td>
<td>1.011</td>
</tr>
</tbody>
</table>

6.4 Mitigation Three — MTNLB $\theta$ plane masking - digital

This mitigation looks at the implementation of a digital MTNLB. The attempt is to see if neuromorphic computing principles can mask power consumption while staying in the digital design domain.

Fig. 6.18 shows the plot of 6 key guesses for the first byte of the digital MTNLB mitigation. None of the subplots show any indication of leaking key information. Fig. 6.18a shows very low correlation values, spikes of $4 \times 10^{-3}$, with no clear indications that any value is correlated to the input. This follows for the rest of the subplots with the 45 nm plots, both full and $\theta$ only, having higher correlation than the 130 nm plots. Fig. 6.18b looks like information is leaked at 320 ns, but this shows an incorrect key with the highest correlation coefficient.

Fig. 6.19 shows the 3-D correlation matrix for the first key byte of the MTNLB mitigation. The same behavior of lower correlation values for the 130 nm configurations can
Figure 6.18: Correlation coefficients for 6 key guesses on key byte 1 of the SHA-3 digital MTNLB implementation. The correct key is 0 with inverse 255. Key guess 8 is a hamming distance of 1 from the correct guess with 247 as its inverse. Key guess 33 is a hamming distance of 2 from the correct guess with 222 as its inverse.
be seen. Fig. 6.19c shows behavior that resembles the initial attack, with lower correlation peaks; however, the correct key guess is denoted as '08' instead of '00'. Fig. 6.19b also looks similar to the initial attack results but follows the same incorrect guess as the 130 nm case. Fig. 6.19d shows an interesting behavior not seen in any other mitigation; there seems to be only two values of correlation, high and low. This is excellent mitigation behavior as this should mean confidence values are near one, and there should be no confidence if a key is correct or not. The behavior is something to explore in the future to attempt to replicate in mitigation implementations.

Fig. 6.20 shows the confidence ratios over different amounts of power traces. For all subplots, the behavior is generally the same with increased power traces not changing confidence values. Only Fig. 6.20a goes against this trend with a few key bytes, but still not enough to confirm any type of trend. These results match well with the 3-D correlation
Figure 6.20: Confidence ratios for all key bytes at three different numbers of power traces used for the SHA-3 digital MTNLB implementation.

plots that had correlations that did not leak correct key guess information.

Fig. 6.21 shows the confidence values for the MTNLB mitigation and the key bytes guessed correctly. Confidence values stay low, with the highest coming from the 130 nm (only $\theta$) configuration. Both 130 nm implementations have a behavior of several correct guesses in the 125,000 and 375,000 range, but than drop in correctness at 500,000. A useful behavior as it puts an upper bound on the mitigation where additional traces do not improve results. Fig. 6.21b had the most correct key guesses at 500,000 traces. For all subplots though, average confidence values remain low, with Fig. 6.21d having the lowest confidence.

Fig. 6.22 shows the success rates for the MTNLB mitigation. All configurations have the same overall behavior of a constant success rate that is insensitive to the number of
Figure 6.21: Confidence ratios of all 40 key bytes of the SHA-3 digital MTNLB implementation. Key guesses that are correct are indicated in green while those that are not are red. Average confidence for all 40 bytes in each trial is included in the plot titles.
Table 6.8: Resource utilization for a single digital MTNLB.

<table>
<thead>
<tr>
<th>Component</th>
<th>Transistors</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>8-bit 2-1 Mux</td>
<td>48</td>
<td>5*48=240</td>
</tr>
<tr>
<td>8-bit weight</td>
<td>96</td>
<td>5*96=480</td>
</tr>
<tr>
<td>8-bit adder</td>
<td>80</td>
<td>4*80=320</td>
</tr>
<tr>
<td>8-bit comparitor</td>
<td>124</td>
<td>124*5=620</td>
</tr>
<tr>
<td>NAND gate</td>
<td>4</td>
<td>5*4=20</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td><strong>1680</strong></td>
</tr>
</tbody>
</table>

power traces used. Fig. 6.22a and Fig. 6.22b deviate most from this behavior with a high increase in success from 5,000 to 100,000 traces and a constant level until 450,000 traces. There is a strange drop from 460,000 to 500,000 in the 130 nm case that brings the success rates below the configuration in Fig. 6.22c. Why that happens is most likely because of the previously presented results showing confidence values; confidence values, and likewise correlation values, are so low, the attack can be thought of as being lucky. The right message bits seem to find some secret keys better, but those message bits are only identifiable after the correct key is known. The assumption is that this same message bits with a different key would likely result in different behavior with the commonality being around the 460,000 to 500,000 success rate results.

To get the data for resource size, the previous transistor count from Table 6.1 can be used with modifications to $\theta$ to be $320 \times (MTNLB\_count) + 1600 \times (10)$. Table 6.8 shows the break down of the number of transistors to implement different components in the MTNLB. Table 6.9 shows the calculated transistor count for the SHA-3 digital MTNLB implementation. Table 6.10 shows the approximate power consumption as measured from Synopsys. Fully digital implementations of neuromorphic computing components can be very inefficient even with optimized designs. The transistor count increased by a factor of 7.8 while power consumption increased by a factor of 2.06. While the digital MTNLB did not have a good result for size, this 2.06 more power consumption is inline with other currently proposed dual-rail logic mitigation solutions.
Figure 6.22: Success rates for guessing the correct 40 key bytes versus the number of power traces used for the SHA-3 digital MTNLB implementation. Data points are generated in increments of 5,000 from 5,000 to 500,000. Curves indicate the top group the correct key guess is in based on CPA results.

Table 6.9: Resource utilization for the SHA-3 digital MTNLB implementation. Numbers in parenthesis are the number of transistors for that gate while the total numbers are for that subsection of SHA-3.

<table>
<thead>
<tr>
<th>SHA-3 Section</th>
<th>Transistor Equation</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>State XOR</td>
<td>1024 * (8)</td>
<td>8192</td>
</tr>
<tr>
<td>θ</td>
<td>320 * (1680) + 1600 * (8)</td>
<td>550400</td>
</tr>
<tr>
<td>ρ</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>π</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>χ</td>
<td>1600 * (6) + 1600 * (2) + 1600 * (8)</td>
<td>25600</td>
</tr>
<tr>
<td>ι</td>
<td>64 * (8)</td>
<td>512</td>
</tr>
<tr>
<td>State Flip-Flops</td>
<td>1600 * (12)</td>
<td>19200</td>
</tr>
<tr>
<td>Total</td>
<td>8192 + 550400 + 25600 + 19200</td>
<td>603392</td>
</tr>
</tbody>
</table>
Table 6.10: Power consumption for the SHA-3 digital MTNLB implementation (full and \( \theta \) only) as reported by Synopsys Prime Time analyzer. \( \theta \) only implementations did not use clocked components.

<table>
<thead>
<tr>
<th>Power Group</th>
<th>130 nm (mW)</th>
<th>45 nm (mW)</th>
<th>130 nm ( \theta ) (mW)</th>
<th>45 nm ( \theta ) (mW)</th>
</tr>
</thead>
<tbody>
<tr>
<td>clock_network</td>
<td>8.431 (22.83%)</td>
<td>3.094</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>register</td>
<td>3.887 (10.52%)</td>
<td>1.085</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>combinational</td>
<td>24.6 (66.65%)</td>
<td>6.941</td>
<td>24.5 (100%)</td>
<td>6.977 (100%)</td>
</tr>
<tr>
<td>total</td>
<td>36.9</td>
<td>11.1</td>
<td>24.5</td>
<td>6.977</td>
</tr>
</tbody>
</table>

6.5 Mitigation Four — MTNLB \( \theta \) plane masking - analog

The analog simulations presented use a different simulator suite called Cadence. The difference between the Synopsys and Cadence simulators is event based versus time step based. This will make the analog plots look different as there is power information given every time step, 5 ns, for what power was consumed in the circuit; however, Synopsys would output nothing if an event was not scheduled. Also, the analog simulations of the NLB were done with a 150 ns clock period to see if any power leakage information might be given at slower clock speeds.

Fig. 6.23 shows 6 key guesses for key byte 1 of the analog MTNLB mitigation implementation. Fig. 6.23b follows what was seen in the initial implementation during the first \( \approx 10 \) ns (dynamic power consumption). After about \( \approx 20 \) ns, when the power consumed is mainly from leakage power, the correlation becomes stronger for a key guess two hamming weights away from the correct guess than a key guess that is one hamming weight away. For the attack, this has no effect since both correlation values are always below the correct guess, but it could be an indicator on properties specific to leakage power loss and CPA. For Fig. 6.23a, more power traces can be seen to drop the correct guess correlation coefficient under a wrong guess.

Fig. 6.24 shows the 3D correlation matrix for the analog NLB. Note the difference in shape that follows the behavior of a power trace in Fig. 6.23. The spiking behavior of Fig. 6.24a looks similar to the spiking behavior of the initial implementation, but with reverse behavior (i.e. correct guess of ‘00’ is the lowest). For Fig. 6.24b, the spiking behavior is more uniform, however, the correct guess of ‘0’ is clearly the highest value.
Figure 6.23: Correlation coefficients for 6 key guesses on key byte 1 of the SHA-3 analog MTNLB implementation. The correct key is 0 with inverse 255. Key guess 8 is a hamming distance of 1 from the correct guess with 247 as its inverse. Key guess 33 is a hamming distance of 2 from the correct guess with 222 as its inverse.

Figure 6.24: The correlation matrix for the SHA-3 analog MTNLB implementation over two technology nodes.
Fig. 6.25 shows the confidence ratios for all 40 key bytes with increasing number of power traces for the analog MTNLB. Fig. 6.25a shows almost no effect from the number of power traces used while Fig. 6.25b shows increasing confidence with more used power traces.

Fig. 6.26 shows the confidence ratios for all key bytes and the guesses that are currently correct for the analog NLB. Fig. 6.26a has increasing correctness with more power traces, but maintains an almost constant average confidence value. In contrast, Fig. 6.26b shows almost all key guesses correctly guessed from 5,000 traces and average confidence increasing all the way to 500,000.

Fig. 6.27 shows the benchmark results for the analog MTNLB implementation. For Fig. 6.27b, after about 25,000 traces all groups report 100% success rate except for the top 1 group with a 99.75% accuracy between traces 95,000 to 155,000. This means that no brute force work is needed and the key can be obtained very quickly with a minimal amount of traces. For Fig. 6.27a, the top 8 group has the highest accuracy with only 72.5% and what looks like converging behavior. A dual-core MTNLB solution may be one that provides as good of mitigation as the digital dual-core with lower power and area.

The size of one MTNLB is given in [24] as \(2n m + 20n + 12m + 30 + 2\text{ceil}(m/2)\) where \(n\) is the number of inputs and \(m\) is the number of decision boundaries needed, \(n + 1\) in this case. The size of one 5-input MTNLB is then 268 transistors. Table 6.11 shows
Figure 6.26: Confidence ratios of all 40 key bytes of the SHA-3 analog MTNLB implementation. Key guesses that are correct are indicated in green while those that are not are red. Average confidence for all 40 bytes in each trial is included in the plot titles.

Figure 6.27: Success rates for guessing the correct 40 key bytes versus the number of power traces used for the SHA-3 analog MTNLB implementation. Data points are generated in increments of 5,000 from 5,000 to 500,000. Curves indicate the top group the correct key guess is in based on CPA results.
Table 6.11: Resource utilization for the SHA-3 analog MTNLB implementation. Numbers in parenthesis are the number of transistors for that gate while the multiplied number is that many of the gate.

<table>
<thead>
<tr>
<th>SHA-3 Section</th>
<th>Transistor Equation</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>State XOR</td>
<td>1024 * (8)</td>
<td>8192</td>
</tr>
<tr>
<td>$\theta$</td>
<td>320 * (268) + 1600 * (10)</td>
<td>101760</td>
</tr>
<tr>
<td>$\rho$</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$\pi$</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$\chi$</td>
<td>1600 * (6) + 1600 * (2) + 1600 * (8)</td>
<td>25600</td>
</tr>
<tr>
<td>$\iota$</td>
<td>64 * (8)</td>
<td>512</td>
</tr>
<tr>
<td>State Flip-Flops</td>
<td>1600 * (12)</td>
<td>19200</td>
</tr>
<tr>
<td>Total</td>
<td>8192 + 101760 + 25600 + 19200</td>
<td>154752</td>
</tr>
</tbody>
</table>

Table 6.12: Power consumption for the SHA-3 analog MTNLB implementation as reported by Cadence NCSIM.

<table>
<thead>
<tr>
<th>Power Group</th>
<th>45 nm $\theta$ (mW)</th>
<th>16 nm (mW) $\theta$</th>
</tr>
</thead>
<tbody>
<tr>
<td>total</td>
<td>2.36e-05</td>
<td>1.35e-05</td>
</tr>
</tbody>
</table>

the calculated transistor count for the SHA-3 analog MTNLB implementation. The total count uses the number of transistors that was used in the digital cases, but these mixed signal circuits were not simulated; the values are indicated for comparisons on total possible circuit sizes. Table 6.12 shows the approximate power consumption as measured from 500,000 power traces with 30 samples each. The analog MTNLB transistor count increased by a factor of 2 while power consumption decreased by a factor of 10,000 when compared to the initial 45 nm $\theta$ implementation.

### 6.6 Mitigation Five — MTNLB $\theta$ plane masking - analog dual-core

Mitigation strategy five attempts to implement both an MTNLB and use concepts of dual-rail logic with a complementary MTNLB.

Fig. 6.28 shows the results for 6 key guesses for key byte one of the analog dual-MTNLB implementation. The behavior of Fig. 6.28b is very similar to the single MTNLB except with the correlation magnitudes being less than half of the single MTNLB implementation. Having the same behavior as a mitigation that did not do well can be seen initially as a negative, but correlation peaks closer to 0 can mean correlations are approaching
Figure 6.28: Correlation coefficients for 6 key guesses on key byte 1 of the SHA-3 analog dual-MTNLB implementation. The correct key is 0 with inverse 255. Key guess 8 is a hamming distance of 1 from the correct guess with 247 as its inverse. Key guess 33 is a hamming distance of 2 from the correct guess with 222 as its inverse.
Figure 6.29: The correlation matrix for the SHA-3 analog dual-MTNLB implementation over two technology nodes for full and \( \theta \) configurations.

what will look like noise. Fig. 6.28a performs similarly, but with a large initial correlation peak for the inverse of the correct key guess.

Fig. 6.29 shows the 3D plot of the correlation matrix. Both plots show good behavior in that the correct key guess does not have the highest peaks. Fig. 6.29a shows an interesting spiking behavior that has similar high spikes and low spikes. In Fig. 6.29b The largest peak happens around key guess 15 and has a correlation coefficient 2 times greater than the correct guess. Also, the incorrect correlation magnitude is the same magnitude of the correct guess in the no mitigation circuit. This shows the analog dual-MTNLB circuit giving results with high correlation, but that are incorrect; in an attack mitigation context, this is optimal to mask what is happening in the circuit.

Fig. 6.30 shows the confidence ratios of all key bytes with increasing number of used power traces. There is no clear trend from the number of power traces used and the confidence ratio. For some cases, more traces increase the ratio, for others, it decreases the ratio. For a large portion, there seems to be no effect on the number of power traces and the confidence ratio values. The not clear behavior could be good in a mitigation stand point if the behavior of the confidence ratio does not indicate correct or incorrect guesses.

Fig. 6.31 shows the confidence ratio of all the key bytes and if the current guess is correct for the analog dual-core analog MTNLB implementation. The average confidence ratio is the lowest of all mitigations applied at 1.062 for 16 nm and 1.046 for 45nm after
Figure 6.30: Confidence ratios for all key bytes at three different numbers of power traces used for the SHA-3 analog dual-MTNLB implementation.

Figure 6.31: Confidence ratios of all 40 key bytes of the SHA-3 analog dual-MTNLB implementation. Key guesses that are correct are indicated in green while those that are not are red. Average confidence for all 40 bytes in each trial is included in the plot titles.
Figure 6.32: Success rates for guessing the correct 40 key bytes versus the number of power traces used for the SHA-3 analog dual-MTNLB implementation. Data points are generated in increments of 5,000 from 5,000 to 500,000. Curves indicate the top group the correct key guess is in based on CPA results.

500,000 traces. Note, the confidence ratio does not always improve with more guesses, but can get worse. An automated attack would be hard to implement if the confidence ratio was used as a threshold. There are cases where a high confidence ratio indicates both correct and wrong guesses, and the same is true for low confidence ratios. The case with the lowest average confidence, 45 nm, also gives the most correct key guesses.

Fig. 6.32 shows the benchmark results for the SHA-3 analog dual-MTNLB implementation. Fig. 6.32a shows accuracies that are very high after only 5,000 traces, however, they do not seem to be effected by adding more power traces. This is similar to Fig. 6.32b.

Table 6.13 shows the calculated transistor count for the SHA-3 analog dual-MTNLB implementation. Table 6.14 shows the approximate power consumption as measured from Synopsys. The power data was obtained from averaging power consumption data reported by Cadence over 500,000 traces. The transistor count for this implementation was higher than the initial implementation by a factor of 3.11 while the power dissipation was less than the initial 45 nm θ implementation by a factor of 4300. Even with the analog circuit being twice as large as its single MTNLB implementation, the large decrease in power consumption could offset the 3.11 times area increase.
Table 6.13: Resource utilization for the SHA-3 analog dual-MTNLB implementation. Numbers in parenthesis are the number of transistors for that gate while the total numbers are for that subsection of SHA-3.

<table>
<thead>
<tr>
<th>SHA-3 Section</th>
<th>Transistor Equation</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>State XOR</td>
<td>1024 * (8)</td>
<td>8192</td>
</tr>
<tr>
<td>( \theta )</td>
<td>( 2 * (320 * (268)) + 1600 * (10) )</td>
<td>187520</td>
</tr>
<tr>
<td>( \rho )</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( \pi )</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( \chi )</td>
<td>( 1600 * (6) + 1600 * (2) + 1600 * (8) )</td>
<td>25600</td>
</tr>
<tr>
<td>( \iota )</td>
<td>( 64 * (8) )</td>
<td>512</td>
</tr>
<tr>
<td>State Flip-Flops</td>
<td>( 1600 * (12) )</td>
<td>19200</td>
</tr>
<tr>
<td>Total</td>
<td>8192 + 187520 + 25600 + 19200</td>
<td>240512</td>
</tr>
</tbody>
</table>

Table 6.14: Power consumption for the SHA-3 analog dual-MTNLB implementation as reported by Cadense NCSIM.

<table>
<thead>
<tr>
<th>Power Group</th>
<th>45 ( \theta ) (mW)</th>
<th>16 ( \theta ) (mW)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total</td>
<td>5.42e-05</td>
<td>3.10e-05</td>
</tr>
</tbody>
</table>

### 6.7 Reconfigurable Circuits

The MTNLB is a reconfigurable computing element that has the ability to learn any \( n \) input function. Using an NLB design and comparing it to the area of an ASIC design can be an unfair comparison. The ASIC design is not reconfigurable in nature, but the MTNLB design could be used to implement many other functions if needed. As such, the MTNLB is comparable to LUTs that are found in FPGAs.

For size comparisons, a minimum sized LUT is defined in [21] as:

\[
2^n * T_m + 4 * (2^n - 1)
\]  

\( n \), as in the MTNLB, is defined as the number of inputs for the function. \( T_m \) is the number of transistors to make a memory element. A 5-input LUT should have approximately 316 transistors if the number a transistors for a memory element is 6. It was shown earlier that a 5-input MTNLB, including training circuitry, contains 268 transistors. This makes MTNLBs 18% smaller than traditional LUTS, making the dual-MTNLB implementation shown in this work to be much less of an overhead (about 60% more area for analog
dual-MTNLB than the initial SHA-3 circuit on FPGA).

The results will still show comparison to ASIC design as that is what the simulations and power profiling was done for; however, it is important to note where MTNLBs become more interesting as replacements to LUTs. The reconfigurable nature can give SCA resistant properties to a wide range of algorithms.

6.8 Summary

Table 6.15 gives the summary of results from all circuits in the results section. For consideration of ASIC design, the dual-core and XNOR circuits had the best performance. The dual-core implementation doubled the area, but had correlations that were half as strong in magnitude and was able to keep all guesses around 20% accuracy. The XNOR mitigation provided a very small form factor while still giving sufficient security to prevent SCAs to determine a secret key. 80% for the top 8 group is a high success rate, but without 100% confidence in one of the groups, it is not possible to reduce the key search space. From the behavior of the success plots, it seems that the addition of any more traces would not dramatically increase the given percentages of the XNOR mitigation. The digital MTNLB performed well, but the area overhead makes it something that would not be usable in every application. The top performing circuit was the dual-MTNLB, with the lowest combined power and success rate, even though average correlation was the highest. There is a significant area overhead close to a factor of 3, however, this is lower than some area overheads that are seen with WDDL mitigation techniques with factors of 4. The dual-MTNLB would shine in reconfigurable computing, where the MTNLB is a fraction of the size of a current LUT. This would give a mitigation technique that would require 60% area overhead to get mitigation results that make SCAs near impossible (at least in the current forms of SPA/DPA/CPA for power analysis).

In comparison to a physical implementation, [12] had results that were similar to the initial SHA-3 circuit simulations. Both initial circuits used the same HDL model provided by the SHA-3 authors and both circuits saw accuracies approaching near 100%, but not quite reaching 100% for the top 1 guess group. The similarities in physical implementation
to the simulation gives some evidence that the mitigation simulations are realistic, and would have similar behavior if implemented in physical hardware. Their work used 4 potential attack models for results while this work only used one, the model that provided the highest correlation values.

Table 6.15: Collected results for all circuits discussed in this work. Average confidence and average max correlation is added. Both metrics are taken from the results of 500,000 used power traces

<table>
<thead>
<tr>
<th>Circuit</th>
<th>#Trans</th>
<th>Pow. (mW)</th>
<th>Avg. Conf</th>
<th>Avg. Corr</th>
<th>Top1</th>
<th>Top2</th>
<th>Top4</th>
<th>Top8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial 130 nm</td>
<td>77312</td>
<td>17.874</td>
<td>1.202</td>
<td>0.0317</td>
<td>97.5%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>Initial 45 nm</td>
<td>77312</td>
<td>5.988</td>
<td>1.042</td>
<td>0.0542</td>
<td>7.5%</td>
<td>12.5%</td>
<td>27.5%</td>
<td>27.5%</td>
</tr>
<tr>
<td>Initial (\theta) 130 nm</td>
<td>24320</td>
<td>0.517</td>
<td>1.299</td>
<td>0.0720</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>Initial (\theta) 45 nm</td>
<td>24320</td>
<td>0.237</td>
<td>1.258</td>
<td>0.0314</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>Dual-Core 130 nm</td>
<td>154624</td>
<td>23.138</td>
<td>1.093</td>
<td>0.0144</td>
<td>20%</td>
<td>20%</td>
<td>22.5%</td>
<td>22.5%</td>
</tr>
<tr>
<td>Dual-Core 45 nm</td>
<td>154624</td>
<td>7.209</td>
<td>1.045</td>
<td>0.0058</td>
<td>15%</td>
<td>17.5%</td>
<td>25%</td>
<td>30%</td>
</tr>
<tr>
<td>Dual-Core (\theta) 130 nm</td>
<td>48640</td>
<td>4.635</td>
<td>1.208</td>
<td>0.1084</td>
<td>17.5%</td>
<td>20%</td>
<td>20%</td>
<td>20%</td>
</tr>
<tr>
<td>Dual-Core (\theta) 45 nm</td>
<td>48640</td>
<td>1.989</td>
<td>1.038</td>
<td>0.0106</td>
<td>5%</td>
<td>7.5%</td>
<td>20%</td>
<td>22.5%</td>
</tr>
<tr>
<td>XNOR 130 nm</td>
<td>85632</td>
<td>17.996</td>
<td>1.051</td>
<td>0.0138</td>
<td>35%</td>
<td>52.5%</td>
<td>67.5%</td>
<td>80%</td>
</tr>
<tr>
<td>XNOR 45 nm</td>
<td>85632</td>
<td>6.386</td>
<td>1.134</td>
<td>0.0174</td>
<td>90%</td>
<td>97.5%</td>
<td>97.5%</td>
<td>97.5%</td>
</tr>
<tr>
<td>XNOR (\theta) 130 nm</td>
<td>32640</td>
<td>2.491</td>
<td>1.190</td>
<td>0.1096</td>
<td>70%</td>
<td>75%</td>
<td>77.5%</td>
<td>87.5%</td>
</tr>
<tr>
<td>XNOR (\theta) 45 nm</td>
<td>32640</td>
<td>1.011</td>
<td>1.265</td>
<td>0.0404</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>Dig NLB 130 nm</td>
<td>603392</td>
<td>36.9</td>
<td>1.046</td>
<td>0.0055</td>
<td>2.5%</td>
<td>2.5%</td>
<td>7.5%</td>
<td>15%</td>
</tr>
<tr>
<td>Dig NLB 45 nm</td>
<td>603392</td>
<td>11.1</td>
<td>1.064</td>
<td>0.0221</td>
<td>42.5%</td>
<td>62.5%</td>
<td>87.5%</td>
<td>97.5%</td>
</tr>
<tr>
<td>Dig NLB (\theta) 130 nm</td>
<td>550400</td>
<td>24.5</td>
<td>1.065</td>
<td>0.0455</td>
<td>10%</td>
<td>25%</td>
<td>37%</td>
<td>45%</td>
</tr>
<tr>
<td>Dig NLB (\theta) 45 nm</td>
<td>550400</td>
<td>6.977</td>
<td>1.013</td>
<td>0.0607</td>
<td>7.5%</td>
<td>7.5%</td>
<td>7.5%</td>
<td>15%</td>
</tr>
<tr>
<td>Analog NLB 45 nm</td>
<td>154752</td>
<td>2.36e-05</td>
<td>1.014</td>
<td>0.0837</td>
<td>32.5%</td>
<td>40%</td>
<td>60%</td>
<td>70%</td>
</tr>
<tr>
<td>Analog NLB 16 nm</td>
<td>154752</td>
<td>1.35e-05</td>
<td>1.223</td>
<td>0.1076</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>Dual NLB 45 nm</td>
<td>240512</td>
<td>5.42e-05</td>
<td>1.047</td>
<td>0.0879</td>
<td>32.5%</td>
<td>50%</td>
<td>65%</td>
<td>82.5%</td>
</tr>
<tr>
<td>Dual NLB 16 nm</td>
<td>240512</td>
<td>3.10e-05</td>
<td>1.062</td>
<td>0.0427</td>
<td>5%</td>
<td>10%</td>
<td>15%</td>
<td>22.5%</td>
</tr>
</tbody>
</table>
Chapter 7

Conclusions And Future Work

7.1 Conclusions

An offshoot of this work is the inherent difficulty observed in side channel attacks when it is applied to linear functions. It was shown that attacks on single power traces are impractical on modern cryptography algorithms. It also gave a demonstration on how the SHA-3 \( \theta \) plane creation provides strong mitigation to any form of DPA. The strength of linear functions for not leaking large amounts of information on input data is clearly demonstrated when 100% accuracy is not achieved even with 500,000 power traces on the base SHA-3 circuit. SHA-3 is a hash algorithm that can only be effectively attacked from input to output, not output to input as in symmetric encryption algorithms. Non-linear functions, such as S-boxes, have long been seen as prime attack points in SCA. These attacks can be done with very few power traces, a couple thousand in the case of AES, in a very short amount of time. SHA-3 also has this weakness, which could be argued if it is a weakness or not, as all cryptography algorithms need a non-linear permutation, but it comes after a linear function. Because of this, all attacks on SHA-3 have to target the \( \theta \) plane creation in some form to obtain the unknown bits of the input data. It suffices to say that cryptography systems could become much more SCA resistant if they were designed with the non-linear function in the middle of the algorithm, sandwiched between two linear functions that are similar to the \( \theta \) plane creation step of SHA-3.

Furthermore, it goes to reason that a much simpler and smaller linear function could provide much more SCA resistance to the base SHA-3 algorithm. As was seen as a common theme in the results, the key bytes located in the first lane were the easiest to guess. This has to do with there being more known bits in the first lane to unknown bits. Mixing of less
known bits to unknown bits could provide a better result in mitigation that can drive down correlation coefficients and confidence ratios. With this result, it makes sense to require a SHA-3 state to have a key that can fill two planes (640 bits) to reduce the number of known to unknown bits from 4:1 to 3:2.

SCA with linear functions gave properties that are normally not discussed in the research community. This is an extremely useful approach to designing mitigations. The idea that the highest correlating key guess will be the same magnitude as its inverse, but with reverse sign, gives an important result for mitigations. If both values are processed together, the correlations can be seen to almost cancel each other out, or at least level out correlation coefficients. The best mitigations were those that used dual-core systems to process both the regular and inverse data. The XNOR mitigation used minimal additional gates and provided huge SCA resistance as it was designed to only perform the inverse function of the linear function. Making linear functions that have both small hardware footprints, but still mix known and unknown bits as seen in $\theta$, can lead to mitigation techniques that follow the same smaller footprint. The non-linear function, and any additional linear functions that are sandwiched between the mitigation linear functions, can be much more complicated and resource intensive to provide any needed mathematical strength. These more complicated inner circuits do not need to be replicated for mitigation as the first and last linear function of the round provide the full defense at hiding what is being processed in the algorithm.

This work also shows that SCA prevention techniques are better thought of during algorithm creation and not in after the fact design solutions. The WDDL family of mitigations provide promising results, but continue to add more resources and power consumption as new vulnerabilities are found. SHA-3 was a hard to attack algorithm due to the leading linear function that forces any attack on the algorithm to focus on it. This was a lucky coincidence, as the authors of SHA-3 mainly discuss how effective the non-linear function can be mapped to dual-rail logic for SCA mitigations. It makes sense when thinking of a physical attack that the physical components should be designed in such a way as to be inherently strong; however, there still is some useful architectures that could be created without specific knowledge of the cryptographic systems that will run on them. The
MTNLB, part of a family of NLBs that could also be useful in SCA mitigations, components work as reconfigurable circuits that can learn any function much like a LUT. They are smaller than LUTs and consume much less power; yet, they are not an immediate solution as this work showed the easiest to attack system was made out of single MTNLBs at 16 nm. The usefulness comes from their small size and variable nature, allowing for multiple MTNLBs to be instantiated to provided a remarkable CPA resistant circuit. In ASIC design, the area overhead of a factor of 3.11 still makes a competitive solution to WDDL logic implementations, with a factor up to 4, but with far less consumed power.

7.2 Future Work

This work proved the usefulness of MTNLBs on linear functions by way of using data and its inverse in computations to mask what was being processed in the algorithm. It would be useful to apply the other NLBs presented in [24] and see if the properties found in the other NLB implementations can provide better mitigations. Also, more analysis on NLBs could be useful to figure out a way to mask power consumption within a single NLB. The NLB can be modified to hide the power leakages that were seen in this work; making a component that uses less power and area while providing mitigation against power analysis.

It was suggested that a two plane key, 640-bits, could prevent a successful attack on $\theta$. Implementing this input configuration and moving attacks to focus on $\chi$ can provide a better full picture idea on possible full SHA-3 mitigations.

Since linear functions were exclusively looked at, looking at algorithms with easier to attack non-linear functions, such as AES, could produce useful results on the mitigation strength of NLBs. Because non-linear functions are easier to attack, good performance with the NLB family on non-linear functions would provide compelling evidence to design FPGA architectures with secure NLB cells to complement the LUT logic, or completely replace the LUTs.
Bibliography


