M.Sc. Thesis

Asynchronous Logic as Counter Measure against Power Analysis Attacks

P.D. Tamerus

Abstract

Cryptographic devices are vulnerable to so-called Side Channel Attacks. As attackers become smarter, hardware designers and chip manufacturers need to keep up with the security demands against these Side Channel Attacks. Side Channel Attacks such as timing analysis, power consumption analysis or electromagnetic analysis, are based upon the principle that the attacker observes the behavior of the side channel (power, electromagnetic emission etc.) while a cryptographic device is performing its operations. The side channel reveals the attacker valuable information about the secret key which ultimately enables the attacker to derive the secret key.

There are several counter measures that minimize the side channel information. This thesis analyzes the influence of using asynchronous logic as a practical countermeasure against Power Analysis attacks by implementing the AES Rijndael cryptographic algorithm in a FPGA device. A Power Analysis attack is a form of a Side Channel Attack where the attacker observes the behavior of the the power consumption during a cryptographic operation.

A feasible asynchronous logic design style is chosen and implemented in a FPGA. In order to compare its effectiveness, a synchronous (clocked) hardware design is made in the same design structure of the AES algorithm. Power Analysis attacks are performed on both designs, and the results are compared.
Asynchronous Logic as Counter Measure against Power Analysis Attacks
Investigation of Asynchronous Logic as Counter Measure against Power Analysis Attacks using a AES Rijndael Cipher on FPGA

THESIS

submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER ENGINEERING

by

P.D. Tamerus
born in Wervershoof, The Netherlands

This work was performed in:

Circuits and Systems Group
Department of Microelectronics & Computer Engineering
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology
The undersigned hereby certify that they have read and recommend to the Faculty of Electrical Engineering, Mathematics and Computer Science for acceptance a thesis entitled “Asynchronous Logic as Counter Measure against Power Analysis Attacks” by P.D. Tamerus in partial fulfillment of the requirements for the degree of Master of Science.

Dated: August 28th, 2012

Chairman: ____________________________________________________________________________
Prof.Dr.ir. A.J. van der Veen, CAS, TU Delft

Advisor: ____________________________________________________________________________
Dr.ir. T.G.R.M. van Leuken, CAS, TU Delft

Committee Members: __________________________________________________________________
Dr.ir. A.J. van Genderen, CE, TU Delft

_____________________________________________________________________________________
Ir. J. de Haas, Riscure B.V., Delft

_____________________________________________________________________________________
Ir. C. Butcher, Fox-IT B.V., Delft
Abstract

Cryptographic devices are vulnerable to so-called Side Channel Attacks. As attackers become smarter, hardware designers and chip manufacturers need to keep up with the security demands against these Side Channel Attacks. Side Channel Attacks such as timing analysis, power consumption analysis or electromagnetic analysis, are based upon the principle that the attacker observes the behavior of the side channel (power, electromagnetic emission etc.) while a cryptographic device is performing its operations. The side channel reveals the attacker valuable information about the secret key which ultimately enables the attacker to derive the secret key.

There are several counter measures that minimize the side channel information. This thesis analyzes the influence of using asynchronous logic as a practical countermeasure against Power Analysis attacks by implementing the AES Rijndael cryptographic algorithm in a FPGA device. A Power Analysis attack is a form of a Side Channel Attack where the attacker observes the behavior of the the power consumption during a cryptographic operation.

A feasible asynchronous logic design style is chosen and implemented in a FPGA. In order to compare its effectiveness, a synchronous (clocked) hardware design is made in the same design structure of the AES algorithm. Power Analysis attacks are performed on both designs, and the results are compared.
I want to thank my supervisor Christo Butcher from Fox-IT B.V. for his tremendous support during the whole thesis project, for the many motivating words and for introducing me to the world of Cryptography and the professional security business.

I want to thank Marc Witteman and my supervisor Job de Haas from Riscure B.V. for giving me the great opportunity to learn about Side Channel Attacks, for their warm and generous welcome in the company and for using their outstanding research facilities. Special thanks to Job de Haas, who taught me many aspects of Side Channel Attacks. I also want to thank Dennis Vermoen from Riscure B.V. who helped me with the Verilog PLI implementation and porting it to Riscure’s Side Channel platform Inspector.

Also many thanks to my supervisor Dr.ir. Rene van Leuken who challenged me, supported me and gave me a good vision to complete my thesis work.

Last but not least I want to thank my wife, Alysa who was my support all the way and gave me strength to keep going and finish the job.

P.D. Tamerus
Delft, The Netherlands
August 28th, 2012
# Contents

Abstract ........................................... v
Acknowledgments .................................... vii

1 Introduction ..................................... 1
   1.1 Motivation ................................... 1
   1.2 Goals ....................................... 1
   1.3 Contributions ............................... 2
   1.4 Thesis Outline .............................. 2

2 Background ....................................... 3
   2.1 AES Rijndael Cipher .......................... 3
      2.1.1 Introduction to the AES Rijndael Cipher .. 3
      2.1.2 Galois Field Arithmetic ................... 5
      2.1.3 Round Operations .......................... 7
      2.1.4 Key Expansion ............................ 8
   2.2 Asynchronous Circuits ....................... 9
      2.2.1 Introduction of Asynchronous Circuits .... 9
      2.2.2 Handshake Protocols ...................... 11
      2.2.3 Asynchronous Design Tools ............... 16
      2.2.4 Asynchronous Circuits in Clocked FPGAs 21
      2.2.5 Comparison of Asynchronous Logic Implementations on a FPGA 23
   2.3 Power Analysis Attacks ...................... 25
      2.3.1 Introduction to Power Analysis Attacks ... 25
      2.3.2 Data Dependent Power .................... 25
      2.3.3 Power Models ............................. 27
      2.3.4 Power Analysis Attack Methodologies .... 28
      2.3.5 Statistical Characteristics ............... 33
      2.3.6 Counter Measures ........................ 38
      2.3.7 Side Channel Analysis Tool .............. 39

3 Hardware Designs ................................. 41
   3.1 Sasebo-GII Side Channel Attack Platform .... 41
      3.1.1 Introduction to Sasebo-GII Side Channel Attack Evaluation Board 41
      3.1.2 Cryptographic FPGA ........................ 41
      3.1.3 AES Component ............................ 43
   3.2 AES Cores .................................... 45
      3.2.1 Synchronous AES Core ..................... 45
      3.2.2 Synchronous AES Core Components ....... 45
      3.2.3 Asynchronous AES core ................. 49
      3.2.4 Asynchronous AES Core Components ...... 49
4 Power Analysis attacks

4.1 Introduction .......................... 57

4.1.1 Setup .............................. 57
4.1.2 CPA Attack Point .................... 58
4.1.3 DPA Attack Point ..................... 59

4.2 Attacks on Synchronous AES .................. 60

4.2.1 CPA Attack .......................... 62
4.2.2 DPA Attack .......................... 64

4.3 Attack on Asynchronous AES .................. 66

4.3.1 CPA Attack .......................... 68
4.3.2 DPA Attack .......................... 70

4.4 Comparison of Power Analysis attacks ............. 73

4.4.1 Ranking Based Comparison ................. 73
4.4.2 Noise-based Comparison .................. 74
4.4.3 Significance and Estimation of Number of Traces .... 79

5 Conclusion and Future Work .................. 83

5.1 Conclusion ................................ 83
5.1.1 Meeting the Goals ....................... 83
5.2 Future Work and Final Thoughts ............... 84

A Appendix .......................... 91

A.1 Simulation based Power Model .................. 91

A.1.1 Embedding Side Channel Analysis in the hardware design process .... 91
A.1.2 Introduction to Verilog PLI .................. 91
A.1.3 Power Simulation Model ................... 94
A.1.4 Results and Performance ................... 96
A.1.5 Conclusions and Future Work ............... 106

A.2 Subtracting Noise Trace ....................... 107

A.2.1 Noise trace ........................... 107
A.2.2 Asynchronous Signal Subtracted With Noise .......... 107

B Appendix: Attack results .................. 121

B.1 Attacks Synchronous AES ..................... 121

B.1.1 CPA 10000 traces ........................ 121
B.1.2 DPA 10000 traces ........................ 126

B.2 Attacks Asynchronous AES .................... 131

B.2.1 CPA 10000 traces ........................ 131
B.2.2 DPA 10000 traces ........................ 136
B.2.3 CPA 50000 traces ........................ 141
B.2.4 DPA 50000 traces ........................ 146

B.3 Key Byte Ranking .......................... 151
## List of Figures

2.1 Schematic view of the AES Rijndael cipher ........................................ 4
2.2 Muller-C element ................................................................. 9
2.3 Four phase bundled data Muller Pipeline ...................................... 10
2.4 Four phase handshake protocol ............................................... 11
2.5 Four phase bundled data Muller Pipeline with Functional blocks and Matched Delay ......................................................... 12
2.6 Two phase handshake protocol ............................................... 12
2.7 Two phase bundled data Muller pipeline with Function blocks and Matched Delay ......................................................... 12
2.8 Dual-rail protocol .................................................................. 13
2.9 Enhancements on Dual-rail protocol: Dual Spacer Dual-rail ............. 14
2.10 Four phase dual-rail Muller Pipeline ....................................... 14
2.11 Dual-rail DIMS (NCL-D) implementation of an OR gate ............... 15
2.12 Dual-rail NCL-X implementation of an OR gate ......................... 16
2.13 Petrify output van Muller-C element-based controller .................. 18
2.14 Balsa handshake schematic of the Sbox ................................... 19
2.15 Balsa handshake schematic of the sboxminv module ................... 20
2.16 Balsa handshake schematic of the Sbox Affine module ................. 20
2.17 Balsa GTKwave view of handshaking of Sbox ............................ 21
2.18 Interconnect of a Virtex 5 v30 FPGA between switch boxes and slices 22
2.19 Resource Utilization of one Sbox using different asynchronous design techniques ................................................................. 24
2.20 CMOS inverter ................................................................... 26
2.21 Current drawn by CMOS inverter and input output voltage plot, PSpice simulation ................................................................. 27
2.22 Steps of CPA and DPA attacks .............................................. 30
2.23 Confidence interval Fisher transformed $\rho = 0.82$ ...................... 35
2.24 Number of calculated traces versus correlation coefficient ........... 37
2.25 Normalized difference vs correlation coefficient ......................... 37
2.26 Side Channel Analysis Tool: Inspector from Riscure B.V. ............ 40

3.1 Schematic of the Sasebo-GII board ........................................ 42
3.2 Schematic of the Sasebo-GII Virtex5 Crypto FPGA ....................... 42
3.3 Post Place and Route simulation AES Comp with trigger pins signals 43
3.4 Abstract toplevel view synchronous AES component .................. 44
3.5 Schematic top level view of the synchronous AES core ................ 45
3.6 Schematic view of the AES SBox ........................................... 46
3.7 Mix Column operation for one column (32 bits operation) .......... 47
3.8 KeyExpansion schematic ..................................................... 48
3.9 Schematic view of AES CORE build with PetriNet controllers .... 49
3.10 Passive and active port controllers AES core .......................... 50
3.11 Port controllers ................................................................. 50
3.12 Matched Delay component in Synplify Pro Technology view
3.13 LUTs in CLBs using RPM’s
3.14 Matched Delay over M0 (multiplicator) of Sbox 0 in Subbytes component
3.15 PetriNet controller in a single LUT
3.16 Schematic of asynchronous MUX implementation
3.17 Schematic of asynchronous DMUX implementation
3.18 Schematic of PetriNet controllers
3.19 Schematic of MixColumn operation using PetriNet controllers
3.20 Schematic of Key Expansion module using PetriNet controllers

4.1 Raw power trace of synchronous AES cipher on Sasebo GII
4.2 Attack point synchronous design, the last AES round, showing 64 traces
4.3 Correlation estimation key byte 1 versus samples using CPA with 10000 traces
4.4 CPA Key byte 1 (value 0x0D0, 208) correlation vs number of traces on synchronous AES cipher
4.5 Correlation estimation key byte 16 versus samples using CPA with 10000 traces
4.6 CPA Key byte 16 (value 0x0A6, 166) correlation vs number of traces on synchronous AES cipher
4.7 Correlation estimation key byte 1 versus samples using DPA with 10000 traces
4.8 DPA Key byte 1 (value 0x0D0, 208) correlation vs number of traces on synchronous AES cipher
4.9 Correlation estimation key byte 16 versus samples using DPA with 10000 traces
4.10 DPA Key byte 16 (value 0x0A6, 166) correlation vs number of traces on synchronous AES cipher
4.11 Raw trace of asynchronous AES cipher on SaseboGII
4.12 Attack point asynchronous design, the last AES round, showing 64 traces
4.13 Correlation estimation key byte 1 versus samples using CPA with 50000 traces
4.14 CPA Key byte 1 (value 0x0D0, 208) correlation vs number of traces on synchronous AES cipher
4.15 Correlation estimation key byte 16 versus samples using CPA with 50000 traces
4.16 CPA Key byte 16 (value 0x0A6, 166) correlation vs number of traces on synchronous AES cipher
4.17 Correlation estimation key byte 1 versus samples using DPA with 50000 traces
4.18 DPA Key byte 1 (value 0x0D0, 208) correlation vs number of traces on synchronous AES cipher
4.19 Correlation estimation key byte 16 versus samples using DPA with 50000 traces
4.20 DPA Key byte 16 (value 0xA6, 166) correlation vs number of traces on synchronous AES cipher ............................................. 72
4.21 Ranking of all key bytes of CPA attack of both synchronous and asynchronous AES hardware implementation ................................. 73
4.22 Signal and Noise of CPA attack synchronous AES .......................... 75
4.23 Signal and Noise of CPA attack asynchronous AES after 10000 traces . 76
4.24 Signal and Noise of CPA attack asynchronous AES ......................... 77
4.25 Probability Density Function synchronous AES of key byte 1 and key byte 16 estimate using 10000 traces ........................................ 79
4.26 Probability Density Function asynchronous AES key byte 1 and key byte 16 estimate using 10000 traces ........................................ 80
4.27 Probability Density Function asynchronous AES key byte 1 and key byte 16 estimate using 50000 traces ...................................... 81

A.1 Verilog PLI sbox input resp. 0x29, 0x23, 0xBE showing data dependency using PLI power model .................................................. 92
A.2 ModelSim simulator using Verilog PLI to make a power simulation .... 94
A.3 PLI power trace of the synchronous AES cipher .............................. 96
A.4 Signal of last round of Synchronous AES PLI simulation ................... 97
A.5 PLI synchronous AES attack result keybyte 1 ................................ 97
A.6 CPA Key byte 1 (value 0xD0, 208) correlation vs number of traces on Synchronous AES cipher ............................................. 98
A.7 PLI synchronous AES attack result key byte 16 ......................... 98
A.8 CPA Key byte 16 (value 0xA6, 166) correlation vs number of traces on Synchronous AES cipher ............................................. 99
A.9 Signal and Noise plot of PLI synchronous AES CPA attack ............... 100
A.10 Schematic view of AES Comp build with PetriNet controllers ........... 101
A.11 PLI power trace of the asynchronous AES cipher ............................ 101
A.12 Zoomed in view of the PLI power trace of the asynchronous AES cipher 102
A.13 Signal of last round of asynchronous AES PLI simulation .............. 102
A.14 PLI asynchronous AES attack result key byte 1 .......................... 103
A.15 CPA Key byte 1 (value 0xD0, 208) correlation vs number of traces on Asynchronous AES cipher ............................................. 103
A.16 PLI asynchronous AES attack result key byte 16 ......................... 104
A.17 CPA Key byte 16 (value 0xA6, 166) correlation vs number of traces on Asynchronous AES cipher ............................................. 104
A.18 Signal and Noise plot of PLI asynchronous AES CPA attack ............... 105
A.19 Verilog PLI simulation time and file size .................................... 106
A.20 Noise trace, no AES cipher implemented in hardware ................. 107
A.21 Absolute Noise trace, no AES cipher implemented in hardware ......... 108
A.22 Absolute Asynchronous AES trace ........................................... 108
A.23 Asynchronous trace with subtracted noise trace ............................ 108
A.24 CPA Key byte 1 (value 0xD0, 208) correlation vs number of traces on asynchronous AES cipher ............................................. 109
A.25 CPA Key byte 16 (value 0xA6, 166) correlation vs number of traces on asynchronous AES cipher ........................................ 109
A.26 DPA Key byte 1 (value 0xD0, 208) correlation vs number of traces on asynchronous AES cipher ........................................ 110
A.27 DPA Key byte 16 (value 0xA6, 166) correlation vs number of traces on asynchronous AES cipher ........................................ 110
A.28 Ranking of all key bytes CPA and DPA attack vs number of traces (50000) on Asynchronous AES cipher ......................... 111
A.29 Partial key hypothesis of key byte 1-8 using CPA attack, Asynchronous AES with Noise subtracted, 50000 traces ................. 113
A.30 Partial key hypothesis of key byte 9-16 using CPA attack, Asynchronous AES with Noise subtracted, 50000 traces ................. 114
A.31 Ranking Partial key hypothesis of key byte 1-8 using CPA attack, Asynchronous AES with Noise subtracted, 50000 traces ................. 115
A.32 Ranking Partial key hypothesis of key byte 9-16 using CPA attack, Asynchronous AES with Noise subtracted, 50000 traces ................. 116
A.33 Partial key hypothesis of key byte 1-8 using DPA attack, Asynchronous AES with Noise subtracted, 50000 traces ................. 117
A.34 Partial key hypothesis of key byte 9-16 using DPA attack, Asynchronous AES with Noise subtracted, 50000 traces ................. 118
A.35 Ranking Partial key hypothesis of key byte 1-8 using DPA attack, Asynchronous AES with Noise subtracted, 50000 traces ................. 119
A.36 Ranking Partial key hypothesis of key byte 9-16 using DPA attack, Asynchronous AES with Noise subtracted, 50000 traces ................. 120

B.1 Partial key hypothesis of key byte 1-8 using CPA attack, synchronous AES ................................................................. 122
B.2 Partial key hypothesis of key byte 9-16 using CPA attack, synchronous AES ................................................................. 123
B.3 Ranking Partial key hypothesis of key byte 1-8 using CPA attack, synchronous AES ................................................................. 124
B.4 Ranking Partial key hypothesis of key byte 9-16 using CPA attack, synchronous AES ................................................................. 125
B.5 Partial key hypothesis of key byte 1-8 using DPA attack, synchronous AES ................................................................. 127
B.6 Partial key hypothesis of key byte 9-16 using DPA attack, synchronous AES ................................................................. 128
B.7 Ranking Partial key hypothesis of key byte 1-8 using DPA attack, synchronous AES ................................................................. 129
B.8 Ranking Partial key hypothesis of key byte 9-16 using DPA attack, synchronous AES ................................................................. 130
B.9 Partial key hypothesis of key byte 1-8 using CPA attack, asynchronous AES, 10000 traces ................................................................. 132
B.10 Partial key hypothesis of key byte 9-16 using CPA attack, asynchronous AES, 10000 traces ................................................................. 133
B.11 Ranking Partial key hypothesis of key byte 1-8 using CPA attack, asynchronous AES, 10000 traces

B.12 Ranking Partial key hypothesis of key byte 9-16 using CPA attack, asynchronous AES, 10000 traces

B.13 Partial key hypothesis of key byte 1-8 using DPA attack, asynchronous AES, 10000 traces

B.14 Partial key hypothesis of key byte 9-16 using DPA attack, asynchronous AES, 10000 traces

B.15 Ranking Partial key hypothesis of key byte 1-8 using DPA attack, asynchronous AES, 10000 traces

B.16 Ranking Partial key hypothesis of key byte 9-16 using DPA attack, asynchronous AES, 10000 traces

B.17 Partial key hypothesis of key byte 1-8 using CPA attack, asynchronous AES, 50000 traces

B.18 Partial key hypothesis of key byte 9-16 using CPA attack, asynchronous AES, 50000 traces

B.19 Ranking Partial key hypothesis of key byte 1-8 using CPA attack, asynchronous AES, 50000 traces

B.20 Ranking Partial key hypothesis of key byte 9-16 using CPA attack, asynchronous AES, 50000 traces

B.21 Partial key hypothesis of key byte 1-8 using DPA attack, asynchronous AES, 50000 traces

B.22 Partial key hypothesis of key byte 9-16 using DPA attack, asynchronous AES, 50000 traces

B.23 Ranking Partial key hypothesis of key byte 1-8 using DPA attack, asynchronous AES, 50000 traces

B.24 Ranking Partial key hypothesis of key byte 9-16 using DPA attack, asynchronous AES, 50000 traces

B.25 ranking of all key bytes CPA and DPA attack vs number of traces on synchronous AES cipher

B.26 ranking of all key bytes CPA and DPA attack vs number of traces (50000) on asynchronous AES cipher

B.27 Ranking Partial key bytes synchronous AES using CPA and DPA attack after 10000 traces

B.28 Ranking Partial key bytes asynchronous AES using CPA and DPA attack after 50000 traces
List of Tables

2.1 Truth table Muller-C element ........................................ 10
2.2 Dual-rail Encoding ..................................................... 13
2.3 Truth table used to create Dual-rail DIMS functions ............. 16
4.1 Setup ........................................................................ 57
4.2 SNR synchronous CPA ............................................... 75
4.3 SNR asynchronous CPA after 10000 traces ....................... 76
4.4 SNR asynchronous CPA after 50000 traces ..................... 77
A.1 Riscure’s Inspector *.trs, trace format ......................... 95
A.2 SNR synchronous CPA PLI Verilog ............................... 99
A.3 SNR asynchronous CPA PLI Verilog ......................... 105
This chapter gives an introduction to the thesis project. Section 1.1 gives a motivation. Section 1.2 describes the goals of this thesis, contribution is briefly described in section 1.3 and section 1.4 gives the outline of the thesis.

1.1 Motivation

Cryptographic devices are vulnerable to so-called Side Channel Attacks. As attackers become smarter, hardware designers and chip manufacturers constantly need to keep up with the security demands against these attacks. Therefore it is important for chip manufactures and hardware designers to gain more insight in how to build secure designs with acceptable costs.

A Power Analysis attack is a form of Side Channel Attacks where the attacker observes the behavior of the power consumption while the device is performing cryptographic algorithms. This thesis investigates the effect of using asynchronous logic over the commonly used synchronous (clocked) logic under exposure of Power Analysis attacks on FPGA devices.

Asynchronous logic has several interesting properties, like low power emission, low power consumption, and high operational speed, that make it an interesting technology that can serve as a countermeasure to Power Analysis attacks. At the time of writing there are not many practical publications about asynchronous logic as a countermeasure to Power Analysis attacks. Several solutions have been proposed but there still lacks written material about cost effective asynchronous logic being used as a countermeasure to Power Analysis attacks.

This thesis will analyze several asynchronous design technologies. The cryptographic algorithm that is chosen to be implemented is the AES Rijndael encryption algorithm. The best cost efficient asynchronous logic design style is chosen and implemented in a FPGA and Power Analysis attacks results are compared with attacks on a similar clocked (synchronous) design.

1.2 Goals

The goals of this thesis project are listed below:

1. Analyze different asynchronous logic design styles and find a feasible and cost efficient solution that fits in the Virtex5 FPGA on Sasebo-GII Side Channel Attack platform.

2. Use this cost efficient logic design style to implement an asynchronous AES Rijndael encryption core. Also create a synchronous AES core that has a comparable
hardware layout structure.

3. Analyze and compare the Power Analysis attack results of the asynchronous AES core and the synchronous AES core.

4. Evaluate the significance of a feasible asynchronous logic implementation as a countermeasure to Power Analysis attacks and draw conclusions.

1.3 Contributions

The contributions of this thesis are listed below:

1. Publishing about the first asynchronous AES cipher implemented on the Sasebo GII Side Channel Attack platform. Comparisons are given between a similar synchronous design implemented on this platform.

2. Publishing about a feasible, cost acceptable solution of asynchronous logic as a countermeasure to Power Analysis attacks.

3. Publishing the design issues regarding an implementation of asynchronous logic in a synchronous clocked FPGA device.

4. Related work is done to create a simulation-based power analysis of hardware designs or even selective parts of a design, based on Post-Place and route simulation. This simulation based approach would bring Side Channel Attack analysis into the design chain, which is very interesting for hardware designers in security applications and chip manufacturers. More future work is required for this. Appendix A.1 describes this work and mentions the open issues. Attack results of simulation-based power traces of both the asynchronous and the synchronous AES ciphers are given.

1.4 Thesis Outline

Chapter 2 discusses the fundamental background information needed for this thesis. A brief introduction is given of the AES Rijndael algorithm, asynchronous logic and Power Analysis attacks. Chapter 3 describes the hardware design implementation of both the synchronous as the asynchronous design. Chapter 4 discusses the Power Analysis setup and attack results performed on both designs, and a conclusion is made in chapter 5.

Appendix A.1 discusses the simulation based power traces, and Appendix A.2 discusses a noise trace subtraction from a power traces of a full encryption round of a asynchronous AES. Appendix B shows attack results of all key bytes of both designs.
This chapter contains the background information that covers the necessary material to follow the discussion in the remaining chapters of this document. Section 2.1 explains the background information of the AES Rijndael cipher. Section 2.2 briefly discusses asynchronous hardware designs. Section 2.3 discusses Side Channel Attacks.

2.1 AES Rijndael Cipher

This section will briefly discuss the architecture and the building blocks of the AES Rijndael Encryption cipher. Section 2.1.1 gives a brief introduction of the AES Rijndael cipher. Section 2.1.2 discusses the Galois Field arithmetic that is used in all building blocks of the AES cipher. Section 2.1.3 will discuss the AES Round operations and section 2.1.4 discusses the key expansion part of the AES cipher.

2.1.1 Introduction to the AES Rijndael Cipher

The AES Rijndael cipher was designed by Joan Daemen and Vincent Rijmen [10]. The AES Rijndael cipher is a symmetric key-iterated block cipher. It was standardized by the NIST (National Institute of Standards and Technology) as AES in November 2001. The standardized AES cipher has a block size of 128 bits and key lengths of 128, 196 or 256 bits. Depending on the key length, the AES is called AES-128, AES-196 or AES-256. From now on the AES-128 cipher is referred to as the AES cipher in this document.

The data blocks of 128 bits are called states. Figure 2.1 shows a schematic overview of the AES cipher.
Figure 2.1: Schematic view of the AES Rijndael cipher
The AES cipher can be divided into two parts: a key expansion part and the AES round operations. An AES-128 encryption consists of 10 rounds. It starts with an addition (Add Round Key) operation with the secret key and the plain text, then nine AES rounds operations are executed followed by a final round. See figure 2.1.

A single AES round operation consists of four partial operations:

1. Sub Bytes
2. Shift Row
3. Mix Column
4. Add Round Key

The Key Expansion module generates a round key for each round which is iteratively calculated by the previous round key. This round key is added by the operation 'Add Round Key'. As can be seen in figure 2.1, the last round does not have a Mix Column operation.

2.1.2 Galois Field Arithmetic

This paragraph discusses the field arithmetic upon which the AES Rijndael cipher is based. In the AES cipher, bytes are considered as polynomials that are elements in a finite Galois Field $GF(2^8)$ [9], see equation 2.1. This means that a byte is a polynomial in $GF(2^8)$ with coefficients in $GF(2)$. Coefficients in $GF(2)$ can only have values that are either 0 or 1, similar to single bit values.

$$b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x^1 + b_0$$

For example, the hexadecimal value $2B_{\text{hex}}$ (0010 1011) corresponds with $x^5 + x^3 + x + 1$

2.1.2.1 Addition

The addition of two bytes, i.e. the addition of two bytes represented as polynomials in $GF(2^8)$, is done by adding the coefficients modulo 2, i.e. $1 + 1 = 0$. This is the same as an XOR operation, denoted with $\oplus$. For example:

$$57_{\text{hex}} \oplus 83_{\text{hex}}$$

$$\equiv 01010111 \oplus 10000011$$

$$\equiv (x^6 + x^4 + x^2 + x + 1) \oplus (x^7 + x + 1) \equiv x^7 + x^6 + x^4 + x^2$$

Note that subtraction is the same as addition.
2.1.2.2 Multiplication

Multiplication of two bytes representing polynomials in $GF(2^8)$, denoted with $a(x) \otimes b(x)$, requires an irreducible polynomial of degree 8 (9 bits).

AES Rijndael specified the following irreducible polynomial $m(x) = x^8 + x^4 + x^3 + x + 1$, i.e. $11B_{hex}$. Using this polynomial, a representation of the Galois field $GF(2^8)$ has been created.

The multiplication of $a(x)$ and $b(x)$ is then defined as the algebraic product of the polynomials modulo the irreducible polynomial $m(x)$:

$$q(x) = a(x) \otimes b(x) \Leftrightarrow q(x) \equiv a(x) \times b(x)(mod(m(x))) \quad (2.5)$$

Example: $57_{hex} \otimes 83_{hex} \equiv 01010111 \otimes 10000011 \equiv (x^6 + x^4 + x^2 + x + 1) \otimes (x^7 + x + 1)$

$$= (x^{13} + x^{11} + x^9 + x^8 + x^7) \oplus (x^7 + x^5 + x^3 + x^2 + x) \oplus (x^6 + x^4 + x^2 + x + 1)$$

Assuming that $m(x) = x^8 + x^4 + x^3 + x + 1 = 0$, the following relations are obtained:

1. $x^8 = x^4 + x^3 + x + 1$
2. $x^9 = x^5 + x^4 + x^2 + x$
3. $x^{10} = x^6 + x^5 + x^3 + x^2$
4. $x^{11} = x^7 + x^6 + x^4 + x^3$
5. $x^{12} = x^8 + x^7 + x^5 + x^4$ substituting $x^8$ becomes: $(x^4 + x^3 + x + 1) + x^7 + x^5 + x^4 = x^7 + x^5 + x^3 + x + 1$
6. $x^{13} = x^8 + x^6 + x^4 + x^2 + x$ substituting $x^8$ becomes: $(x^4 + x^3 + x + 1) + x^6 + x^4 + x^2 + x = x^6 + x^3 + x^2 + 1$

With these relations concerning the modulo $m(x)$, the modulo $m(x)$ of the (intermediate) multiplication result $x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1$ can be calculated as follows:

1. $x^{13} \oplus x^{11} = (x^6 + x^3 + x^2 + 1) \oplus (x^7 + x^6 + x^4 + x^3) = x^7 + x^4 + x^2 + 1$
2. $(x^7 + x^4 + x^2 + 1) \oplus x^9 = (x^7 + x^4 + x^2 + 1) \oplus (x^5 + x^4 + x^2 + x) = x^7 + x^5 + x + 1$
3. $(x^7 + x^5 + x + 1) \oplus x^8 = (x^7 + x^5 + x + 1) \oplus (x^4 + x^3 + x + 1) = x^7 + x^5 + x^4 + x^3$
4. then the remaining terms (below $x^8$) are added ($\oplus$), with the last result:
   \(x^7 + x^5 + x^4 + x^3\) + \((x^6 + x^5 + x^4 + x^3 + 1)\) = $x^7 + x^6 + 1$

The answer is: $x^7 + x^6 + 1 \mod (x^8 + x^4 + x^3 + x + 1)$ which is $C1_{hex}$ in hexadecimal byte representation.
2.1.3 Round Operations

This section briefly discusses each of the four AES round operations.

2.1.3.1 Sub Bytes

The Sub Bytes operation is the only non-linear transformation in the AES cipher. The Sub Bytes operation is a bricklayer transformation that consist of sixteen bricklayer functions that are applied to each byte of the state. A bricklayer function is a function that can be decomposed into several Boolean functions operating independently on subsets of bits of the input vector[10]. These non-linear functions are known as Sboxes. When the bricklayer functions are linear they are called D-boxes (diffusion boxes). Since Sub Bytes operates on the whole state (128 bits), it is also called an bricklayer transformation. The Sub Bytes function uses one type of Sbox for all bytes. The creation of the S-box consist of two steps:

1. $x^{-1}$ in $GF(2^8)$ corresponds with the MI (multiplicative inverse) in of $x$ in $GF(2^8)$
2. Bit mangling by an affine transformation. This destroys the correlation on a bit level but maintains a mathematical relation.

2.1.3.2 Shift Row

Shift Row is a boolean permutation that moves the positions of the bytes of the state with the goal to achieve a diffusion of the state bits. This operation shifts the rows of the state cyclically with a certain offset. Each row has a different offset; for row 0 to 3 the following offsets are used: 0, 1, 2, 3, respectively. In hardware this is simply the changing of connections from Sub Bytes output to the next module (Mix Column) input.

2.1.3.3 Mix Column

The Mix Column operation is a bricklayer permutation that operates on the state in a column-wise way. Besides the permutation, values are also changed with D-boxes (linear functions, called diffusion boxes). The columns are each considered as polynomials in $GF(2^8)$ and multiplied modulo $x^4 + 1$ with a fixed polynomial $c(x) = 3x^3 + x^2 + x + 2$

The Mix Column operation can be written as:

$$b(x) = c(x).a(x)mod(x^4 + 1)$$

Equation 2.7 is performed on each column of the state, i.e. four times an Mix Column operation on 32 bits of the state.
2.1.3.4 Add-Round Key

The key addition adds the round key with the state data. As mentioned in section 2.1.2, addition in finite Galois Fields is equal to a bitwise XOR (exclusive OR) operation, the round key is XOR-ed with the state data. The round key is iteratively generated by the key expansion module.

2.1.4 Key Expansion

This section will briefly discuss the Key expansion part of the AES algorithm. The key expansion module uses the secret key as an input to produce 10 round keys. The round key is derived by its predecessor key (round key 0 uses the secret key as input) \( \text{Roundkey}_i = \text{KEYEXP}(\text{Roundkey}_{i-1}) \).

The key expansion module contains four Sbox operations and several XOR operations. Section 3.2.2.5 discusses the key expansion module in more detail.
2.2 Asynchronous Circuits

This section discusses asynchronous circuits. Section 2.2.1 gives a short introduction to asynchronous circuits and introduces the fundamental building block of asynchronous circuits, the Muller-C element. Section 2.2.2 discusses handshake protocols. Section 2.2.3 discusses asynchronous hardware design tools and section 2.2.4 discusses the issues of implementing asynchronous circuits in an FPGA. Section 2.2.5 gives a comparison of different asynchronous design technologies implemented on a Xilinx Virtex 5 FPGA.

2.2.1 Introduction of Asynchronous Circuits

This section gives an introduction to asynchronous circuits and discusses its fundamental building block, the Muller-C element and the Muller-C pipeline.

In synchronous systems, all modules are synchronized through a global clock signal. Within the next tick of the global clock, all modules in the system must have their data ready, i.e. the global clock places a timing constraint on the whole system. In asynchronous logic there is no global clock signal that synchronizes all the modules, instead it synchronizes locally through a handshake protocol.

Asynchronous designs have special properties, some advantages are:

1. Low power consumption
2. Low electro magnetic emission
3. High operation speed
4. No clock distribution problems or skew problems

some disadvantages are:

1. Absence of many asynchronous design tools
2. Existing design tools cannot handle well with clockless designs
3. Not a common used design approach

2.2.1.1 Muller-C and Muller Pipeline

The Muller-C element is a fundamental element in asynchronous circuits. Figure 2.2 is the symbol.

![Muller-C element symbol](image)

Figure 2.2: Muller-C element

The Muller-C element has the property that it indicates when both inputs are high or when both inputs are low. This is a very useful property to synchronize handshake signals.
Table 2.1: Truth table Muller-C element

The boolean equation for the Muller-C element is \( c = ab + ac + bc \). Table 2.1 displays the truth table.

The Muller-C elements can be used to make latch controllers and make a Muller pipeline \([43]\). The Muller pipeline is a fundamental principle in Asynchronous designs. Almost every asynchronous design is based upon the Muller pipeline.

Figure 2.3 shows the Muller pipeline using C-elements.

![Figure 2.3: Four phase bundled data Muller Pipeline](image-url)
2.2.2 Handshake Protocols

This section discusses the various handshake protocols used in asynchronous systems. The two most common types of handshaking protocols will be discussed.

As can be seen in figure 2.3, handshaking is being performed using a handshake protocol. Handshaking can be initiated by both the sender or the receiver. When the sender is initiating the data transfer over a channel it is called a push-channel. When the receiver is asking for new data it is called a pull channel. In section 2.2.3.3 this will be discussed in more detail.

There are two main types of handshaking protocols:

1. bundled data (single rail)
2. dual-rail

2.2.2.1 Bundled Data Protocols

This subsection discusses the bundled data protocol and discusses the two flavors of signaling: the four-phase handshaking protocol and the two-phase handshaking protocol.

In bundled data protocols, the handshake signals 'request' and 'acknowledge' are separated from the data signals. When using functional blocks, delay elements must be used to delay the request signal for the next latch in order for the functional block to be able to complete its computation. In bundled data protocols, delays must be accordingly matched over combinatorial logic of a function block.

Bundled data protocols can be implemented with a two phase or a four phase handshake protocol.

Level Based - Four Phase Handshaking Figure 2.4 shows the four phase handshake protocol in a Muller-Pipeline [12]. The four-phase protocol is based on 'level signaling'. Here the return to zero (RTZ) is overhead in terms of time and power because of more transitions when comparing it with a two phase handshaking protocol.

![Figure 2.4: Four phase handshake protocol](image)

Figure 2.4: Four phase handshake protocol

Figure 2.5 shows an example of a four phase bundled data Muller pipeline with function blocks.
Transition Based - Two Phase Handshaking  Figure 2.6 shows the two phase handshake protocol. The two-phase protocol is based on 'transition signaling'. This means that the control signals are interpreted as events or as transitions [41]. The two phase bundled data approach was pioneered by Ivan Sutherland [43]. The two phase protocol has no return to zero stage, i.e. when the logical level of the request or acknowledge changes, it means start of the next stage. Two phase protocol should lead to faster circuits than four phase and consume even less power than the four phase implementation.

Figure 2.6: Two phase handshake protocol

Figure 2.7 shows a two phase bundled data Muller pipeline. It can be seen that the two-phase handshake protocol uses capture and pass latches. The two-phase protocol requires these special latches because events (transitions) alternate on the capture and pass input. It can be seen as two level sensitive latches operating in an alternating way [41].

Figure 2.7: Two phase bundled data Muller pipeline with Function blocks and Matched Delay
2.2.2.2 Dual-rail Protocols

This subsection discusses the dual-rail protocol.

The dual-rail protocol encodes data with two wires. The dual-rail protocol is known to be Delay Insensitive and is widely used to represent data in self-timed circuits. Dual-rail protocols have the request signal embedded with the data signal. Only an acknowledge signal is used to let the sender know that the data is processed and that new data can be latched into the register. In [42],[41] some good guidelines are given to create self-timed circuits.

Four Phase Dual-Rail Protocol In a dual-rail design, data is encoded with two wires and is called a token. Using the four phase handshake protocol, the protocol alternates between empty and valid tokens. Empty tokens (E) are inserted between proper data to implement the RTZ (return to zero).

Tokens are encoded using two wire pairs: x.f and x.t, according to table 2.2. The empty token (E) is defined as both wires having the logical value zero and is also called the ”spacer”. The valid token (T), i.e. true, is encoded as x.f=0 and x.t =1 and the valid token (F), i.e. false, is encoded as x.f=1 and x.t=0. The token state when both wires are logically 1 is not used in this protocol.

<table>
<thead>
<tr>
<th>Value</th>
<th>d.t</th>
<th>d.f</th>
</tr>
</thead>
<tbody>
<tr>
<td>Empty (E)</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>valid '0' (F)</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>valid '1' (T)</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Not used</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 2.2: Dual-rail Encoding

As can be seen in table 2.2 the dual-rail protocol works in three states: i.e. empty (E), valid '0' (F) and a valid '1' (T). The switching between these states is depicted in figure 2.8.

![Figure 2.8: Dual-rail protocol](image)

The dual-rail protocol mentioned in table 2.2 and figure 2.8 is also called single spacer dual-rail. The single spacer dual-rail protocol always starts any transition from
the spacer (empty token (E) i.e. all zero’s) to a valid token i.e. (T) or (F). The spacer state is also used to indicate that there is no data processed in that time. The up and down switching of the single spacer dual-rail protocol is similar to how a gate always switches. Although the dual-rail encoding balances the switching activity, in practice experiments have shown [40] that it still leaks data.

In [4], [40] a dual-spacer dual-rail protocol is proposed, using two spacers, i.e. {00} all-zero-spacer and {11} all-one-spacer. The choice of spacer can be arbitrary or even random, see figure 2.9.a. Figure 2.9.b shows an refinement for the dual-spacer dual-rail protocol known as alternating spacers.

The latter has the advantage that all bits are switching in each cycle of operation. This has the advantage that the power consumption within each cycle of operation is theoretically perfectly constant.

![Diagram of Dual-rail Arbitrary Order of Spacers](image)

(a) dual-rail arbitrary order of spacers

![Diagram of Dual-rail Alternating Order of Spacers](image)

(b) dual-rail alternating order of spacers

Figure 2.9: Enhancements on Dual-rail protocol: Dual Spacer Dual-rail

A dual-rail protocol can also be implemented in a Muller pipeline. Figure 2.10 shows a four phase dual-rail Muller pipeline using a single spacer dual-rail protocol.

![Diagram of Four Phase Dual-rail Muller Pipeline](image)

Figure 2.10: Four phase dual-rail Muller Pipeline

14
As can be observed in figure 2.10 there is no REQ signal used. This is because the C element detects when there is data coming. The OR gate between the two C elements, indicates that the line is valid or empty. The two lines go to the Muller-C element. When both are 1, ACK is given. When using larger latches, one can simply use an OR gate for each line and use C elements to check all the lines together.

**Two Phase Dual-Rail Protocol** A two phase dual-rail protocol also uses two wires to encode data. The only difference with the four phase dual-rail protocol is that the two phase dual-rail protocol does not have an empty value because a valid message is acknowledged and followed by another message that is acknowledged.

### 2.2.2.3 Dual-Rail Function Blocks

This section discusses the implementation of dual-rail function blocks.

There are several ways to implement dual-rail function blocks. Two well-known are DIMS (or NCL-D) and NCL-X.

**DIMS** Figure 2.11 gives an example of an OR gate using DIMS (NCL-D).

In the DIMS implementation, also known as NCL-D (Null Convention Logic with full completion detection through the dual-rail signals), the C-elements are used in order to make sure that the input signals are valid.

Figure 2.11: Dual-rail DIMS (NCL-D) implementation of an OR gate

The possible states are mentioned in table 2.3. The OR function is created by connecting each state to the correct output (z.t or z.f). For each state of the input, one of the four C-elements will produce a "1". Note that the empty state is valid, i.e. when input a and b are both 00, 00, respectively.

Other functions can also be made. Table 2.3 shows the truth tables of other functions.

**NCL-X** Another dual-rail technique is NCL-X (Null Convention Logic with separate completion detection), proposed by A. Kondratyev and Lwin [21]. This technique is more area efficient than the NCL-D or DIMS implementation. Figure 2.12 shows again the OR gate but then in NCL-X implementation.
As can be seen the dual-rail function selects the correct output for c.t and c.f, and a completion detection part is implemented by using an OR-gate and a three input Muller-C element.

The dual-rail logic is also used in synchronous design [48], [16], proposed by Tiri and Verbauwhede and is known as WDDL (Wave Dynamic Differential Logic).

### 2.2.3 Asynchronous Design Tools

This section discusses asynchronous design tools. Section 2.2.3.1 gives a brief introduction to asynchronous design tools. Section 2.2.3.2 discusses a well-known tool to create asynchronous controllers called Petrify. Section 2.2.3.3 discusses an other asynchronous circuit design tool called Balsa.

#### 2.2.3.1 Introduction to Asynchronous Design Tools

Asynchronous circuit design started to become most popular in the academic world from 1990 to 2003 [35]. The University of Manchester gave a list of twenty six functional tools used for asynchronous designs [50]. Besides this list there are more tools available.
One of the most used tools is Petrify [7] proposed by Cortadella. This tool is used to create asynchronous controllers.

There are also tools that use a high level description language to describe asynchronous circuits. Examples of these are Balsa[11], CHP[29], Tangram [18].

Since Balsa is the only free available tool using a high level description language, and since Petrify is a free tool that is used very often, this thesis project chose to use these two tools in asynchronous hardware design.

2.2.3.2 PetriNets

Cortadella proposed in [6] a model to specify asynchronous controllers called PetriNets. PetriNets are based on STG (Signal Transition Graph). Events of an asynchronous system can be seen as rising and falling transitions of signals.

A rising signal goes from 0 to 1 and is denoted with a +, e.g. sig+. A falling signal is defined as going from 1 to 0 and is denoted with a -, e.g. sig-.

An STG can be described with the graph description as mentioned in the code below.

```plaintext
1 .model hscntrl
  .inputs b a
  .outputs c
  .graph
  a+ c+
  c+ b+ a-
  b- c+
  b+ c-
  a- c-
  c- b- a+
11 .marking { <c-,b-> <c-,a+> }
.end
```

A very helpful tool is VSTGL [46]. VSTGL can also be used to draw STG in a graphical user interface. The STG above describes a Muller-C pipeline controller. It is a Muller-C element with an inverted input ‘b’.

In order to synthesize a PetriNet description, PetriNets can be transformed to an binary encoded state graph (SG) using the Petrify tool [7].

States can be classified into 4 regions:

1. Positive excitation region (ER(sig+))
   includes all states where a rising transition of sig is enabled

2. Negative excitation region (ER(sig-))
   includes all states where a falling transition of sig is enabled

3. Positive quiescent region (QR(sig+))
   includes all states where sig is 1 and sig- is not enabled

4. Negative quiescent region (QR(sig-))
   includes all states where sig is 0 and sig+ is not enabled
Figure 2.13: Petrify output van Muller-C element-based controller

The figure 2.13 shows the STG and the stage graph of the Muller-C element based controller.

From these state graphs (SG) (Figure 2.13.b), the Petrify tool [6] can generate logic equations or generate a Verilog netlist implementation of the controller.

# EQN file for model hscntrl
# Generated by petrify 4.2 (compiled 15-0ct-03 at 3:06 PM)
3 # Outputs between brackets ”[out]” indicate a feedback to input ”out”
# Estimated area = 48.00

INORDER = b a c;
OUTORDER = [c];
8 [c] = c (a + b’) + b’ a;

# No set/reset pins required.

As can be seen in the equation in the output above, for output c, the controller can be implemented as a Muller-C element, [c] = ac + bc + ba, see section 2.2.1.1.

The designer should still be aware of hazardous implementations. Using only logic gates may result in hazardous behavior when implemented with AND and OR gates. For example, when one of the AND gates is slow, there will be a glitch that results in unstable behavior.

Since this equation fits into a single LUT, it can be guaranteed that it has no hazardous behavior. Therefore it is a useful component in the asynchronous FPGA implementation. Read more about this controller in section 2.2.4.4.
2.2.3.3 Balsa

This section discusses a high level hardware design language for asynchronous circuits called Balsa.

The Balsa language [11],[8] is a high level asynchronous design language that compiles 'Balsa' language into a communicating network that is composed of a set of approximately 40 handshake components. These components are connected by channels. Channels are categorized as push- or pull channels.

Figure 2.14 shows the Balsa handshake graph schematic of the Sbox.

The push channels can be recognized by the direction of the data flow as it goes from an active port to a passive port. An active port is marked with a filled black dot, and passive ports are marked with open dots.

The pull channel is the other way around, it goes from a passive port to an active port.

Figure 2.14: Balsa handshake schematic of the Sbox

The number listed at each channel in figure 2.14 corresponds to the order of execution sequence. Figure 2.14 shows a Balsa implemention of an Sbox. The Sbox is first activated by the activate channel. Then the input 'i' is being assigned to variable 'si' using the fetch operator. The next statement can only be executed when this handshake is done (denoted with the sequence operator ‘;’ in the code and in the schematic). Then the second handshake action is performed by a push channel (black dot to a black dot) to the sboxminv module. First sboxminv uses a pull channel to establish a handshake with ‘si’ and reads its value. Then the second handshake statement of sboxminv is a push channel communication to 'sinv' where it writes the result in inv. When sboxminv handshake operations are done, it returns to the sequence operator. The se-
The sequencer establishes the third handshake using a push channel to the 'afftrans' module. Using a pull channel, the 'afftrans' reads variable 'si' and performs the affine operation of the Sbox. When this handshake operation is done, the 'afftrans' module performs the second handshake operation i.e. writing the result using a push channel to variable 'so'. When this is performed, the handshake with the sequencer and the 'afftrans' module is done, and the sequencer is ready to perform its last handshake operation. The sequencer communicates with the fetch operation (→). The fetch operation uses a pull channel to read variable 'so' and uses a push channel to write 'so' to the output. When this is performed the fetch operation handshaking is completed, and it returns to the sequencer. The sequencer is also finished with its handshake operations, and it returns to the activation port and closes the communication.

The sboxminv module and the affine module of figure 2.14 are decomposed as separate modules. Figure 2.15 shows the handshake schematic of sboxminv module and figure 2.16 shows the handshake schematic of the affine transformation module.

![Figure 2.15: Balsa handshake schematic of the sboxminv module](image1)

![Figure 2.16: Balsa handshake schematic of the Sbox Affine module](image2)

Balsa can also simulate the handshake components using a handshake view in GTKwave. Figure 2.17 shows the handshake simulation view of the Sbox performing two sequential Sbox transformation of byte inputs.

20
With the Balsa tool, different asynchronous handshake protocols can be chosen and compiled to a Xilinx Verilog netlist.

2.2.4 Asynchronous Circuits in Clocked FPGAs

This section discusses the issues of implementing asynchronous circuits in an FPGA. Section 2.2.4.1 discusses the FPGA architectural issues. Section 2.2.4.2 discusses timing constraints. Section 2.2.4.3 addresses the need for relative placement macro’s (RPMs). Section 2.2.4.4 discusses the implementation of hazard-free asynchronous controllers.

2.2.4.1 FPGA Architecture

Most commercial FPGA are designed for synchronous circuits and not for asynchronous circuits. The main problem is the mapping process of asynchronous logic on synchronous (clocked) FPGAs. The structure is made in such a way that an output of a slice can not be directly routed to a nearby slice. This is because the connection has to be routed through switch-boxes, which results in inefficient and long delays. Figure 2.18 shows two slices and the interconnect structure with the switching boxes. There is one local switch box and a global switch box. As can be intuitively seen, this architecture is not optimal for routing. For asynchronous circuits where timing and wire delays are important, it is hard and sometimes even infeasible to implement asynchronous logic designs in clocked FPGAs.

Some approaches have been made in making asynchronous FPGAs. In [33], Payne gave a summary of several asynchronous FPGAs. The University of Washington designed the first asynchronous FPGA, MONTAGE [14]. PGA-STC [24] is another approach by U.C.Davis and is targeted for two phase bundled data micro-pipelined based systems. STACC [34] is developed by the University of Edinburgh, and it is targeted for four phase bundled data based system. Another proposal by Manohar and Teifel used
a fine-grain asynchronous pipeline principle in their asynchronous FPGA [47]. This FPGA model is later even made into a 3D asynchronous FPGA.

The problem is that these asynchronous FPGAs do provide better architectural layouts for asynchronous circuits, but in order to use in practice, a good support of development tools is still lacking. For this reason it has been chosen to proceed with a clocked Xilinx FPGA in this project.

2.2.4.2 Timing Constraints

This section discusses the various issues of using timing constraints.

There are two types of timing constraints: global timing constraints and specific, or local timing constraints. In Xilinx FPGA IDE, all timing constraints are assigned in the *.ucf (user constraint file).

In a design that contains only asynchronous logic, there is no need to define a global clock constraint. However the Xilinx tool will try to look for signals in the design and identify them as local clock signals. When a signal is being identified as a clock, and when clock buffers are still available, clock buffers will be inferred on that signal (unless this option is set to 0 in the synthesis setting).

In mixed designs (containing both synchronous circuits and asynchronous circuits), all the asynchronous components have be excluded from the timing analysis performed by the FPGA tool. This is necessary in order to make sure that the combinatorial delay of the asynchronous components will not be included in the maximum path delay that is used to determine the clock frequency by the FPGA tool.

By using the timing ignore (tig) constraint, all paths that fan out from a signal with the tig constraint will be excluded from the timing analysis. The tig constraint is set on all incoming signals of the asynchronous module.
2.2.4.3 Relationally Placed Macro (RPM) for Timing Critical Circuits

This section briefly discusses the Relationally Placed Macro (RPM).

Circuits that are timing critical, such as the matched delay elements, should be implemented using relationally placed macro (RPM). RPM is used for relational placement of elements and can be implemented using the rloc statement. Using the rloc statement the place and route tool is forced to implement the rloc-ed module entity in the specified way. This results in fixed and predictable delays, which is very important for asynchronous designs that need matched delays.

2.2.4.4 Hazard-Free Asynchronous Controllers

This section discusses the need to implement hazard-free asynchronous controllers.

The functional blocks in a bundled data protocol do not have to be free from hazards, since the data is sampled as soon as the handshake signal arrives. The handshake controller modules however need to be hazard-free in order to prevent unpredictable behavior that could result in wrong results. The controller can be made from a Muller-C element. The Muller-C element can be implemented with three AND gates and one three inputs OR gate. For a FPGA implementation this will most like not result in a hazard-free implementation because when one of the gates is slow due to placement or routing, hazards occur. Therefore, it is most convenient to implement the Muller-C element directly in a LUT [15].

The Virtex 5 FPGA (which is used in this project) can implement Muller-C elements up to 5 inputs without a reset and up to 4 inputs with reset. This is achievable since the Virtex 5 FPGA has the nice feature that it contains 6-input LUTs.

2.2.5 Comparison of Asynchronous Logic Implementations on a FPGA

This section discusses the comparison of several asynchronous logic design techniques that are implemented on the Xilinx Virtex5 xc5v30 FPGA that is made available for this thesis project.

The goal of this comparison is to analyze the feasibility and the cost of an asynchronous logic implementation in the Virtex5 xc5v30 FPGA. Several asynchronous logic design styles are discussed, but not all are feasible to be used to design a AES cipher. The cost of using an asynchronous logic design style is also important since security comes with a price, in this case area is a critical cost component.

Figure 2.19 shows the resource utilization of an Sbox according to [51] using different asynchronous logic design techniques. A synchronous version of this Sbox is also included in the comparison, noted as “VHDL Synchronous”.

As can be seen in figure 2.19, both Balsa implementations use many resources, in fact too many resources that it would not fit in the Xilinx Virtex 5 xc5v30 FPGA.

Also the VHDL Dual-Rail DIMS technique is very area consuming and would also not fit in the Virtex 5 xc5v30. Also the VHDL DRL NCL-X would also not fit.

These designs would not fit because there are 20 Sboxes used in the AES cipher, and there is more logic involved for other AES operations (see figure 2.1 in section 2.1.1).
The PetriNet four phase bundled data design however would fit in the FPGA device and is practically equal in terms of resource utilization. For this reason, this design is chosen to be used in this thesis project.
2.3 Power Analysis Attacks

This section discusses the background information on Power Analysis attacks.

Section 2.3.1 gives an introduction to Power Analysis attacks. Section 2.3.2 discusses data dependency of power consumption. Section 2.3.3 discusses power models that are used in Power Analysis attacks. Section 2.3.4 discusses different type of Power Analysis attacks. Section 2.3.5 discusses statistical characteristics of power traces. Section 2.3.6 discusses the counter measures. Section 2.3.7 briefly discusses the software tool that is used to perform the Power Analysis attack.

2.3.1 Introduction to Power Analysis Attacks

This section gives an introduction to Power Analysis attacks.

Since the existence of cryptographic devices, a significant number of attacks have been successfully exploited on them. Paul Kocher is a great pioneer in the academic research area of Side Channel Attacks (SCA) when in 1999 he presented the first attack that used side channel information that determined parts of the cryptographic key [20].

Nowadays many publications have been made, and it is still an active research area with new innovations at the moment of writing.

Power Analysis attacks can be categorized under the non-invasive attacks. This means that the cryptographic device is attacked using only direct accessible interfaces or channels.

There are two categories of non-invasive attacks: active and passive attacks.

Active non-invasive attacks have the strategy to insert a fault in a device without de-packaging it. This can be done by asserting glitches or by changing the temperature of the device.

Passive non-invasive attacks, also called Side Channel Attacks, have the strategy to determine the secret key of a cryptographic device by measuring the execution time of the device, the power consumption or the EM (Electro Magnetic) field.

This thesis focuses primary on the passive non-invasive attacks.

2.3.2 Data Dependent Power

The detection of the secret key is possible because of the dependency between the power consumption and the intermediate values of the cryptographic device. Most cryptographic hardware designs are implemented with complementary CMOS technology. CMOS cells consist of pull-up and pull-down networks, that are constructed in such a way that the pull-up and pull-down network never conduct at the same time for constant input signals. The discussion of an inverter is representable for other logic, since for more complex gates we only need more PMOS and NMOS transistors.

The power consumption of an inverter, see figure 2.20, can be divided into two parts that together form the total power consumption of a cell: static power consumption and dynamic power consumption.

Static power consumption is the power that is consumed when there is no switching activity. When input $V_{in}$ is set to VDD or to GND, there is no direct connection between $V_{dd}$ and GND, and therefore only a small leakage current ($I_{leak}$) is going through the
transistor). The capacitor in figure 2.20 represents the intrinsic capacitances and the extrinsic capacitances of the CMOS cell. Intrinsic capacitances are the capacitances that are connected to the output of the CMOS cell. Extrinsic capacitances represent the capacitances of the wires that are connected to following CMOS cells and the input capacitances of those. The static power can be calculated with $P_{\text{stat}} = I_{\text{leak}} \cdot V_{\text{dd}}$.

The dynamic power consumption is usually the most dominant part of the power consumption. The dynamic power consumption depends on the data that is processed by the CMOS circuit and can have the following transitions:

- $0 \Rightarrow 0$, static
- $0 \Rightarrow 1$, static + dynamic
- $1 \Rightarrow 0$, static + dynamic
- $1 \Rightarrow 1$, static

The dynamic power behavior of the total design is used in the Power Analysis attack.

The dynamic power consumption can be divided in two groups: the charging current and the short-circuit current.

Figure 2.21 shows the PSpice simulation of the CMOS inverter. The following settings are used: $V_{\text{dd}} = 5V$, $Cl = 10fF$, $PMOS/NMOS_{\text{gatewidth}} = 6\mu m$ and $PMOS/NMOS_{\text{gatelength}} = 3\mu m$.

A short circuit appears for a small period of time by the switching of the output, the PMOS and NMOS are then both conducting. This happens for the 0 to 1 and 1 to 0 transition.

When input $V_{in}$ changes from 0 to 1, output $V_{out}$ changes from 1 to 0, and capacitor $Cl$ is discharged by the NMOS, and only the short circuit current is drawn from the power supply.

When input $V_{in}$ changes from 1 to 0, output $V_{out}$ changes from 0 to 1, and capacitor $Cl$ is charged by the PMOS. Now not only the short circuit current is drawn, but also the current that is consumed by the charging of the capacitor.

Figure 2.21 shows the PSpice simulation of the CMOS inverter.
Logic designs typically consist of multiple multistage combinatorial circuits where inputs of these cells typically do not arrive at the same time. These different arrival times lead to temporarily output values, known as glitches or dynamic hazards. Glitches usually propagate through the whole circuit and the occurring of glitches is data depended, which is good for Power Analysis attacks.

2.3.3 Power Models

This section discusses the two most commonly used power models used in Power Analysis attacks.

In Power Analysis attacks, it is generally necessary to map data values that are being processed to the power consumption. The Hamming weight and Hamming distance model are very common models used for this.

2.3.3.1 Hamming Weight Model

The Hamming weight model assumes that the power consumption is proportional to the number of bits that are logically '1' in the data, i.e:

$$HW(D) = \sum_{j=0}^{m-1} d(j), d(j) \in 0, 1$$

Here $m$ is the number of bits. If $D$ contains $m$ independent and uniformly distributed bits, the whole word has an average Hamming weight of $\mu_h = m/2$ and a variance $\sigma^2_h = m/4$ The data values before or after this data value are ignored.
2.3.3.2 Hamming Distance Model

Brier [2] proposed a model that is closer to the power behavior of CMOS technology than the Hamming Weight model.

It is based on the switching of bits from one state to the other, also known as the Hamming Distance model.

The Hamming Distance model assumes that there is no difference between 0 to 1 and 1 to 0 transitions. The Hamming Distance between two values can be determined by:

\[ HD(X, X') = HW(X \oplus X') \]  

(2.9)

The number of flipping bits is described by: \( HW(X \oplus X') \). The Hamming Weight model assumes that when \( X' = 0 \), and \( D \) is an uniform random variable, then \( X \oplus X \) is also uniform distributed and has the same mean and variance as the Hamming Weight model: \( \mu_h = m/2 \) and a variance \( \sigma^2_h = m/4 \) The Hamming Distance model fits the reality of CMOS technology based circuits quite well. That it does not completely represent the entire power consumption of the chip, it approaches the data dependent part quite well.

Brier gave in [2] the following formula to model data dependency:

\[ W = a.HW(X \oplus X') + b \]  

(2.10)

Here all the remaining parts of the power consumption can be assigned to the term ‘\( b \)’ that encloses offsets, time dependent components and noise. ‘\( b \)’ is assumed to be independent from the other variables. And ‘\( a \)’ is a scalar gain between the Hamming Distance and \( W \) the consumed power.

Although the Hamming weight and Hamming Distance power models are the most well-known, improved versions of these models are proposed e.g., [36], where distinction is made between the ‘0’ to ‘1’ transitions and ‘1’ to ‘0’ transitions in the Hamming Distance model.

2.3.4 Power Analysis Attack Methodologies

There are several methodologies to perform a Power Analysis attacks.

- Simple Power Analysis (SPA) Attack
- Differential Power Analysis (DPA) Attack
- Correlation Power Analysis (CPA) Attack

2.3.4.1 Simple Power Analysis

Simple Power Analysis (SPA) attacks require only a few number of traces and are mainly performed along the time axis. It does not imply that this attack is simple, maybe even on the contrary; the attacker tries to derive the key with only one or a few traces.
There are different types of SPA attacks, like visual inspection of power traces or template based SPA attacks. [19] gives an example of an SPA attack on a DES encryption in hardware.

Attacks based on visual inspection only have a chance of success if the sequence of instructions depends on the key. An application of these kinds of attacks could be a micro controller implementation of the AES cipher, where round functions are implemented with instructions of the microprocessor, such as add, xor move and branch, memory read or memory writes. When a certain instruction only occurs when a key bit is 1, and another instruction only occurs when that bit is 0, then it is possible to derive the secret key.

Template attacks consider that the power consumption also depends on the data that is being processed. The power traces are here characterized by a multivariate distribution. This is fully defined by a mean vector and a covariance matrix \((m,C)\) A template attack basically characterizes the device under attack, e.g. certain instructions or sequences of instructions can be put in a template.

One example is as follows: one has two the same devices, where one device is the device under attack, and the other device is a device that one can fully control. On the one device where we have full control, sequences of instructions with different data and keys are executed and the power consumption is recorded. The data and the key pairs are grouped, and one can estimate the mean vector \((m)\) and the covariance matrix \((C)\) of the multivariate normal distribution. This results in a template for every pair of data and key. This characterization, together with the power traces from the device under attack, can be used to determine the key. This means that one evaluates the probability density function of the multivariate normal distribution with \((m,C)\) and the power trace of the device under attack. For every template one gets several probabilities. The probability tells how well a template fits in a given trace. The highest probability will indicate the correct template. Since each template is associated with a key, one will get an indication what the correct key is.

### 2.3.4.2 Differential Power Attacks (DPA) and Correlation Power Attacks (CPA)

A typical DPA or CPA attack has the following strategy:

- Choose an intermediate result of the encryption algorithm.
- Measure the power consumption.
- Calculate hypothetical intermediate values.
- Map intermediate values to power consumption values.
- Compare the hypothetical power consumption values with power traces.

These steps are illustrated in figure 2.22

A DPA or CPA attack starts with the gathering of power traces. For each encryption or decryption cycle, power traces are recorded using an oscilloscope and the known input data (plain text) is stored. Usually the plain text data is chosen to be random data, using a random number generator function.
After recording the power traces, hypothetical intermediate values are calculated using the known input data and all possible key values. These hypothetical intermediate values are then mapped on a power model such as the Hamming Distance power model or the Hamming Weight power model, see section 2.3.3. This mapping results in a hypothetical power consumption, that will be statistically compared with the sample data of the measured power trace, and results in a correlation. The result lists the correlation of every key hypothesis with the sample data. For each key byte or bit of the key byte, the highest correlation value is the best estimated key value.

The secret key is determined by repeating this process for every key byte or bit of a key byte, depending on the type of attack.
2.3.4.3 DPA Attack - Difference of Means

DPA attacks can be performed by analyzing the intermediate values of one bit (mono-bit DPA) or a set of several bits (multi-bit DPA). They do not require detailed knowledge about the attacked device and generally require a lot of power traces in order to analyze the power consumption at a fixed moment of time as a function of the processed data. One can distinguish first order and second order DPA attacks. A first order DPA attack is being performed at one instance of time. A higher order DPA attack is observed at more instances of time. Higher order DPA attacks are usually performed on masked encryption implementations (see section 2.3.6 to read more about masking).

Differential Power Attacks (DPA) is proposed by Paul Kocher[19] and evaluates the difference of means of power traces. The difference of means attack approach determines the relationship between the H and the T columns (see figure 2.22) using a binary matrix H that assumes that manipulating a bit to a logical '1' is different in terms of power consumption than manipulating a bit to a logical '0'.

The sequence of zero’s and one’s in matrix H (see figure 2.22) is a function of the input data \(d\) and the key hypothesis \(k_i\). To verify if a key hypothesis is correct, matrix T can be split in two sets of rows, i.e. two sets of power traces according to \(h_i\). The first set contains the rows of T whose indices correspond to the indices of zeros in vector \(h_i\). The second set contains all rows of T whose indices correspond to the indices of ones in vector \(h_i\). [26].

\[
m_{1i,j} = \frac{1}{n_{1i}} \sum_{l=1}^{n} h_{l,i} t_{l,j} \tag{2.11}
\]

\[
m_{0i,j} = \frac{1}{n_{0i}} \sum_{l=1}^{n} (1 - h_{l,i}) t_{l,j} \tag{2.12}
\]

\[
n_{1i} = \sum_{l=1}^{n} h_{l,i} \tag{2.13}
\]

\[
n_{0i} = \sum_{l=1}^{n} (1 - h_{l,i}) \tag{2.14}
\]

\[
R = M_1 - M_0 \tag{2.15}
\]

\(R\) corresponds to the difference between the mean vector \(m'_{0i}\) and \(m'_{1i}\). The closer the difference approaches zero, the less dependency between matrix H and T. The larger the difference is between \(m'_{0i}\) and \(m'_{1i}\), the larger the dependency between matrix H and T is. The highest peak in the trace leads to the best estimation of the key bit (or key byte in case of multi-bit DPA).

In the equations 2.11 till 2.15, 'n' is the number of power traces that is used for the attack, i.e. the number of rows of H.

This procedure is repeated for all possible key values in order to estimate the key byte value. The equations in the following section describe this in more detail.
Converting Hamming Weight Power Model to Binary Model  A Hamming Weight power model can not be used directly since the difference of means attack needs a binary power model. When the Hamming Weight model is used, (in this thesis project it is used in all DPA attacks) this Hamming Weight model needs to be reduced to a binary model. This is done by using the following selection function:

\[ h_{i,j} = 1 \iff HW(v_{i,j}) \geq 4 \]  

\[ h_{i,j} = 0 \iff HW(v_{i,j}) < 4 \]  

The binary power model, even with a Hamming Weight model converted to a binary model, is less accurate than a Hamming Weight model. 

Note that the Hamming Distance model can be used as well to be converted to a binary model (see equation 2.9. However this conversion to a binary model makes DPA attacks based on difference of means less effective than attack methods that are fully based on the Hamming Distance model, for example correlation based power attacks (CPA).

2.3.4.4 Correlation Power Attacks (CPA) - Pearson Correlation

The Correlation Power Attack (CPA) has been proposed by Brier [2]. Correlation between two variables express the degree to which the two variables are related. The CPA attack uses a Pearson Correlation, also known as Pearson product-moment correlation coefficient (PMCC) to determine the correlation between the measured samples and the Hamming Distance modeled power consumption. The Pearson Correlation assumes that the two random variables have a linear relationship. A correlation of ±1 is a perfect linear relationship between the two variables. A correlation of 0 means that there is no linear relationship between the two variables.

The CPA attack in this thesis project uses the Pearson correlation coefficient. The correlation coefficient is the covariance between two random variables, normalized by the product of their standard deviations see formula 2.18.

\[ \rho(X,Y) = \frac{cov(X,Y)}{\sqrt{var(X).var(Y)}} \]  

In formula 2.18 the covariance and the variance can be decomposed into equation 2.20 and equation 2.19 respectively.

In equation 2.19 var is the variance. The variance is determined as follows:

\[ var(x) = \frac{1}{n-1} \sum (x_i - \bar{x})^2 \]  

Where \( \bar{x} \) is the mean of x and n is the size of x. Obviously formula 2.19 applies for \( var(Y) \) as well. The square root of the variance is equal to the standard deviation.

Equation 2.20 shows the formula to calculate the covariance (cov).

\[ cov(x,y) = \frac{1}{n-1} \sum (x_i - \bar{x}).(y_i - \bar{y}) \]
When observing equation 2.20 and equation 2.21, the term \((X_i - \overline{X})(Y_i - \overline{Y})\) determines if a correlation value is positive or negative. The Pearson Correlation coefficient can be positive if and only if \(X_i\) and \(Y_i\) are on the same side of their means, (e.g. minus times minus is positive and positive times positive is positive). The correlation coefficient is negative when \(X_i\) and \(Y_i\) are on opposite sides of their corresponding mean values. This explains the positive and negative peak values in the correlation plots.

When working with \(N\) measurements (traces) of \(X\) and \(Y\), written as \(x_i\) and \(y_i\) where \(i = 1, 2, \ldots, N\), the 'sample correlation coefficient' equation can be used to calculate the Pearson Correlation coefficient. Equation 2.21 shows the sample correlation coefficient equation. For \(X\) and \(Y\) the random variables \(T\) and \(H\) are filled in, where \(T\) corresponds to the power traces and \(H\) to the hypothetical power consumption.

\[
r_{i,j} = \frac{\sum_{d=1}^{D}(h_{d,i} - \overline{h_i})(t_{d,j} - \overline{t_j})}{\sqrt{\sum_{d=1}^{D}(h_{d,i} - \overline{h_i})^2 \sum_{d=1}^{D}(t_{d,j} - \overline{t_j})^2}}
\]  

(2.21)

When observing figure 2.22, the CPA attack uses the correlation coefficient to determine the linear relation between the columns \(h_i\) and \(t_j\) for \(i = 1, \ldots, K\) and \(j = 1, \ldots, T\) for matrix \(H\) and \(T\), respectively. The resulting correlation coefficients are placed in matrix \(R\). Each value \(r_{i,j}\) is estimated based on \(D\) elements of columns \(h_i\) and \(t_j\).

2.3.5 Statistical Characteristics

This subsection briefly discusses the statistical characteristics of Power Analysis attacks and discusses some basic principles which are later used in the analysis of Power Analysis attacks.

Power Analysis attacks consider the fact that the power consumption is both data dependent and operation dependent. Besides these two components, also electronic noise is present and a constant leakage component. The constant leakage component can be, for example, the switching of transistors that occurs independently of the operation or the data value.

Every point can be modeled as a sum of the above mentioned components

\[
P_{\text{total}} = P_{\text{op}} + P_{\text{data}} + P_{\text{el.noise}} + P_{\text{const}}
\]  

(2.22)

All of these four components are functions of time and for each point in time these components could be different. For Power Analysis attacks, \(P_{\text{op}}, P_{\text{data}}\) and \(P_{\text{el.noise}}\) are the most important components.

The following subsections discuss statistical concepts that are used later in the attack analysis in chapter 5.

2.3.5.1 Sampling Distribution

This section will briefly discuss the sampling distribution.

The sampling distribution is the distribution resulting from the collection of data. The sampling distribution determines how well a certain parameter can be estimated.
The sampling distribution of Pearson Correlation coefficients (CPA attack) is best described using a Fisher’s Z Transformation. Using Fishers transformation, the correlation coefficient \( r \) from matrix \( R \) is transformed to a variable \( z \), see formula 2.23:

\[
z = \frac{1}{2} \ln \frac{1 + r}{1 - r}
\]  

(2.23)

The variable \( z \) has variance of \( \sigma^2 = \frac{1}{n-3} \), where \( n \) is the number of traces.

### 2.3.5.2 Confidence Intervals for \( \rho \)

Confidence intervals quantify how close an approximation is to the real parameter. An important concept related with confidence intervals are quantiles. Quantiles are necessary for interval estimation.

The quantile \( z_{\alpha/2} \) marks the point where the area (see figure 2.23) that is equal to \( \alpha/2 \), starts. The following definition holds:

\[
p(z_{\alpha/2} \leq Z \leq z_{1-\alpha/2}) = 1 - \alpha.
\]

Here \( \alpha \) is also known as the error probability.

A quantile can be determined by taking the inverse of the standard normal distribution function to \( \alpha \).

For example, figure 2.23 shows the confidence intervals for a correlation coefficient of \( \rho = 0.82 \) using 10000 traces and \( \alpha = 0.01 \). Since the sample distribution for correlation coefficient \( \rho \) is best described by transforming \( r \) by fisher transformation to \( z \) as mentioned in formula 2.23, \( z = 1.1568 \) and \( \sigma \) is calculated by \( \sigma_\rho = \sqrt{\frac{1}{n-3}} = 0.01 \).

The interval \([z_{\alpha/2}, z_{1-\alpha/2}]\) is the confidence interval for \( Z \). The confidence interval for the correlation coefficient \( \rho \) is given by formula 2.24

\[
\left[\frac{1}{2} \ln \frac{1 + \rho}{1 - \rho} - \frac{z_{1-\alpha/2}}{\sqrt{n-3}}, \frac{1}{2} \ln \frac{1 + \rho}{1 - \rho} + \frac{z_{1-\alpha/2}}{\sqrt{n-3}}\right]
\]  

(2.24)

In this example, the confidence interval is \([1.1311, 1.1826]\). The quantile \( z_{1-\alpha/2} = 2.5758 \) is the blue marked area on the right in figure 2.23.

A hypothesis test is used to test whether a certain hypothesis is correct or not. There are two hypothesizes \( H_0 : \mu = \mu_0 \) or \( H_1 : \mu \neq \mu_0 \). The first one is also known as the null-hypothesis. The hypothesis test accepts all the \( \mu_0 \) of the null hypothesis that are in the confidence interval of \( \mu \). Since \( \mu \) is not known, it is estimated. And because \( \mu \) is estimated, there is always a chance of an error. To minimize the error, a constant \( c \) is chosen such that the error of making a wrong decision is small, i.e. \( p(|\bar{X} - \mu_0| > c) = \alpha \). If \( c \) is chosen such that \( c = \sigma . z_{1-\alpha/2}/\sqrt{n} \) than the probability to decide wrongly against \( H_0 \) is \( \alpha \).

In the case of a hypothesis test of the correlation coefficient \( \rho \), \( \mu \) is described by using Fisher transformation \( \mu = \frac{1}{2} \ln \frac{1 + \rho}{1 - \rho} \). To test \( H_0 : \mu = \mu_0 \) against \( H_1 : \mu \neq \mu_0 \) can be done by checking if \( \mu_0 = \frac{1}{2} \ln \frac{1 + \rho_0}{1 - \rho_0} \) is in the confidence interval.

In order to determine a lower bound of the number of traces needed to estimate the mean of a normal distribution \( \mathcal{N} \left( \frac{1}{2} \ln \frac{1 + \rho}{1 - \rho}, \frac{1}{\sqrt{n-3}} \right) \) with a confidence level of \( 1 - \alpha \) and
precision \(c\), formula 2.25 from [26] can be used:

\[
n = 3 + 4. \frac{z_{1-\alpha/2}^2}{\ln^2 \frac{1+\varepsilon}{1-c}}
\]

(2.25)

In the given example of figure 2.23, the lower bound of the number of traces is equal to 66348 traces.

2.3.5.3 Confidence Interval for \(\rho_0 - \rho_1\)

In order to test if two correlation coefficients are different, the following Fisher transformation formulas are used:

\[
Z_0 = \frac{1}{2} \ln \frac{1 + r_0}{1 - r_0}
\]

(2.26)

\[
Z_1 = \frac{1}{2} \ln \frac{1 + r_1}{1 - r_1}
\]

(2.27)

First the correlation coefficients are transformed using Fisher’s transformation (see formula 2.26 and formula 2.27). The variance of both \(Z_0\) and \(Z_1\) is \(\sigma^2 = \frac{1}{n-3}\).

Then the confidence interval is given in formula 2.28

\[
\left[ \left( Z_0 - Z_1 \right) - z_{1-\alpha/2} \sqrt{\frac{2}{n-3}}, \left( Z_0 - Z_1 \right) + z_{1-\alpha/2} \sqrt{\frac{2}{n-3}} \right]
\]

(2.28)
In order to determine the number of traces of two different correlation coefficients, with a confidence of $1 - \alpha$ that the two distributions $Z_0 \sim \mathcal{N}\left(\frac{1}{2}ln\frac{1+\rho_0}{1-\rho_0}, \sqrt{\frac{1}{n-3}}\right)$ and $Z_1 \sim \mathcal{N}\left(\frac{1}{2}ln\frac{1+\rho_1}{1-\rho_1}, \sqrt{\frac{1}{n-3}}\right)$ are different, formula 2.29 can be used.

$$n = 3 + 8 \frac{z_{1-\alpha}}{(ln\frac{1+\rho_0}{1-\rho_0} - ln\frac{1+\rho_1}{1-\rho_1})^2}$$  (2.29)

In practice, the number of traces that is needed to see a peak is mainly determined by the distance between the sampling distributions with $\rho = 0$ and $\rho = \rho_{\text{max}}$ [25]. Formula 2.29 is then simplified by formula 2.30.

$$n = 3 + 8 \frac{z_{1-\alpha}^2}{(ln\frac{1+\rho}{1-\rho})^2}$$  (2.30)

For example: when $\rho_0 = 0$ and $\rho_1 = 0.1$, and with the error probability $\alpha = 0.0001$ and using 10000 traces, using the Fisher transformation $\rho_0$ and $\rho_1$ become respectively $Z_0 = 0$ and $Z_1 = 0.1003$. Furthermore $\sigma = 0.0141$, then performing $\mathcal{N}^{-1}(\alpha = 0.0001, \mu = 0, \sigma = 1)$, $z_{1-\alpha/2}$ is calculated and becomes $z_{1-\alpha/2} = 3.8906$.

The confidence interval is calculated according to formula 2.28 and is equal to $[-0.1554, -0.0453]$. The number of estimated traces can be calculated using formula 2.30 and is equal to $n = 2751$ traces.

In [26] it is mentioned that in practice CPA attacks lead to correlation coefficients below 0.2. In Appendix section A.1.4 this is confirmed when, using simulated FPGA traces, the maximum correlation coefficient is just below 0.2.

Using formula 2.30, a plot is constructed in figure 2.24 showing the calculated correlation coefficient versus the number of traces for correlation coefficients between 0 and 0.2.

### 2.3.5.4 Quantitative Evaluation Power Analysis attacks

In [23] an quantitative evaluation of Power Analysis attacks is presented by analyzing the normalized difference. The normalized difference can be used to analyze statistical significance. The normalized difference can be calculated using formula 2.31

$$\Delta z = \frac{Z_0 - Z_1}{\sqrt{\frac{2}{n-3}}}$$  (2.31)

$Z_0$ and $Z_1$ are the Fisher transformed correlation coefficients by formula 2.26 and 2.27. When $\Delta z$ is larger then the critical value (i.e. outside the confidence interval) then the difference is statistically significant. The larger the difference, the easier to detect the correct key guess and the less secure the cryptographic hardware is.

It is interesting to evaluate a certain correlation $\rho$ with the distribution where $\rho = 0$. Figure 2.25 shows the normalized difference versus the correlation coefficient ranging from 0 to 0.2.
In figure 2.25, a line is drawn at the x-axes at point -3.8906. This point is critical point $z_{1-\alpha/2}$ where $\alpha = 0.0001$.

For $n = 10000$ traces, the normalized difference intersects with the critical point at a correlation coefficient of 0.0246. This means that correlation coefficients that are lower
than 0.0246 are inside the confidence interval and are considered to be insignificant. Correlation coefficients higher than 0.0246 when using 10000 traces are considered to be significant and will most likely result in a peak.

Likewise, for \( n = 50000 \), the normalized difference intersects with the critical point at a correlation coefficient of 0.055. This means that when using 50000 traces, correlation coefficients lower than 0.055 are insignificant and correlation coefficients higher than 0.055 are out of the confidence interval and will most likely result in a peak.

### 2.3.6 Counter Measures

This section briefly discusses the possible countermeasures.

The goal of countermeasures is to make the power consumption independent of the intermediate data values of the cryptographic algorithm. While countermeasures do not guarantee that attacks will be prevented, it purposes to make the life of the attacker more difficult.

Countermeasures against Power Analysis attacks can be divided into two groups:

1. Protocol countermeasures
2. Implementation countermeasures

#### 2.3.6.1 Protocol Countermeasures

Protocol countermeasures keep the implementation intact, but the cryptographic protocol is changed in such a way that an attacker is not able to retrieve enough leakage. This can be done, for example, by frequent key updates or other protocol related changes.

#### 2.3.6.2 Implementation Countermeasures

Implementation countermeasures leave the protocol unchanged but focus on modifications of the implementation. Some of the most well-known implementation countermeasures are masking and hiding.

**Masking** Masking is an countermeasure that randomizes the intermediate values that are processed by a cryptographic device. The advantage of masking is that it makes the power consumption independent at an algorithmic level without changing the power consumption characteristics of the cryptographic device.

Using the masking countermeasure, each intermediate value \( v \) is concealed with an random value \( m \), called the mask. The mask can be applied by logically XORing the mask with the intermediate value and later apply another XOR operation with the same mask. A logical XOR is also known as modular addition, i.e. \( v_m = v + m \text{mod}(n) \), where \( n \) is defined by the cryptographic algorithm.

Another masking technique is modular multiplication, i.e. \( v_m = v \times m \text{mod}(n) \), ([13] discusses multiplicative masking). There are several proposed masking schemes, [32] gives a good example of a masked AES Sbox.

However, masked implementations can be attacked using higher order DPA or CPA attacks [37], [27], [28].
Hiding  With hiding, one can bury the power consumption that is produced by the cryptographic circuit in noise, or one can hide information by processing the sensitive data at random time intervals. Most hiding countermeasures are based on logic that complements or reduces bit flips.

Another type of hiding is to have some dummy logic running that creates extra noise. In [1], even an extra processor is used to exactly balance the bit flips of the processor that executes the actual AES algorithm. This countermeasure can be easily beaten when using a EM probe that only scans the chip area of the non-balancing processor that executes the encryption algorithm.

The goal of the hiding countermeasure is to lower the SNR of the side channel signal. The lower the SNR, the lower the correlation between the power consumption of the device and the hypothetical power consumption of the device [25]. Hiding is the type of countermeasure that is used in this research. Section 4.4.2 discusses extensively the SNR of the CPA attacks performed in this research.

In Appendix A.2, related work is done to subtract a pure noise trace from the power trace. In this case the same hardware design was used but the cryptographic core is replaced by dummy logic that simply wires the input connection to the output. The results showed that subtracting a pure noise trace effects samples containing valuable side channel information. After subtracting, valuable side channel information got lost and the design was even harder to attack.

**Costs of Countermeasures**  Countermeasures come with a cost. Although security is an important factor, it also has to be feasible in terms of power, performance and area. A quote of Stephan Mangard, one of the writers of the DPAbok[26] during a conference in Leiden, The Netherlands: "The challenge is not to build a secure system, but to build a secure system with acceptable cost." Where cost can be defined in power, performance and area. In this research, in section 2.2.5, it can be seen that some countermeasures are in fact too costly the device does not allow certain countermeasure implementations (e.g. dual-rail logic that requires too much area.)

### 2.3.7 Side Channel Analysis Tool

Riscure B.V., in Delft, the Netherlands, developed a Side Channel Analysis tool to perform Side Channel Attacks, called Inspector[3]. Inspector supports various cryptographic cryptanalysis algorithms, various data acquisition modules, filtering tools editing, statistical analysis and more. Inspector is a Java-based platform that can be extended with customized user modules. Figure 2.26 shows a screen shot from Inspector.

In this thesis project, Inspector is used to gather all the data related to Side Channel Attacks.

#### 2.3.7.1 Related Work: Verilog PLI based Simulation Model

Appendix A.1 discusses a simulation-based power modeling method that is based on Post place and route simulation. It is a first attempt to show that it is possible to use Post place and route simulation data to acquire Power traces that are based on
FPGA simulations. Future work is needed to make the model more perfect and perform optimizations in terms of file size of the traces and the speed of acquiring the simulation traces.

Currently there is no solution yet available to implement Side Channel Attacks in the early stages of the hardware design process. At the time of writing, hardware designers first need to design and implement the hardware, upload the design to an actual FPGA, and then collect the power traces by measuring the voltage or current of the FPGA device multiple times. The actual Side Channel Attack can only be performed in the final stages of the hardware design. This work would enable the designer to acquire power traces by simulation. An advantage of simulations is that they are not exposed to measurement noise or other environmental noises. One can acquire traces of complete designs, of specific parts of the design, or one can zoom in to specific modules or IP cores and acquire power traces. Once the traces are acquired, tools like Riscure’s Inspector or MATLAB, can be brought into the design process and used to perform attacks on these simulated traces.
This chapter discusses in more detail the hardware designs and the submodules of the cryptographic core that are used in this thesis as well as the used hardware platform. Section 3.1 discusses the used hardware platform Sasebo-GII Side Channel Attack Evaluation board, and section 3.2 discusses the used AES cores and their hardware implementation.

3.1 Sasebo-GII Side Channel Attack Platform

This section briefly introduces the Sasebo-GII Side Channel Attack platform that was generously made available for this research project by Riscure B.V., Delft, The Netherlands. Section 3.1.1 gives an introduction of the Sasebo-GII board. Section 3.1.2 discusses the Virtex5 Cryptographic FPGA and section 3.1.3 discusses the AES component, which is the component where both the synchronous and the asynchronous logic AES cores are implemented.

3.1.1 Introduction to Sasebo-GII Side Channel Attack Evaluation Board

As a research project, the national institute of Advanced Industrial Science and Technology (AIST) of Japan and Tohoku University developed a Side-channel Attack Standard Evaluation BOard (SASEBO)[17] funded by the Ministry of Economy Trade and Industry (METI) of Japan. At the moment of writing, there are five different Sasebo boards available. The Sasebo board used in this thesis is the Sasebo-GII board. Figure 3.1 shows the schematic of the Sasebo-GII board.

The Sasebo-GII board consists of two FPGAs. The Xilinx Spartan-3A FPGA is the control FPGA, and it handles the USB communication by interfacing with FTDI USB controller chip FT2232D. The Spartan 3A FPGA transforms the USB data to a local bus interface. Using this bus interface, the Spartan 3A FPGA communicates with the Virtex5 FPGA. The actual Cryptographic core runs on the Virtex 5 FPGA, see figure 3.1.

By placing a resistor between the power line of the Virtex 5 FPGA, the voltage over that resistor is measured with an oscilloscope. This is the starting point to collect power traces.

3.1.2 Cryptographic FPGA

This section discusses the architecture of the Virtex5 Cryptographic FPGA.

Figure 3.2 displays the top level design of the Cryptographic FPGA on the Sasebo-GII board. HDL code of a the complete FPGA logic design, including a synchronous
AES core came along with the Sasebo-GII board and is designed by Akashi Satoh of AIST and Tohoku University [39].

The actual cryptographic core is the AES Comp module. Around the AES Comp module is a local bus interface that communicates with the local bus controller. The local bus controller communicates with a read FIFO a write FIFO that are connected...
to the actual local bus. The local bus controller and the two FIFOs are the building blocks that enable communication between the two FPGAs.

Figure 3.3 shows the post place and route simulation of the AES Comp module with the signals on the trigger pins.

![Figure 3.3: Post Place and Route simulation AES Comp with trigger pins signals](image)

The write FIFO (FIFO W) receives the key and the plain text data in chunks of bytes. The local bus controller makes two byte packages of it and sends it to the local bus interface (LBUS I/F). The LBUS I/F sends the data to the AES Comp in a 128 bit data word. When the bytes are loaded it raises the ready signal (for the plain text data DRDY and for the key KRDY). When both are ready, the AES encryption core starts encrypting the plain text. When the plain text is encrypted, the DVLD signal is raised and the LBUS I/F sends the cipher text in 2 byte packages to the local bus controller. The local bus controller sends the cipher text to the read FIFO (FIFO R). Figure 3.3 shows these signals.

One of the three trigger pins are used by the oscilloscope to trigger the AES encryption. Trigger pin 'trig startn' is an active low signal indicating the start of the encryption operation. Trigger pin 'trig endn' is an active low signal indicating the end of the encryption operation. Trigger pin 'trig exec' is an active high signal indicating the total time of the executing of the AES encryption cipher. As can be seen an encryption operation takes ten clock cycles. The trigger signals are routed to the header pin on the Sasebo-GII board.

The different AES cipher designs are placed in the AES Comp module.

3.1.3 AES Component

This section discusses the synchronous AES component.

The AES Component (AES Comp) module is modified and a VHDL description of the AES cipher (noted as AES CORE in figure 3.4) module is embedded in the existing code.

---

1In figure 3.3 the valid and ready signals from the key (KRDY and KVLD) didn’t fit on the wave window, they were asserted earlier in the wave view.)
Figure 3.4 shows an abstract toplevel view of the synchronous AES component.

As can be seen in figure 3.4 the AES component consist of several modules. The Rrg is a round counter that shifts a logical ‘1’ on each clock cycle. This Rrg signal is used to select the muxes and select the mux in the AES core.

The first round key is generated in the first round as follows; The secret key is loaded into register KrgX. Then the key is presented to the AES core and the the first round key is generated and appears at output Ko of the AES core. The Ko output is connected to the KrgMux. The KrgMux is controlled by a bit of the Rrg signal. The first round key is stored in register Krg. The output of the Krg register is connected to the key input of the AES CORE. In the first round, the key, which is stored in the KrgX register, is logically XORed with the plain text input Di. The result goes to the DrgMux which is controlled by a bit of the Rrg signal. The output of the DrgMux goes to the Drg register. The Drg register is connected to the data input of the AES core and to the output Dout.

In the remaining rounds, the DrgMux is set to select input ‘1’ and the KrgMux is set to select input ‘1’. On each clock cycle a new encryption round is processed and the bit in the Rrg register is shifted with one position.

After 10 rounds the AES component is done executing, and the cipher text remains in register Drg until a new encryption is started.

\[^2\text{Not all components and signals are mentioned in the schematic of figure 3.4, including the valid and ready signals of the key and data signals.}^2\]
3.2 AES Cores

This section discusses the implemented AES cores used in this thesis.

Section 3.1.3 discusses the synchronous AES core and section 3.2.3 discusses the asynchronous PetriNet-based AES core.

3.2.1 Synchronous AES Core

This section discusses the synchronous AES core and the hardware design of the subcomponents.

Figure 3.5 shows a top level view of the synchronous AES core.

Figure 3.5: Schematic top level view of the synchronous AES core

Figure 3.5 gives a more detailed overview of the AES CORE module from figure 3.4. The AES round operations can be easily detected in this figure. The Rrg signal is used to keep track of the round number and controls multiplexer. The multiplexer makes sure that when Rrg bit 0 is logical ’1’, i.e. the last round, that the Mix Column operation is excluded from the round operation (see section 2.1.1). The RCON module is a lookup table that selects the correct round constant that is used as input for the key expansion module. The key expansion module calculates the new round key and presents it at output Kout. The new round state is presented at output Dout. On each clock cycle a new round key and round state is being calculated until all the ten AES rounds are being performed.

3.2.2 Synchronous AES Core Components

This section discusses the implementation of various subcomponents of the synchronous AES core.
3.2.2.1 Sub Bytes

The Sub Bytes operation performs 16 non-linear transformations, known as Sbox operations. There are several possible hardware implementations for Sboxes proposed. For this thesis project the design of [51] by Johannes Wolkerstorfer, Elisabeth Oswald and Mario Lamberger is chosen.

The Sbox consist of two components: the multiplicative inverse operation and the affine operation. Wolkerstorfer, Oswald and Lamberger implemented the multiplicative inverse operation efficiently by implementing it in logic that works in $GF(2^4)$ instead of in $GF(2^8)$. This is area efficient and uses less resources than an $GF(2^8)$ implementation.

Since the logic works in another finite field, it is necessary to convert it from $GF(2^8)$ to $GF(2^4)$ and visa versa. This is done by the map and $map^{-1}$ function. This Sbox implementation is also used in [26].

Figure 3.6 shows a schematic overview of this Sbox.

Between the map functions, the Sbox consist of two square operations, several multipliers and adders, and an inverse operation. These functional blocks all work in $GF(2^4)$. Most of these functional blocks consist of XOR operations and shifts. See section 2.1.2 to read more about how adders and multipliers are constructed in GF arithmetic.

After the inverse map an affine transformation is applied. Please see [51] for more implementation details over this Sbox.

![Figure 3.6: Schematic view of the AES SBox](image)

3.2.2.2 Shift Row

In hardware this is simply the rerouting of connections from the sub byte output to the next module (Mix Column) input.

3.2.2.3 Mix Column

The mixed column operation is executed in each round except for the last round of the encryption.

The hardware implementation is a one to one translation of matrix equation 2.7 of section 2.1.3.3.
This equation performs its operation on each column of the state, i.e. four times a 'Mix Column operation' on 32 bits of the state. This equation is implemented according to figure 3.7. The 01 in the matrix of equation 2.7 requires no multiplication. The 02 is a multiplication by 2, and the 03 is a multiplication by 3.

\[
\begin{bmatrix}
b_0 \\
b_1 \\
b_2 \\
b_3 \\
\end{bmatrix} =
\begin{bmatrix}
m_2_0 \oplus m_3_1 \oplus a_2 \oplus a_3 \\
a_0 \oplus m_2_1 \oplus m_3_2 \oplus a_3 \\
a_0 \oplus a_1 \oplus m_2_2 \oplus m_3_3 \\
m_3_0 \oplus a_1 \oplus a_2 \oplus m_3_3 \\
\end{bmatrix}
\] (3.1)

Input a0 till a3 is positioned on the top from left to right in figure 3.7. Also the fixed multipliers 02 and 03 are listed from left to right, as well as output b.

The multiplication by two and three can be efficiently implemented in hardware. Multiplying with 2 is a bitwise shift to the left and the multiplication with 3 is a bitwise shift left of its original value with an XOR (plus '1'). (see section 2.1.2 for details about Galois Field arithmetic).

After both of multiplications are performed, the modulo of m(x) needs to be executed on the subset of the state. Note that the m(x) is changed from 0x11B to 0x1B. This is because any value greater than 255 (8 bits) will be XORed to zero since the 283-polynomial MSB is the 9th bit which will always be zero. Therefore this bit can be ignored.

### 3.2.2.4 Add-Round key

Since addition in finite Galois fields is equal to a bitwise XOR (exclusive OR) operation, the round key is XOR-ed with the state data. The round key itself is iteratively generated by the key expansion module.
### 3.2.2.5 Key Expansion

The key expansion module contains four Sboxes and consist of several XOR operations and is built according [10]. Figure 3.8 shows the key expansion module.

![Figure 3.8: KeyExpansion schematic](image)

The key expansion module uses the secret key as input and produces a set of round keys that are used in each round according to a schedule. In figure 3.8 the numbers represent the keybytes that are transformed in this operation. Four S-boxes are used and they transform bytes 12, 13, 14 and 15. On byte 13 the round constant (RCON) is added (XOR-ed). Each round has a different round constant. The round constants are respectively: 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36. Then the bytes are grouped in words of 4 bytes and permutated and transformed by addition (XOR-ed) according to figure 3.8.
3.2.3 Asynchronous AES core

This section discusses the top level design of asynchronous AES core that is implemented using PetriNet controllers.

Figure 3.9 shows the asynchronous AES CORE that is built with PetriNet Controllers.

![Figure 3.9: Schematic view of AES CORE build with PetriNet controllers](image)

As can be seen in figure 3.9, the asynchronous core is similar in layout as the synchronous core of figure 3.5. In figure 3.9 the keypart and the data part are separated. Each part (data and key) have a separate Rrg latch, that keeps track of the current encryption round. The reason that the data part and the key part have separate Rrg latches, is that the first round key needs to be independently calculated and be ready for the data part when the Add Round Key operation is being performed.

Figure 3.9 shows the PetriNet controllers as square gray blocks. The 'request in' and 'acknowledge out' input and output are positioned on the left side of this block, and on the right side the request output and acknowledge input are located. The request signals are marked in blue and the acknowledge signals are marked in red.

The square gray blocks can be both simple latch controllers or a top level of a subcomponent of the AES cipher.

Muller-C elements are used when two or more handshake signals need to be both in an active state, i.e. merged or synchronized.

Section 3.2.4 will discuss each of the subcomponents of figure 3.9 in more detail.

3.2.4 Asynchronous AES Core Components

This section discusses the AES core components as well as important asynchronous components.
3.2.4.1 Port Controllers

In order to connect the asynchronous cipher with the synchronous part of the design, port controllers are used. Two types of controllers are necessary; one type for the inputs of the AES core, also called active port controllers, and one type for the outputs of the AES core passive port controllers. Figure 3.10 shows how the port controllers are used to interface the asynchronous logic.

![Passive and active port controllers AES core](image)

Figure 3.10: Passive and active port controllers AES core

The active port controller enables communication by asserting the request signal, which occurs when the enable is pulled high and ack is low. The enable signals are only asserted when new data is available (new plain text, or new round key).

The passive port controller is enabled every clock cycle. Only when a request signal is asserted, the input an acknowledge will be set high. The AES core will detect the acknowledge and will bring the request out signal low. Then the passive port controller will consequently set the acknowledge output low again.

Both controllers have a reset input that will bring the outputs low and all Muller-C elements have reset inputs as well.

In [5] and [52] some proposals for port controllers are made. A modified version that suits the interface to the asynchronous CORE is displayed in figure 3.11

![Port controllers](image)

Figure 3.11: Port controllers
3.2.4.2 Matched Delay

The matched delay is constructed of a series of AND gates. Figure 3.12 shows the matched delay component using a Synplify Pro Technology view.

![Figure 3.12: Matched Delay component in Synplify Pro Technology view](image)

This delay implementation has a quick fall down time due to a shared signal on the input of each AND gate. The matched delay is implemented using the RPM (relatively placed macro) RLOC, adapted from [22].

Figure 3.13 shows how the RLOC RPMs make sure the delays are in a fixed positioned in the FPGA design. In figure 3.13 the delay element construction starts with the RLOC of the AND gate that is implemented in LUT 0 and the next LUT is in the CLB above it, this pattern repeats itself as long as the matched delay is.

The number of AND gates that forms the matched delay is empirically determined for each logical function block. Figure 3.14 shows an example of the matched delay over the multiplicator M0 of Sbox0 in the Sub Bytes component. As can be seen the matched delay is long enough to prevent the glitches to appear at output q.
3.2.4.3 PetriNet Controller

As mentioned in section 2.2.4.4, in order to avoid hazards, a controller could be implemented in a single LUT. In [15] was shown how to implement a Muller-C element in a LUT. The controller mentioned in figure 2.13 in section 2.2.3.2 can be implemented using a modified Muller-C element.

Figure 3.15 shows how the PetriNet controller can be implemented in one LUT.

After post synthesis, the two buffers connected to signal ‘ack in’ and ‘req out’ in figure 3.15 will be removed and the two signals ‘ack in’ and ‘req out’ will be merged as one signal. Using the approach of [15] the initialization string of the LUT becomes x’00B2’ in order to implement logic function:

\[ LO = \overline{I_0}.LO + \overline{I_1}.LO + \overline{I_1}.I_0 \]

3.2.4.4 Mux and DMUX

This section discusses briefly the multiplexer (Mux) and the demultiplexer (Dmux) of the asynchronous core.

Figure 3.16 shows the mux. The mux and dmux are used to switch in the last round where there is no Mix Column operation.
As can be seen in figure 3.16 the mux is implemented using Muller-C elements a mux and inverters. For both the request out as the acknowledge input, matched delays are used (see section 3.2.4.2 to read more about matched delays). Since the mux is a area costly component, the signal glitches before the data becomes valid, therefore the matched delays are important to bridge this time period of glitching.

Figure 3.16 shows the demultiplexer.

The demultiplexer is implemented with latch controller and two output controllers depending what output is selected. The $ctl_i$ input is connected to the Rrg register of the data part (not the key part). The Rrg register value shifts one bit in a 11 bits vector each round. The data signal from the latch output is simply forked. Only one of the two sides will get a handshake request and will pass the data through. The demultiplexer is implemented on a LUT level with RLOC RPMs. This is done to make sure that the LUTs are closely positioned to one another to avoid unnecessary delays of signals and prevent as best as possible instable behavior.

### 3.2.4.5 Sub Bytes

The Sub Bytes operation contains 16 Sboxes. The request and acknowledge signals of all Sboxes are synchronized towards single request and acknowledge signals using Muller-C elements. The design of the Sbox is straight forward and very similar to the
synchronous Sbox implementation of figure 3.6.

Figure 3.18: Sbox schematic of PetriNet controllers

Figure 3.18 shows different components of each Sbox. Each component uses corresponding Galois Field arithmetic functions. These functions are at the lowest level matched with a matched delays. Muller-C elements are used to synchronize the operations.
3.2.4.6 Mix Column

Figure 3.19 shows the Mix Column module performing the function of formula 2.7 in section 2.1.3.3 on each 32 bit column of the AES state data.

Figure 3.19: Schematic of MixColumn operation using PetriNet controllers

The Mix Column module consist of four fixed multiplication by two modules (gfmul2) and four fixed multiplication by three modules (gfmul3). When all multiplications are done (this is synchronized by the Muller-C elements) the Mix Column function is executed. The Mix Column function is implemented as described in section 3.2.2 with PetriNet control latches on the output and a matched delay over the Mix Column function. In figure 3.19 this is module ‘Mixcolumn cl hs’.

After the delay, when the input data is rippled through the Mix Column function, the last controller controls four output latches for each byte of the column.

Four of these 32 bit Mix Column operations are used for the Mix Column operation and all four modules are synchronized with each other using Muller-C elements.

3.2.4.7 Key Expansion

The KeyExpansion module has four Sboxes and four modules $w_0$ till $w_3$ performing combinations of logical XOR operations. Figure 3.20 shows the Key Expansion module.

The Muller-C elements synchronize the four Sbox operations. After the four Sboxes have their data ready, the round key state will be processed by the first stage ($w_0$). After $w_0$ has its data ready, data will be processed by the next stage, etc. till the new round key is stored in a PetriNet controlled latch ffout.
Stage $w_1$ till $w_3$ consist of a simply 4 bytes Galois Field addition i.e. an XOR operation over 4 bytes. Over this operation a matched delay is placed and a PetriNet controller that controls an output latch. Stage $w_0$ is slightly different implemented since it has an extra XOR operation for the round key byte addition (RCON). Over these two XOR operations a matched delay is placed and, similar to the $w_1$ till $w_3$ stages, this stage has an output latch controller that stores the output of this stage.
This chapter discusses the Power Analysis attacks on both the synchronous AES cipher and the asynchronous AES cipher.

Section 4.1 gives an introduction to the Power Analysis attacks. Section 4.2 discusses the characteristics of the Power Analysis attacks on the synchronous AES cipher. Section 4.3 discusses the attacks performed on the asynchronous AES cipher. Finally section 4.4 compares the results of both attacks.

4.1 Introduction

This section gives an introduction about the performed Power Analysis attacks in this thesis project. Section 4.1.1 discusses the laboratory setup for the attacks. Two type of attacks are performed on each trace set: a CPA and a DPA attack. Section 4.1.2 discusses the CPA attack point, and section 4.1.3 discusses the DPA attack point.

Both attacks are performed using the same trace set at the same time positions. For more information about the background of these attacks see section 2.3.4.2

4.1.1 Setup

This section discusses the lab setup for the Power Analysis Attack. Both synchronous and asynchronous hardware designs are implemented on the Sasebo-GII board in the Virtex 5 Cryptographic FPGA, see section 3.1.2. Table 4.1 lists the main components and settings for the setup.

<table>
<thead>
<tr>
<th>Oscilloscope</th>
<th>PicoScope 5302</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sample frequency</td>
<td>1Gsample/sec</td>
</tr>
<tr>
<td>Probe</td>
<td>Riscure Inspector PowerTracer</td>
</tr>
<tr>
<td>FPGA</td>
<td>Xilinx Virtex 5 xc5vlx30-3ff324</td>
</tr>
<tr>
<td>Operating frequency</td>
<td>2 MHz (via local bus Spartan3 FPGA)</td>
</tr>
<tr>
<td>Trigger signal</td>
<td>header pin of trig endn signal</td>
</tr>
<tr>
<td>Measure point</td>
<td>Voltage drop over a 1 Ohm resistor, inserted in the VDD line of Virtex 5 FPGA</td>
</tr>
<tr>
<td>Secret Key</td>
<td>2B 7E 15 16 28 AE D2 A6 AB F7 15 88 09 CF 4F 3C</td>
</tr>
<tr>
<td>Last round key</td>
<td>D0 14 F9 A8 C9 EE 25 89 E1 3F 0C C8 B6 63 0C A6</td>
</tr>
</tbody>
</table>

Table 4.1: Setup

Using Riscure’s PowerTracer probe, the voltage drop is measured over a 1 Ohm resistor that is inserted in the VDD line of the Virtex 5 FPGA. The oscilloscope measures this voltage at a sample rate of 1 Gsample/sec.

Using Riscure’s Inspector Software, an acquisition software module makes sure that for each trace in the trace set, the generated (random) plain text data, the cipher text
data and the sampled measurement data is stored. The set of traces is stored in a special trace file (*.trs), see table A.1 in section A.1.2. In order to sample one full encryption (all rounds) for both hardware designs, 8000 samples are needed.

After this acquisition phase and prior to the actual attack, several signal processing steps are necessary. Section 4.2 discusses these signal processing steps in more detail.

4.1.2 CPA Attack Point

The CPA attack, as discussed in section 2.3.4.4, analyzes the correlation between power traces and the hypothetic Hamming distance power model using the Pearson Correlation formula.

As an attack point the round key input of the last encryption round is chosen, with the goal to recover the last intermediate round key $k_{10}$. The Hamming distance between the input of the Sub Bytes and the output of the Add Round Key operation is used. When the last round key is recovered, the secret key can be easily determined using the known behavior of the AES algorithm.

The V-matrix of figure 2.22 containing the hypothetical intermediate values is created according to equation 4.1 that describes the point of attack, i.e. the input of the last round operation:

$$v_{i,j} = S^{-1}(c_{10}(i) \oplus k_j) \quad (4.1)$$

The last round input (intermediate cipher text) in equation 4.1 is determined by reversing the last round operations by $S^{-1}(c_{10}(i) \oplus k_j)$. Note that the Shift Row operation is not present in this equation since in hardware this is just a rerouting of inputs to different outputs. This means that it doesn’t contribute to the power behavior in CMOS. For this reason the Shift Row operation can be ignored in the power model.

The V-matrix is a $D \times K$ sized matrix, where $D$ is the number of plain text inputs (same number as the number of traces, when using one different plain text for each trace) times all the key possibilities of a key byte, this is 256. After the creation of the V-matrix, the hypothetical intermediate values need to be mapped on a power model.

When using the Hamming distance power model, the Hamming distance between intermediate cipher text $C_9$ and $C_{10}$ is determined.

$$h_{i,j} = HW(v_{i,j} \oplus c_{10}(i)) \quad (4.2)$$

Equation 4.1 describes the Sbox input of the last round, which is equal to intermediate cipher text $C_9$. XOR-ing formula 4.1 with $C_{10}$ simply describes the Hamming distance model, see equation 4.3.

$$h_{i,j} = HD(v_{i,j}, c_{10}(i)) \quad (4.3)$$

Substituting all equations together the Hamming distance power model can be described as equation 4.4:

$$h_{i,j} = HW(S^{-1}(c_{10}(i) \oplus k_j) \oplus c_{10}(i)) \quad (4.4)$$

Formula 4.4 is the hypothetical power model that will be used to correlate a selection of the sample data of the power traces using formula 2.21. For each key byte, 256
estimations are made. The best estimation of a key byte is the one with the highest correlation. In order to estimate all 16 key bytes, a total of 4096 estimations are performed.

4.1.3 DPA Attack Point

The DPA attack, as discussed in section 2.3.4.3, analyzes the correlation between the power traces and the hypothetic power model using the difference of means formula. The DPA attack that is performed in this thesis uses the same attack point as the CPA attack (see section 4.1.2) and also the same power model described in equation 4.1 to 4.4. However, the DPA attack can only deal with a binary model. Therefore a selection function is used over the Hamming distance power model (see formula 2.16 and formula 2.17 in section 2.3.4.3) that transforms the Hamming distance power model into a binary model. This same selection function can be used for a Hamming Distance model.

The H matrix is constructed using a the selection function over the V -matrix as follows:

\[ h_{i,j} = 1 \iff HW(v_{i,j} \oplus c_{10}(i)) \geq 4 \] (4.5)

\[ h_{i,j} = 0 \iff HW(v_{i,j} \oplus c_{10}(i)) < 4 \] (4.6)

After the H matrix is constructed, the DPA traces are ordered into two groups (binary model) according to the Hamming distance, using the same selection function. The first set contains the rows of T whose indices correspond to the indices of the zero’s in the vector h. The second set contains all the remaining indices.

Then the mean over the rows in the first and second set are determined.

\[ m_{1i,j} = \frac{1}{\sum_{l=1}^{n} h_{l,i}} \cdot \sum_{l=1}^{n} h_{l,i} \cdot t_{l,j} \] (4.7)

\[ m_{0i,j} = \frac{1}{\sum_{l=1}^{n} (1 - h_{l,i})} \cdot \sum_{l=1}^{n} (1 - h_{l,i}) \cdot t_{l,j} \] (4.8)

The final result is the subtraction of the mean of the set of one’s minus the mean of the set of zero’s.

\[ R = M_{1} - M_{0} \] (4.9)

See equations section 2.3.4.3.

Since DPA uses a binary model, precision is lost and it can be seen in the rest of this chapter that DPA attacks perform significantly worse than CPA attacks. This is because matrix H (which describes the power model), consist only of values 0 or 1, as apposed to the power model of CPA that contains the full Hamming weight range.
4.2 Attacks on Synchronous AES

This section discusses the Power Analysis attacks on the synchronous AES cipher with traces measured from the Sasebo-GII board. Figure 4.1 shows the raw trace of the synchronous AES cipher.

As can be seen in figure 4.1 there are 11 peaks counted. These peaks correspond to the AES rounds executed in hardware. AES-128 however counts 10 rounds. As can be seen in figure 3.4 in section 3.1.3, the output of the design is connected to the output of register Drg and the output also branches to the AES CORE. Since the output of the last round is also directed to the AES CORE, the end result (cipher text) ripples through the logic of the AES CORE. This explains the '11th' round in the power trace of figure 4.1.

The peak voltage of the signal is close to 1V and -0.5V.

Both DPA and CPA attacks will estimate the intermediate key of the last AES round (round 10). Once this last round key is estimated, using the AES algorithm it is then quite easy to determine the secret AES key.

The last round of the synchronous AES design is displayed in figure 4.2. Before this round was attacked, the following signal processing steps were performed:

First the last AES round is selected, and the trace set was trimmed back to only the samples of interest, i.e. the samples of the last round operation of AES.
Secondly, all traces are aligned. The traces that are acquired by the oscilloscope are not accurately aligned with each other. Therefore aligning is an essential signal processing step before the cryptanalysis in order to gain more precision during the cryptanalysis. For the alignment operation, the static alignment module is used from Riscure’s Inspector software [3]. Static alignment correlates the selected part of a reference trace to parts in the other traces. The relative position, where the best correlation appears, is used as the shift value to align the trace with the reference trace. This is done for each trace in the trace.

After aligning, all traces are re-sampled in order to gain a smaller set of samples and improve the attack speed and to make sure the PC has enough memory available to perform the cryptanalysis. The re-sampling is performed from 1Gs/s to 100Ms/s, resulting in a trace set with each trace containing 66 samples.
4.2.1 CPA Attack

This section discusses the CPA (Correlation Power Analysis) attack results. Each key byte of the last round key $k_{10}$ is estimated. In this section, only the results of the first key byte (byte 1) and the last key byte (byte 16) will be discussed. Appendix B.1.1 lists the results of the other key bytes of the CPA attack of the synchronous AES design.

4.2.1.1 CPA key byte 1

Figure 4.3 shows the graphical representation of the CPA attack of key byte 1, having used a total of 10000 traces.

![Figure 4.3: Correlation estimation key byte 1 versus samples using CPA with 10000 traces](image)

Using a CPA attack, the attacker is searching for the best key byte estimation out of the set of 256 key byte estimations. In figure 4.3 the correct key byte is marked with the black line color. It can be clearly seen that the correct key byte has the highest correlation values compared to the other 255 key byte guesses. At sample position 18 and 25, the correct key byte has a significant higher correlation.

In the plots of figure 4.4.a and upcoming 'correlation versus number of traces plots'), only the absolute correlation value has been plotted. This is done to give a more clear view of the curve. Figure 4.4.a expresses how good a correlation is and how the correlation of the right key byte changes over the increasing number of traces. The left plot only shows 64 of the 256 key byte candidates in the key byte estimation.

Figure 4.4.a shows that after 5742 traces, key byte 1 is estimated with the highest correlation and the value slowly increases above the correlation values of the other key byte guesses after the number of traces increases.

Figure 4.4.b shows the key byte estimation of all the 256 key byte candidates ranging from value 0 to 255 on x-ax and the correlation on the y-ax. It can be clearly seen that the correlation of the correct key byte 208 (0xD0) has clearly the highest correlation.

4.2.1.2 CPA key byte 16

Figure 4.5 shows the correlation of the estimation of key byte 16 using a CPA attack.
Three significant peaks can be seen in the attack result, these peaks all belong to the correct key byte. The correct key byte is marked with a black line.

The correct estimation of the key byte after a number of traces is relatively quick for key byte 16. Keybyte 16 had a ranking 1 estimate after 1386 traces. Figure 4.6.a and figure 4.6.b show the correlation results.

Over all 16 key bytes, the average number of traces to estimate a key byte correctly is 2846 traces with a minimum of 693 traces and a maximum of 7029 traces.
4.2.2 DPA Attack

This section discusses briefly the DPA attacks performed on the synchronous AES cipher design using a trace set of 10000 traces.

The DPA attack for the synchronous AES design was not successful. None of the key bytes were correctly retrieved after 10000 traces. Only key byte 4 arrived in the top 4 ranking candidates. And 6 out of the 16 key bytes were not even in the top 64 ranking key bytes candidates. For the DPA attack to be successful, more traces are needed.

4.2.2.1 DPA key byte 1

Figure 4.7 shows the attack result of the DPA attack. It can be clearly seen that the correct key byte (marked with a black line) doesn’t have a significant higher correlation compared to the other key byte estimates. In fact other key bytes correlate higher than the correct key byte value.

Figure 4.8.a shows that using a trace set of 10000 traces doesn’t result in a good estimate. Keybyte 1 arrived after 3465 traces in the top 64 ranking key byte candidates and after 7128 traces in the top 32 key byte candidates. Furthermore it did not come in a higher ranking scale after 10000 traces. Figure 4.8.b shows that there are 3 peaks and the correct key byte value has a peak, but the peak value of the correct key byte has a correlation that is smaller than the other 4 bigger peaks that can be seen.

4.2.2.2 DPA key byte 16

Figure 4.9 shows the DPA attack performed on key byte 16.

Key byte 16 arrived after 3465 traces in the top 64 ranking key bytes and after
Figure 4.7: Correlation estimation key byte 1 versus samples using DPA with 10000 traces

Figure 4.8: DPA Key byte 1 (value 0xD0, 208) correlation vs number of traces on synchronous AES cipher

7524 traces in the top 16 ranking key bytes. After 9009 traces it reached the ranking top 8 and after 9207 traces it even reached the top 4 ranking key byte candidates. However it did not reach the top 1 ranking using a trace set of 10000 traces in the attack. Although it was estimated better than key byte 1, several other peaks can be seen in figure 4.10.b. These are the ranking top 4 key byte candidates and had a higher ranking position than the correct key byte.
4.3 Attack on Asynchronous AES

This section discusses the attacks on the asynchronous 4-phase PetriNet based AES cipher, implemented on a Virtex5 FPGA on the Sasebo-GII Side Channel Attack platform board.

Since the secret key of the asynchronous AES cipher was harder to estimate correctly than the synchronous AES cipher, more traces were needed. Using 50000 traces, the secret key could be retrieved successfully.

This section discusses the key estimation results after 50000 traces. Appendix B figure B.9 till figure B.16 show the attack results after 10000 traces. And figure B.17 till figure B.23 show the attack results after 50000 traces.
Figure 4.11 shows the raw trace of the asynchronous AES cipher.

![Figure 4.11: Raw trace of asynchronous AES cipher on SaseboGII](image)

Compared to the synchronous AES signal in figure 4.1, the asynchronous AES signal has a peak voltage of 200mV and -300mV resulting in an approximately 5 times smaller peak to peak voltage signal.

The point of attack for both CPA as DPA attack, is the last round of the AES, see figure 4.12.

![Figure 4.12: Attack point asynchronous design, the last AES round, showing 64 traces](image)

Before the cryptanalysis is performed on the asynchronous power traces, also signal processing steps are performed. Details of the signal processing steps are similar as for the synchronous AES power traces and are in more detail described in section 4.2. Interesting to note is that the asynchronous power traces were not harder to align, there was not significant enough variation in terms of timing that made asynchronous logic alignment harder.

First, the last AES round is selected and the trace set was trimmed back to only the samples of the last round operation of AES. Secondly, all traces are aligned, using static alignment. Finally, all traces are re-sampled from 1Gs/s to 100Ms/s, resulting in a trace set with each trace containing 81 samples.
4.3.1 CPA Attack

This section describes the CPA attack performed on the asynchronous AES cipher using a trace set of 50000 traces.

4.3.1.1 CPA key byte 1

Figure 4.13: Correlation estimation key byte 1 versus samples using CPA with 50000 traces

In figure 4.14.a it can be seen that after 7920 traces key byte 1 was ranking number 1 of the key byte estimation, i.e. the right key guess. And figure 4.14.b shows that after 50000 traces the correct key byte can be easily detected by its high correlation value, which is significantly higher than the other estimates.

Figure 4.14: CPA Key byte 1 (value 0xD0, 208) correlation vs number of traces on synchronous AES cipher
4.3.1.2 CPA key byte 16

Figure 4.15 shows the correlation of the estimation of key byte 16 using a CPA attack.

Figure 4.15: Correlation estimation key byte 16 versus samples using CPA with 50000 traces

In figure 4.16.a it can be seen that after 20295 traces key byte 1 was ranking number 1 of the key byte estimation, i.e. the right keyguess. And figure 4.16.b shows that after 50000 traces the correct key byte can be easily detected by its highest correlation value, which is significantly higher that the other estimates.

Figure 4.16: CPA Key byte 16 (value 0xA6, 166) correlation vs number of traces on synchronous AES cipher

As can be seen in figure 4.14 and figure 4.16, both key bytes can be easily detected by their higher correlation value. For all 16 key bytes, the average number of traces to estimate a key byte correctly is 14416 with a minimum of 4950 traces and a maximum of 35145 traces.
4.3.2 DPA Attack

This section describes the DPA attack using a trace set of 50000 traces.

4.3.2.1 DPA key byte 1

Figure 4.17 shows the attack result of the DPA attack on key byte 1. It can be observed that the correct key byte value (marked with the black line) has peaks with a significant higher correlation value compared to the other key byte estimates.

![Correlation estimation key byte 1 versus samples using DPA with 50000 traces](image)

Figure 4.17: Correlation estimation key byte 1 versus samples using DPA with 50000 traces

Figure 4.18.a shows that key byte 1 is correctly estimated with a continuous ranking number 1 after 23265 traces. A small ghost peak can be observed in figure 4.18.b of key byte candidate 45 (0x2D) but the correct key byte clearly has a higher correlation than this ghost peak.
Figure 4.18: DPA Key byte 1 (value 0xD0, 208) correlation vs number of traces on synchronous AES cipher
4.3.2.2 DPA key byte 16

Figure 4.19 shows the correlation of the estimation of key byte 16 using a DPA attack with 50000 traces.

![Figure 4.19: Correlation estimation key byte 16 versus samples using DPA with 50000 traces](image)

Figure 4.20.a shows that key byte 16 is correctly estimated with a continuous ranking number 1 after 27225 traces. In figure 4.20.b it can be seen that this estimate has a significant higher correlation that other key byte estimates.

![Figure 4.20: DPA Key byte 16 (value 0xA6, 166) correlation vs number of traces on synchronous AES cipher](image)

Figure 4.20: DPA Key byte 16 (value 0xA6, 166) correlation vs number of traces on synchronous AES cipher

72
4.4 Comparison of Power Analysis attacks

This section compares the Power Analysis attacks on asynchronous design versus the synchronous design.

4.4.1 Ranking Based Comparison

This section discusses the comparison of the Power Analysis attacks on the asynchronous AES hardware design and the synchronous AES hardware design based on key byte ranking. Since the CPA attack performed best, the results of the CPA attacks of both design types are compared in this section.

Substituting the ranking results of all the key bytes of the CPA attack for both synchronous and asynchronous AES attacks, the following comparison based on key byte ranking can be made: figure 4.21.a shows the ranking of both designs. Figure 4.21.b shows the number of correct estimated key bytes of each design versus the number of traces. Both of these figures are constructed using figure B.27 and figure B.28 in Appendix B.3.

The ranking displayed in figure 4.21 is defined as follows: when the key estimation of all key bytes (number is determined by the highest ranking key byte) was equal or less than the ranking number (64, 32, 16, 8 or 4), a point was made. A perfect match, i.e. when the correct key byte was estimated, is indicated with a ranking of 1.

Figure 4.21: Ranking of all key bytes of CPA attack of both synchronous and asynchronous AES hardware implementation

As can be seen in figure 4.21.a there is almost a constant factor of 5 times difference between the attack ranking results of the synchronous and the asynchronous AES cipher.

In figure 4.21.b it can be seen that for almost each correct number of estimated key bytes, there is a factor 5 between the number of traces of the two different designs. On average, over each correctly estimated number of key bytes, the difference is 5.43
times. After 7029 traces all the key bytes of the synchronous AES cipher were correctly estimated, and after 35145 traces all the key bytes of the asynchronous AES cipher were correctly estimated.

### 4.4.2 Noise-based Comparison

This section compares the CPA attacks of the synchronous AES attacks with the asynchronous AES attacks based on the attack noise [45], [26].

Since the DPA attacks did not succeed with all attack attempts in this research, the DPA attack results are not considered in the noise based comparison. One remark of the DPA attacks is that DPA attacks reduce the attack noise energy by averaging the signal parts that induce the same change of energy [30] (due to the same bit value, or the same hamming weight).

Noise is estimated by computing the standard deviation of all traces. Noise consists of different components:

1. measurement noise
2. algorithmic noise

Measurement noise covers quantization noise that is due to the A/D converter used to sample power traces and other noise like environmental parasitics etc. Algorithmic noise is the noise due to the randomness of the data bytes being processed.

By analyzing the attack noise, one can make a conclusion of how effective a countermeasure is, e.g. using asynchronous logic against Power Analysis attacks.

#### 4.4.2.1 SNR of Correlation Coefficient

The SNR (Signal-to-Noise Ratio) is the best way to define how much the signal (correlation peak) is hidden in the noise [25]. The lower the SNR, the lower the correlation between the hypothetical power consumption (matrix H) and the traces (matrix T), see section 2.3.4.2.

The noise is calculated by taking the standard deviation of the correlations of each attacked key byte. The signal is the correlation of the correctly estimated key byte. In the next sections the measurement noise and the algorithmic noise of the attacks are determined for both designs.

**SNR Synchronous CPA** Figure 4.22 shows in the top figure the correlation of each correctly estimated key byte (total of 16) over the samples. The bottom figure shows the noise of all key byte guesses.

In the bottom graph of figure 4.22, peaks can be observed at sample positions 18, 26 and 35. These points have the following noise levels, respectively 0.01169, 0.01139 and 0.01136.

The different types of noise can be observed in figure 4.22. At a noise level of 0.01, a constant noise level can be observed. This noise can be categorized as measurement noise. The peaks above 0.01 are categorized as algorithmic noise.
Figure 4.22: Signal and Noise of CPA attack synchronous AES

<table>
<thead>
<tr>
<th>sample pos/key byte</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>18</td>
<td>5.62</td>
<td>2.75</td>
<td>6.77</td>
<td>5.41</td>
<td>6.20</td>
<td>5.97</td>
<td>7.86</td>
<td>5.53</td>
<td>7.08</td>
<td>8.08</td>
<td>6.14</td>
<td>9.26</td>
<td>4.92</td>
<td>4.72</td>
<td>6.93</td>
<td>7.03</td>
</tr>
<tr>
<td>26</td>
<td>5.72</td>
<td>4.16</td>
<td>5.82</td>
<td>4.71</td>
<td>7.93</td>
<td>5.71</td>
<td>6.96</td>
<td>5.33</td>
<td>5.86</td>
<td>9.57</td>
<td>5.15</td>
<td>7.71</td>
<td>5.89</td>
<td>4.34</td>
<td>6.19</td>
<td>7.20</td>
</tr>
</tbody>
</table>

Table 4.2: SNR synchronous CPA

The SNR is determined for each of these three sample positions 18,26 and 35. For each key byte the SNR results are listed in table 4.2
SNR Asynchronous CPA Using 10000 Traces  Figure 4.23 shows the signal and noise plot for of the attack.

![Signal CPA (keybyte 1:16) samples vs Correlation ρ](image)

**Figure 4.23: Signal and Noise of CPA attack asynchronous AES after 10000 traces**

Table 4.3 shows the SNR values for sample positions 34, 39 and 60 after 10000 traces.

<table>
<thead>
<tr>
<th>sample pos/key byte</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>34</td>
<td>3.88</td>
<td>3.45</td>
<td>3.09</td>
<td>3.84</td>
<td>3.18</td>
<td>3.09</td>
<td>1.93</td>
<td>4.73</td>
<td>1.53</td>
<td>0.33</td>
<td>2.97</td>
<td>3.90</td>
<td>3.87</td>
<td>3.57</td>
<td>3.46</td>
<td>1.52</td>
</tr>
<tr>
<td>39</td>
<td>3.85</td>
<td>1.55</td>
<td>1.40</td>
<td>1.07</td>
<td>1.57</td>
<td>2.00</td>
<td>2.63</td>
<td>2.06</td>
<td>3.47</td>
<td>2.46</td>
<td>2.3</td>
<td>1.53</td>
<td>2.25</td>
<td>2.39</td>
<td>1.93</td>
<td>1.97</td>
</tr>
<tr>
<td>60</td>
<td>2.18</td>
<td>0.35</td>
<td>0.28</td>
<td>0.016</td>
<td>0.43</td>
<td>1.28</td>
<td>1.38</td>
<td>1.29</td>
<td>0.97</td>
<td>1.32</td>
<td>1.66</td>
<td>0.41</td>
<td>1.14</td>
<td>2.35</td>
<td>0.68</td>
<td>0.70</td>
</tr>
</tbody>
</table>

Table 4.3: SNR asynchronous CPA after 10000 traces

As can be seen in table 4.3 the SNR values are significantly lower than the CPA attack on the synchronous design after 10000 traces. The SNR values range from close to 0 to 3.47. It can be observed that the side channel signal is more buried in the noise.

SNR Asynchronous CPA using 50000 traces  Using five times more traces, the SNR of asynchronous CPA is determined again using three peak points of the CPA noise after 50000 traces, see figure 4.24.

In figure 4.24 the peaks are at positions 34, 41 and 49. The noise levels for each of those points are respectively 0.005277, 0.005225 and 0.0051. The results are shown in table 4.4.
As can be observed, the SNR is now significantly higher, which means that more side channel signal is present than when using 10000 traces. It is also interesting to see that the measurement noise is reduced by a factor 2.23 (0.0101/0.00452 = 2.23, see red lines figure 4.23 and figure 4.24) when using 5 times more traces.

**Summary** The SNR of the correlation coefficients of the synchronous design using 10000 traces range between 2.75 to 9.75. The SNR of the correlation coefficients of the asynchronous design using 10000 traces range between close to 0 and 3.47. This is significantly lower than the synchronous design. Clearly for the asynchronous design the signal is buried into more noise. However, when using more traces, more side
channel signal appears. When using 50000 traces for the asynchronous design, the SNR of the correlation coefficients ranges between 2.81 and 7.87. The SNR values of the synchronous design using 10000 traces are quite similar to the SNR values of the asynchronous design using 50000 traces. Using the asynchronous design resulted in a lower SNR. It took more traces to get the SNR of the correlation coefficient towards the same SNR level of the synchronous design.
4.4.3 Significance and Estimation of Number of Traces

This section discusses the estimation of the number of traces needed to break the secret key.

In [30], a countermeasure of adding random calculations is discussed with the purpose of increasing the noise level. A rough estimation of the noise was made by observing the probability density function (p.d.f.) of the DPA bias signal. The peak (i.e. the correct key guess) should obviously stand out. Several definitions, formulas and techniques that are described in section 2.3.5 are used in the calculations of the following sections.

**Synchronous AES after 10000 Traces** Figure 4.25 shows a histogram and a probability density plot of key byte 1 and key byte 16 of the synchronous AES after 10000 traces. The correlation coefficients are first transformed using Fisher’s transform. Then Matlab is used to make a normal distribution fit. This displayed with a red line in figure 4.25.

The CPA of key byte 1 had a maximum ranking peak value of \( \rho = -0.06567 \). Using Fisher transformation this becomes \( Z = -0.0318 \). For the CPA of key byte 1, in order to distinguish if the peak \( \rho = -0.06567 \) is distinguishable from \( \rho = 0 \), the normalized difference can be determined using formula 2.31. The normalized difference with a \( \alpha = 0.0001 \) is 4.6496, the critical value \( z_{1-\alpha/2} = 3.8906 \). Since 4.6496 is outside the critical value boundary, \( \rho = -0.06567 \) is statistically significant and will give a peak (as is the case). The lower bound of calculated traces to distinguish \( \rho = 0 \) with \( \rho = -0.06567 \) is \( n=6399 \) traces.

The CPA of key byte 16 has a maximum ranking peak value of \( \rho = 0.08619 \) using Fishers transformation \( Z = 0.0451 \). The normalized difference with a \( \alpha = 0.0001 \)
is -6.1088. This is above the critical value. Therefore $\rho = 0.08619$ is statistically significant and will give a peak as can be observed in the attack results. The lower bound of calculated traces to distinguish $\rho = 0$ with $\rho = 0.08619$ is $n=3708$ traces.

Both estimations for key byte 1 and 16 are considered statistically significant for a correct key byte estimation. As can be seen in the results, key byte 16 has a stronger significance than key byte 1.

Asynchronous AES after 10000 Traces Figure 4.26 shows a histogram and probability density plot of key byte 1 and 16 of the asynchronous AES after 10000 traces.

In this attack, key byte 1 had a maximum ranking peak value of $\rho = 0.0498$. Using Fisher transformation this becomes $Z = 0.0255$. Using formula 2.31, the normalized difference for the peak of key byte 1 is -3.5238, which is smaller than the critical value $z_{1-\alpha/2} = \pm 3.8906$. This means that no significant peak will be observed, which is true as can be seen in the attack result of key byte 1. In figure 2.25 in section 2.3.5, it can also be observed that $\rho = 0.0498$ is lower than the critical $\rho = 0.055$.

The CPA of key byte 16 has a peak value of $\rho = -0.02758$. Using Fishers transformation, this becomes $Z = -0.0136$. The normalized difference of key byte 16 is 1.9504, which is smaller than the critical value $z_{1-\alpha/2} = \pm 3.8906$. This also means that no significant peak will be observed, which is confirmed in the attack results of key byte 16.

Although key byte 1 was more significant than key byte 16, both these key bytes are not considered statistically significant for a correct key byte estimate with an error probability of $\alpha = 0.0001$. Observe that key byte 1 was close to the critical value. It is clear that more traces are needed in order to have significant estimates.
Asynchronous AES after 50000 Traces Figure 4.27 shows a histogram and a probability density plot of key byte 1 and key byte 16 of the asynchronous AES after 50000 traces.

The CPA of key byte 1 has a maximum ranking peak value at $\rho = 0.04014$. Using the Fisher transformation this is $Z = 0.0205$. For the CPA of key byte 1, in order to distinguish if the peak $\rho = 0.04014$ is distinguishable from $\rho = 0$, the normalized difference formula 2.31 is used. The normalized difference with $\alpha = 0.0001$ is $Z = -6.3499$ and the critical value $z_{1-\alpha/2} = 3.8906$. Since -6.3499 is outside the critical value boundary, $\rho = 0.04014$ is statistically significant and will give a peak (as is the case). The lower bound of calculated traces to distinguish $\rho = 0$ with $\rho = 0.04014$ is $n= 17153$ traces.

The CPA of key byte 16 has a maximum ranking peak correlation coefficient of $\rho = 0.02839$. Using Fishers transformation this becomes $Z = 0.0144$. The normalized difference for key byte 16 is -4.4899. This is higher than critical value $z_{1-\alpha/2} = 3.8906$ and will result in a peak. The lower bound of calculated traces to distinguish $\rho = 0$ with $\rho = 0.02839$ is $n= 34305$ traces with $\alpha = 0.0001$.

Both key byte 1 and 16 are considered statistically significant for a correct key byte estimation. As can be seen, key byte 1 has a stronger significance than key byte 16.

Summary The attack results of the synchronous design using 10000 traces has a similar order of normalized difference as the asynchronous design using 50000 traces. Furthermore, the results of these two attacks are statistically significant and result in peaks. Both normalized differences are between 4 and 6.

While the asynchronous design using 10000 traces had a lower normalized difference (between 1.95 and 3.5) both key byte results were lower than the critical value and are considered statistically insignificant. The sensitive data was well hidden among
the other key byte correlations. Clearly more traces were needed in order to obtain significant correlations.
Conclusion and Future Work

This chapter describes the conclusion and future work. Section 5.1 discusses the conclusion. Section 5.2 discusses future work and final thoughts.

5.1 Conclusion

The final conclusion is based on the meeting of goals that were set in the beginning of this document in section 1.2. Section 5.1.1 discusses the meeting of these goals.

5.1.1 Meeting the Goals

This section addresses each of the goals from section 1.2 set for this thesis project along with the results.

5.1.1.1 Analyze Different Asynchronous Logic Design Styles that Meet the Cost Requirement

In section 2.2.5 it was shown that only a limited set of asynchronous logic implementations were feasible to implement in the FPGA device in terms of area. This means that the cost of area of implementing asynchronous logic styles like dual-rail based designs is too high. There was an asynchronous logic style based on PetriNet controllers that is a light weight and area efficient that fits in terms of the area costs.

5.1.1.2 Implement a Synchronous and an Asynchronous AES Core with Comparable Hardware Layout

The PetriNet-based controller asynchronous logic design style was chosen to be implemented in the FPGA, and it was analyzed against Power Analysis attacks and compared with a synchronous (clocked) AES cipher design.

In terms of architecture, both designs are equal, only the logic design style is different.

Chapter 3 discussed hardware design details of both designs. For the asynchronous design the surrounding interface logic in FPGA is not changed in order to make sure that only the AES core implementations are different.

5.1.1.3 Analyze and Compare Power Analysis Attack Results

As expected, the CPA attack performed clearly better than the DPA attack, because the power model of DPA is a binary model and precision was lost when converting a
hamming weight model to a binary model. For that reason the comparison of Power Analysis attacks were done using CPA attacks.

Section 4.4 made two type of comparisons. One is based on key byte estimation ranking, and the other one is based on noise comparison.

In the key byte estimation ranking comparison, it can be seen that asynchronous hardware is a factor 5.43 times harder to break in terms of number of needed power traces.

The noise based comparison showed that after taking an equal amount of traces of both synchronous and asynchronous AES cores, the SNR of synchronous core ranged from 2.75 to 9.57 and the asynchronous core ranged from 0 to 3.47. The asynchronous implementation clearly buries the signal into more noise.

However, after more traces it is takes less than five times more traces for the signal to appear significantly over the noise. Gaining five times more traces is a number that is not hard to achieve to in terms of time. It is still achievable within an hour, in contrast of multiple days, weeks or months.

As noted in section 4.3 the asynchronous traces were not hard to align, so also no advantage was booked in terms of dis-alignment.

5.1.1.4 Evaluate how significant asynchronous logic is a countermeasure to Power Analysis attacks

The PetriNet controller based asynchronous design did bring more security by burying the signal more into noise, but it did not significantly resulted into a more secure design than the synchronous design implementation.

By accepting asynchronous logic styles that cost more in terms of area, most likely a higher level of security can be reached e.g. dual-rail designs with alternating or arbitrary order of spaces or other power balancing techniques. The question is how much is one willing to pay (in terms of area for example by buying larger and more expensive chips, power usage, design time cost etc.) for a more secure chip or system?

5.2 Future Work and Final Thoughts

The area of attacking cryptographic hardware and securing the hardware is an ongoing game where the improvement of attacks, and the research to cost efficient counter measures, are still lively areas of research.

At the time of writing, there is a great need to bringing security aspects earlier into the hardware design chain. A start of this was made in section A.1 with examples and attack results. By bringing security analysis early in the design cycle, this would greatly improve the design time and would support the analysis and improvement of security in a early stage of the design time. By exposing countermeasures and different logic cell libraries more earlier into the design chain and by making them accessible for designers, more awareness will be introduced to the designers that will ultimately give a boost to the design of security applications.

Asynchronous logic will most likely not become a number one candidate in security
applications unless it becomes easy accessible (more tools and documentation supporting it) and with a better costs in terms of area.

Making secure designs with an acceptable costs is a challenge that requires ongoing research and innovations until the day comes that the attackers will throw in the towel.
Bibliography


[16] David D. Hwang, Kris Tiri, Alireza Hodjat, Student Member, Bo cheng Lai, Student Member, Shenglin Yang, Student Member, Patrick Schaumont, Ingrid Verbauwhede, and Senior Member. Aes-based security coprocessor ic in 0.18um cmos with resistance to differential power analysis side-channel attacks, 2006.


[27] Stefan Mangard, Norbert Pramstaller, and Elisabeth Oswald. Successfully attacking masked aes hardware implementations. In in Proc. Workshop on Cryptographic


This appendix discusses work that is related to the thesis and work that is left for future endeavors. Section A.1 discusses future work of simulation-based power model that enables designers to bring Power Analysis attacks into the design chain. Section A.2 discusses the effect of subtracting a related pure noise trace from a power trace.

### A.1 Simulation based Power Model

The Verilog Programming Language interface, commonly called Verilog PLI is used to build a power model based on Post place and route simulation. Section A.1.1 introduces how Side Channel Analysis can be embedded into the hardware design process. Section A.1.2 gives an introduction to the Verilog PLI. Section A.1.3 describes the use of Verilog PLI as a simulation model. The last section A.1.4 briefly evaluates the results and the performance of the Verilog PLI application.

#### A.1.1 Embedding Side Channel Analysis in the hardware design process

This section discusses the motivation to embed Side Channel Analysis in the hardware design cycle.

Using Verilog PLI power traces fulfills a need of designers to able to evaluate their designs on side channel leakage in the design chain. This enables the designer to build side channel resistant hardware during the design phase, not as an add on later. A problem is that there is no good solution to implement side channel analysis in the early stage of the hardware design. Designers had to first implement a design and do the Power Analysis in the latest design phase, when the design is implemented in the chip. Some approaches have been made [38], [31],[49] but none of them proposed a lightweight solution and a solution without being bounded to a certain design technology. Different design methodologies can be tested and evaluated.

Another big advantage of this solution is that a designer is not restricted to only simulate the total design, rather one is able to simulate any module of the design separately. Figure A.1 shows simulated power traces of an Sbox with input values 0x29, 0x23, 0xBE respectively and output values 0xA5, 0x26 and 0xAE. It can be clearly seen that the power model is data dependent.

#### A.1.2 Introduction to Verilog PLI

Verilog PLI [44] is a user programmable procedural interface for Verilog digital logic simulators. It is a mechanism to invoke C or C++ functions from Verilog code. The Verilog PLI standard has a library of C functions, also called PLI routines. Verilog PLI
Verilog PLI provides access routines, also called acc routines, to access the internal data structure of the Verilog simulator. Another set of routines, known as task and function routines, abbreviated as tf routines. The acc and tf routines are described in PLI 1.0. The latest release PLI 2.0 is backwards compatible and contains newer routines called vpi routines. A user can use these routines to create application specific PLI routines. This opens a world of opportunities to interface any program to Verilog simulators.

Verilog PLI provides system tasks and system functions that a user can define. These are noted with the prefix of the $ sign. When a system task is called, the simulator branches to the particular C or C++ function and executes that particular code. When the execution is completed, the simulator executes the next statement in the Verilog test bench. When a system function is called, the same steps as with a system task are performed with the addition that a system function returns a value into the simulation. The return value can be a bit value, a bit vector or an int or a floating point number.

Each system task or function is associated with four types of functions:

1. Checktf
2. Calltf
3. Sizetf
4. Misctf

These four functions are only invoked by the simulator itself and not by each other.

**A.1.2.1 Steps to implement Verilog PLI**

This section describes briefly how to implement a C or C++ function and call it from a Verilog test bench.

First one writes C/C++ code containing PLI1.0 or PLI2.0 routines and any other desired code. Then this code needs to include libraries and included files from the
specific simulator and altogether compiled to a shared library in Windows this is a *
.dll file in in UNIX a *.so file. The shared library can be included in the ModelSIM simulator. In ModelSIM, the modelsim.ini file needs to be modified and a path needs to be set to the shared library containing the users C functions. Also the Vendor IDE, e.g. Xilinx ISE simulation properties need to be configured to include the PLI shared library.

Second the PLI routines need to be called from the Verilog test bench, with a prefix a "$" sign the corresponding C function can be called.

Finally, the simulator can be run like any Verilog simulator and the PLI functions will be executed during simulation.

A.1.2.2 Verilog PLI functions for Power Simulation

This section describes the Verilog PLI functions that are created to interface with the shared library.

Figure A.2 shows the Verilog PLI functions in the shared library PLIpoweranalysis.dll. These functions are invoked from the Verilog test bench in a ModelSIM simulation. Using these PLI functions, the C code has functions that can generate power trace in a trace format *.trs or in a comma separate file format *.csv. The *.csv file can be used for many applications and e.g. parsed by scripts.

The PowerTrace.trs file, is a file that is compatible with Riscure’s Side Channel Analysis software and it contains the set of all the traces and additional data. Each trace is generated using a fixed key and random plain text. The plain text data and the cipher text are attached for each trace in the PowerTrace.trs file. After completing the full simulation and after acquiring all the trace data, the PowerTrace.trs file can be imported in Riscure’s Side Channel Analysis software, Inspector [3]. By embedding Riscure’s Inspector[3] in the design process tools, or use this Side Channel Analysis platform together with the design tools, the Side Channel Analysis has been brought in the early stage of the hardware design cycle.

The PLIpoweranalysis.dll contains implementations the of all the PLI functions mentioned in figure A.2:

1. $init
2. $RandomPlainText
3. $SendCipherText
4. $finish trace
5. $closeHDfile

The $init function initializes all the available signals and starts a monitor that monitors all the signal changes during the simulation. The init function also sets the header file of the generated output file ”PowerTrace.trs”

The $RandomPlainText function generates a random plain text input for each single trace. It creates 16 random bytes and returns it as a hexadecimal string to the simulator. The simulator takes this hexadecimal string as input stimuli to the AES core. In the
PowerTrace.trs file the random plain text data is added at the beginning of each trace (see table A.1).

The $SendCipherText function sends the result of encryption of each single trace back to the PLI shared library. Then this value is written to the PowerTrace.trs file directly after the plain text field (see table A.1).

The $finish trace function adds samples with the value zero in the remaining space of the total sample time for each trace. All traces have the same length, i.e. the same sample time. Before and after the encryption operation there is a little extra sample space. When the encryption has been executed, this 'finish trace' function adds samples with the value zero before starting a new trace.

The $closeHDfile closes the PowerTrace.trs file and when selected, it also closes the *.csv file when the simulation has executed all the traces.

The PowerTrace.trs file has the following structure, see table A.1.

### A.1.3 Power Simulation Model

This section discusses the power simulation model.

In order to create a power model of the design, the total switching activity of all the nets in the design needs to be determined. This can be done using the post synthesis place and routed netlist of the design.

With Verilog PLI it is possible to get a list of all signals in the design and record the change of logical level for each time unit by gaining full access to the simulator of the FPGA IDE, e.g. ModelSIM simulator.

To model the power, the Hamming Distance model is used according to the following
Table A.1: Riscure’s Inspector *.trs, trace format

\[
Power(t) = \sum_{i=1}^{t_s} \sum_{i=0}^{n-1} HW(sig_i(t-1) \oplus sig_i(t)) \quad (A.1)
\]

Here \( t_s \) represents the total sample time of the encryption operation. \( n \) represents the total number of all the signals in the hardware design. The model above records each logical change from 0 \( \Rightarrow \) 1 or 1 \( \Rightarrow \) 0 for each signal as equal in terms of power consumption. This is done by determining the Hamming distance by adding a 1 for each logical change of a signal. Every hamming distance value is recorded and added for each single time unit. This gives a power value for each time interval \( t \).

As explained in section 2.3.2 CMOS technology based devices, consume power in the transitions. The Hamming Distance model does not take into account the difference between logical 0 \( \Rightarrow \) 1 or 1 \( \Rightarrow \) 0 transitions, which do exist in CMOS power characteristics. The Hamming Distance model does not represent the entire power consumption of the chip, it only represents the data depended part. Since the data dependent part is the part that useful for Side Channel Attacks this model is sufficient to use for SCA.
A.1.4 Results and Performance

This section discusses the results of the Verilog PLI simulation based power traces. A synchronous AES cipher will be discussed as well as the asynchronous version. Also the performance in terms of simulation time and file size will be discussed.

A.1.4.1 PLI Synchronous AES

This section discusses briefly the attack results of the Verilog PLI based acquired power traces of the same synchronous AES design, that is attacked in section 4.2. The only difference is that all surrounding control logic is removed, purely the synchronous AES component implementation is attacked.

Figure A.3, A.11, A.12 display the Verilog PLI acquired PowerTrace.trs. This format can be directly used in Riscure’s Side Channel Platform Inspector [3] for further signal processing and analysis and eventually to perform the attack.

Figure A.3 is the Verilog PLI power trace of the synchronous AES cipher.

![Figure A.3: PLI power trace of the synchronous AES cipher](image)

In figure A.3 the clock cycles can be clearly seen (the smaller peaks with value 7) also can be seen that the logic is clearly ready before the next clock cycle as mentioned in section 2.2. The big peaks are the glitches that occur through the modules and end they go back to practically the value 0 which is the time that the data is ready and no more changes occur through the logic until the next clock cycle. The peak value of this trace is 212; this is equal to 212 transitions on that sample.
The last round was selected and statically aligned. After alignment all traces were re-sampled to a smaller number of samples so the attack will be taking less computation time. This re-sampled set forms the attack point of the synchronous AES PLI simulation and is displayed in figure A.4.

Figure A.4: Signal of last round of Synchronous AES PLI simulation

The CPA attack result on the selected last round from figure A.4 of keybyte 1 is displayed in figure A.5. A strong correlation approximately 0.14 can be seen.

Note that the noise is very minimal. There is sampling noise due to quantization of the simulation precision but no parasitic noise or environmental noise. In [26] it was mentioned that in practice CPA attack results range from 0 to 0.2. In this simulation, the results come close to this upper bound value.

Figure A.5: PLI synchronous AES attack result keybyte 1

Figure A.6 shows the key byte correlation of key byte 1 after 10000 traces and the
evaluation of the correlation coefficient as the number of traces increases. As can be seen, after a small number of traces, key byte 1 was estimated with high confidence.

Figure A.6: CPA Key byte 1 (value 0xD0, 208) correlation vs number of traces on Synchronous AES cipher

Figure A.7 shows the attack result of key byte 16 after 10000 traces. Again, strong correlation of approximately 0.13 can be observed.

Figure A.7: PLI synchronous AES attack result key byte 16

Figure A.8 shows the key byte correlation of key byte 16 after 10000 traces and the evaluation of the correlation coefficient as the number of traces increases. As can be seen, key byte 16 is after a small number of traces also estimated with high confidence.

SNR  As discussed in section 4.4.2 this subsection will briefly display the signal and noise characteristics of the Verilog PLI based simulation. Figure A.9 shows the signal
Figure A.8: CPA Key byte 16 (value 0xA6, 166) correlation vs number of traces on Synchronous AES cipher

and noise plot of the complete (i.e. all estimated key bytes) synchronous AES CPA attack.

As can be observed in figure A.9 there is relatively constant noise level of 0.01. This noise can be seen as measurement noise. The three highest values are used to determine the SNR of those sample locations and are displayed in table A.2.

<table>
<thead>
<tr>
<th>sample pos/keybyte</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>8.20</td>
<td>7.36</td>
<td>11.40</td>
<td>6.96</td>
<td>1.99</td>
<td>3.75</td>
<td>2.16</td>
<td>3.31</td>
<td>2.52</td>
<td>4.85</td>
<td>6.45</td>
<td>6.46</td>
<td>5.50</td>
<td>5.11</td>
<td>5.41</td>
<td>5.73</td>
</tr>
<tr>
<td>15</td>
<td>0.73</td>
<td>1.05</td>
<td>3.80</td>
<td>1.22</td>
<td>1.43</td>
<td>2.08</td>
<td>0.40</td>
<td>1.13</td>
<td>1.20</td>
<td>0.42</td>
<td>1.96</td>
<td>1.96</td>
<td>1.39</td>
<td>0.90</td>
<td>2.84</td>
<td>2.07</td>
</tr>
</tbody>
</table>

Table A.2: SNR synchronous CPA PLI Verilog

As can be observed in table A.2, at sample 9 the SNR is the highest. Most key bytes have a SNR around 9. Key byte 5,7 and 9 have relatively lower SNR compared to the other key bytes.
Figure A.9: Signal and Noise plot of PLI synchronous AES CPA attack
A.1.4.2 Fully Asynchronous AES PLI

This subsection describes the attack on a fully asynchronous AES PLI. This design is different than the asynchronous design implemented in the FPGA as discussed in section 3.2.3.

In the fully asynchronous AES design, all the surrounding logic and the latches around the AES core component are also made asynchronous.

Figure A.10 shows the fully asynchronous AES cipher implemented with PetriNet controllers.

Due to time limitation, this implementation could not be implemented in the Virtex5 FPGA and its surrounding logic on the Sasebo-GII board.

However the asynchronous design could be analyzed and attacked using Verilog PLI simulations. This section shows the CPA attack results of this fully asynchronous AES design.

Figure A.11: PLI power trace of the asynchronous AES cipher
Figure A.11 shows the Verilog PLI asynchronous AES power trace. The big peaks are the moments that the system gets a reset signal, this changes all the register values to their initial state. This causes many transitions in the FPGA.

Figure A.12 shows a zoomed in view of figure A.11. As can be observed there is no clock in this design. As soon as the data is ready the logic continues computing each round, this is accomplished by the handshaking mechanism. This trace has a peak value of 87. This is approximately 2.4 times smaller than the synchronous version. Figure A.12 counts 10 rounds in the power trace of AES-128 algorithm. In contrast to the synchronous version in figure A.3 that had an output register and counted 11 rounds in the power spectrum. Asynchronous logic is only active when necessary and the logic is stored in the output latch, and the output data does not go through the remaining logic.

Figure A.13 shows the attack point of the simulated asynchronous AES trace. The point of attack is the last AES round.

Figure A.13: Signal of last round of asynchronous AES PLI simulation
Figure A.14: PLI asynchronous AES attack result key byte 1

Figure A.15: CPA Key byte 1 (value 0xD0, 208) correlation vs number of traces on Asynchronous AES cipher
Figure A.16: PLI asynchronous AES attack result key byte 16

Figure A.17: CPA Key byte 16 (value 0xA6, 166) correlation vs number of traces on Asynchronous AES cipher
This subsection briefly discusses the signal and noise results of the asynchronous AES Verilog PLI CPA attack. Figure A.18 shows the signal and noise plot of the complete CPA attack results.

The top three sample locations with the highest noise value are taken to determine the SNR. The results are displayed in table A.3

<table>
<thead>
<tr>
<th>sample pos/keybyte</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>383</td>
<td>6.98</td>
<td>7.90</td>
<td>7.92</td>
<td>2.27</td>
<td>2.80</td>
<td>2.69</td>
<td>5.23</td>
<td>11.18</td>
<td>17.17</td>
<td>11.30</td>
<td>8.86</td>
<td>8.73</td>
<td>10.47</td>
<td>7.28</td>
<td>9.34</td>
<td>4.17</td>
</tr>
<tr>
<td>457</td>
<td>8.86</td>
<td>4.27</td>
<td>7.81</td>
<td>5.06</td>
<td>6.26</td>
<td>5.73</td>
<td>2.43</td>
<td>11.42</td>
<td>13.16</td>
<td>13.62</td>
<td>9.39</td>
<td>2.73</td>
<td>11.74</td>
<td>11.60</td>
<td>3.21</td>
<td>5.86</td>
</tr>
</tbody>
</table>

As can be seen in table A.3 the signal is high on different sample locations. There are high SNR values in both three sample locations. Key byte 9 has very high signal at sample position 383 with a SNR of 17.17. From sample position 383, the cipher text of the last round is written in output latch and correlates greatly with the hypothetical Hamming weight model used in this attack.
Figure A.19.a displays the simulation time to create the PowerTrace.trs file.

Simulation Time  The simulation time is significantly slower compared to measuring FPGA traces with an oscilloscope. But as can be seen in figure A.19.a it is usable. In Cryptanalysis, running traces for a few days is not exceptional.

The simulation time is dependent on the resolution of the simulation as well as the total sample time for the trace. This is also true for the file size of the trace file.

Figure A.19.b displays the file size of the PowerTrace.trs file.

Because the fully asynchronous AES cipher operates faster, more samples were needed and this the file size is significantly larger.

A.1.5 Conclusions and Future Work

Embedding side channel analysis in a early stage of the design chain can definitely contribute to create secure designs in shorter time. It will also give developers more insight into which techniques actually work and which don’t work when working on making designs more secure.

Future work is needed to enhance the power model to a more accurate power model that is similar to the power consumption of the FPGA. More constraints need to be added like environmental noise, crosstalk noise and more detailed device specific noise, accurate voltage values of switching. An open problem to this work is that only power traces running on relatively high frequencies give a usable power trace. When having the device clk running on relatively low frequencies in the order of 2 MHz and lower, the transitions of the logic only appear on the rising and falling edge of the clk, the time in between there is no contribution to the power trace.
A.2 Subtracting Noise Trace

This section discusses the effect of recording a pure noise trace and the subtraction of this trace from the AES encryption trace.

A.2.1 Noise trace

In order to gain insight into what the remaining logic in the Sasebo-GII board does and what the AES cipher actually performs, a special noise trace is generated. To get a noise trace of the influence of the surrounding logic in the FPGA, the AES cipher hardware logic has been completely removed and replaced by a ’transparent’ dummy block that only connects the appropriate input signals directly with its corresponding output signals.

Figure A.20 shows the raw noise trace that is comparable in terms of time to the executing of a whole encryption operation.

![Figure A.20: Noise trace, no AES cipher implemented in hardware](image)

The big peaks at the beginning are the restart signals given to the AES cipher and all of its subcomponents. Following is a continues pattern of nearly positive peaks and larger peaks having a double amplitude but centered around 0 Volts. These larger peaks in this repeating pattern is the system clk period that runs on 2MHz.

A.2.2 Asynchronous Signal Subtracted With Noise

This section analyzes a signal processing step describing the subtraction of the Asynchronous AES signal subtracted with the mean signal of the noise trace.

A.2.2.1 Subtraction

This section discusses the subtraction of the noise trace from the asynchronous AES trace. When observing the signal amplitude and position of the higher peaks of the noise signal, it can be seen that the smaller intermediate peaks between the higher peaks of the asynchronous AES signal match with the noise signal. When subtracting the noise signal from the AES signal it would thus result in three large peaks.

In order to subtract the sampled signals, both traces are in the same sample space and both traces need to be aligned with one another. The subtraction operation has been performed using a mixing operation of Riscure’s Inspector software [3]. The mixing operation used mixes a trace set with a single reference trace using a mix function, e.g.
the subtract operation. Using this mixing operation, the asynchronous AES signal will be more isolated by subtracting from each trace in the trace set the mean of the noise trace set.

Figure A.21 shows the absolute mean noise trace with a low pass filter operation applied.

![Figure A.21: Absolute Noise trace, no AES cipher implemented in hardware](image)

Figure A.22 shows the absolute raw asynchronous AES trace.

![Figure A.22: Absolute Asynchronous AES trace](image)

Both traces are statically aligned before the subtraction operation is executed in order to perform an accurate subtraction. Figure A.23 shows the result of the subtraction of the noise trace from the asynchronous trace.

![Figure A.23: Asynchronous trace with subtracted noise trace](image)
A.2.2.2 CPA Attack

Figure A.24 shows the CPA attack results of keybyte 1.

![Graph showing CPA attack results for keybyte 1](image1)

Figure A.24: CPA Key byte 1 (value 0xD0, 208) correlation vs number of traces on asynchronous AES cipher

Figure A.25 shows the CPA attack results of keybyte 16.

![Graph showing CPA attack results for keybyte 16](image2)

Figure A.25: CPA Key byte 16 (value 0xA6, 166) correlation vs number of traces on asynchronous AES cipher

A.2.2.3 DPA Attack

Figure A.26 shows the DPA attack results of keybyte 16.
Figure A.26: DPA Key byte 1 (value 0xD0, 208) correlation vs number of traces on asynchronous AES cipher

Figure A.27 shows the DPA attack results of keybyte 16.

Figure A.27: DPA Key byte 16 (value 0xA6, 166) correlation vs number of traces on asynchronous AES cipher

A.2.2.4 Conclusion

The figure A.28 shows the ranking 0-63 of all the keybytes.
As can be seen in figure A.28 the DPA attack performed very bad, key byte 6 and key byte 10 were not in the top 64 of the key bytes in the attack after 50000 traces.
By subtracting the noise, valuable sample data of the AES cipher got subtracted as well. The conclusion is, that performing this signal processing step, it did not bring any improvement in the attack, in fact, it made the attack even harder.
A.2.2.5 Attack Plots Asynchronous AES CPA with Noise Subtracted
Figure A.29: Partial key hypothesis of key byte 1-8 using CPA attack, Asynchronous AES with Noise subtracted, 50000 traces
Figure A.30: Partial key hypothesis of key byte 9-16 using CPA attack, Asynchronous AES with Noise subtracted, 50000 traces
Figure A.31: Ranking Partial key hypothesis of key byte 1-8 using CPA attack, Asynchronous AES with Noise subtracted, 50000 traces
Figure A.32: Ranking Partial key hypothesis of key byte 9-16 using CPA attack, Asynchronous AES with Noise subtracted, 50000 traces
Figure A.33: Partial key hypothesis of key byte 1-8 using DPA attack, Asynchronous AES with Noise subtracted, 50000 traces
Figure A.34: Partial key hypothesis of key byte 9-16 using DPA attack, Asynchronous AES with Noise subtracted, 50000 traces
Figure A.35: Ranking Partial key hypothesis of key byte 1-8 using DPA attack, Asynchronous AES with Noise subtracted, 50000 traces
Figure A.36: Ranking Partial key hypothesis of key byte 9-16 using DPA attack, Asynchronous AES with Noise subtracted, 50000 traces
Appendix: Attack results

This appendix lists the attack results of all bytes for each attack.

B.1 Attacks Synchronous AES

B.1.1 CPA 10000 traces
Figure B.1: Partial key hypothesis of key byte 1-8 using CPA attack, synchronous AES
Figure B.2: Partial key hypothesis of key byte 9-16 using CPA attack, synchronous AES
Figure B.3: Ranking Partial key hypothesis of key byte 1-8 using CPA attack, synchronous AES
Figure B.4: Ranking Partial key hypothesis of key byte 9-16 using CPA attack, synchronous AES
B.1.2 DPA 10000 traces
Figure B.5: Partial key hypothesis of key byte 1-8 using DPA attack, synchronous AES
Figure B.6: Partial key hypothesis of key byte 9-16 using DPA attack, synchronous AES
Figure B.7: Ranking Partial key hypothesis of key byte 1-8 using DPA attack, synchronous AES
Figure B.8: Ranking Partial key hypothesis of key byte 9-16 using DPA attack, synchronous AES
B.2 Attacks Asynchronous AES

B.2.1 CPA 10000 traces
Figure B.9: Partial key hypothesis of key byte 1-8 using CPA attack, asynchronous AES, 10000 traces
Figure B.10: Partial key hypothesis of key byte 9-16 using CPA attack, asynchronous AES, 10000 traces
Figure B.11: Ranking Partial key hypothesis of key byte 1-8 using CPA attack, asynchronous AES, 10000 traces
Figure B.12: Ranking Partial key hypothesis of key byte 9-16 using CPA attack, asynchronous AES, 10000 traces
B.2.2 DPA 10000 traces
Figure B.13: Partial key hypothesis of key byte 1-8 using DPA attack, asynchronous AES, 10000 traces
Figure B.14: Partial key hypothesis of key byte 9-16 using DPA attack, asynchronous AES, 10000 traces
Figure B.15: Ranking Partial key hypothesis of key byte 1-8 using DPA attack, asynchronous AES, 10000 traces
Figure B.16: Ranking Partial key hypothesis of key byte 9-16 using DPA attack, asynchronous AES, 10000 traces
B.2.3 CPA 50000 traces
Correlation keybyte1 using CPA Attack

(a) key byte 1

Correlation keybyte2 using CPA Attack

(b) key byte 2

Correlation keybyte3 using CPA Attack

(c) key byte 3

Correlation keybyte4 using CPA Attack

(d) key byte 4

Correlation keybyte5 using CPA Attack

(e) key byte 5

Correlation keybyte6 using CPA Attack

(f) key byte 6

Correlation keybyte7 using CPA Attack

(g) key byte 7

Correlation keybyte8 using CPA Attack

(h) key byte 8

Figure B.17: Partial key hypothesis of key byte 1-8 using CPA attack, asynchronous AES, 50000 traces
Figure B.18: Partial key hypothesis of key byte 9-16 using CPA attack, asynchronous AES, 50000 traces
Figure B.19: Ranking Partial key hypothesis of key byte 1-8 using CPA attack, asynchronous AES, 50000 traces
Figure B.20: Ranking Partial key hypothesis of key byte 9-16 using CPA attack, asynchronous AES, 50000 traces
B.2.4  DPA 50000 traces
Figure B.21: Partial key hypothesis of key byte 1-8 using DPA attack, asynchronous AES, 50000 traces
Figure B.22: Partial key hypothesis of key byte 9-16 using DPA attack, asynchronous AES, 50000 traces
Figure B.23: Ranking Partial key hypothesis of key byte 1-8 using DPA attack, asynchronous AES, 50000 traces
Figure B.24: Ranking Partial key hypothesis of key byte 9-16 using DPA attack, asynchronous AES, 50000 traces
### B.3 Key Byte Ranking

This section shows the Key byte ranking results. Figure B.25 and figure B.26 show the precise ranking results of each key byte estimation with ranking positions ranging from 64 to 1. Figure B.27 and figure B.28 show the results after the ranking has been greater or equal after each ranking border respectively 64, 32, 16, 8, 4 and 1.

![Figure B.25: ranking of all key bytes CPA and DPA attack vs number of traces on synchronous AES cipher](image1)

(a) CPA ranking key bytes 1-16

![Figure B.26: ranking of all key bytes CPA and DPA attack vs number of traces (50000) on asynchronous AES cipher](image2)

(b) DPA ranking key bytes 1-16

Figure B.25: ranking of all key bytes CPA and DPA attack vs number of traces on synchronous AES cipher

Figure B.26: ranking of all key bytes CPA and DPA attack vs number of traces (50000) on asynchronous AES cipher
Figure B.27: Ranking Partial key bytes synchronous AES using CPA and DPA attack after 10000 traces

Figure B.28: Ranking Partial key bytes asynchronous AES using CPA and DPA attack after 50000 traces