8-11-2020 10:00 AM

Performance Evaluation, Comparison and Improvement of the Hardware Implementations of the Advanced Encryption Standard S-box

Doaa Ashmawy, *The University of Western Ontario*

Supervisor: Reyhani-Masoleh, Arash, *The University of Western Ontario*
A thesis submitted in partial fulfillment of the requirements for the Master of Engineering Science degree in Electrical and Computer Engineering

Recommended Citation

Ashmawy, Doaa, "Performance Evaluation, Comparison and Improvement of the Hardware Implementations of the Advanced Encryption Standard S-box" (2020). *Electronic Thesis and Dissertation Repository*. 7179.
https://ir.lib.uwo.ca/etd/7179

Abstract

The Advanced Encryption Standard (AES) is the most popular algorithm used in symmetric key cryptography. The efficient computation of AES is essential for many computing platforms. The S-box is the only nonlinear transformation step of the AES algorithm. Efficient implementation of the AES S-box is very crucial for AES hardware. The AES S-box could be implemented by using look-up table method or by using finite field arithmetic. The finite field arithmetic design approach to implement the AES S-box is superior in saving the hardware resources. The main objective of this thesis is to evaluate, compare and improve the finite field hardware implementations of the forward, inverse and combined AES S-box in terms of area and/or delay. Both the composite field $GF((2^4)^2)$ and the tower field $GF(((2^2)^2)^2)$ are considered. Our first improvement is the optimization of the input and output linear mappings of the S-box in order to design a more compact circuit. Our second improvement aims at modifying the architecture of the S-box to achieve a higher speed.

We used multiplication of the S-box input by an 8-bit binary field element to optimize the input and output transformation matrices of the S-box as suggested in [54, 80]. A Matlab® search is then conducted to find more compact linear mappings for the S-box. The simulation results show no area improvement for the $GF((2^4)^2)$ S-box [69], however an improvement of 4.25 GEs in STM 65nm technology is realized for the $GF((2^2)^2)^2)$ S-box [70]. We also used the fast S-box architectural improvement in [54], in addition to optimizing and searching the extended linear input mappings to improve the speed of Reyhani et al. fast S-box [69]. The improved fast S-box, Fast 3, is the fastest and most efficient (measured by area $\times$ delay) AES S-box available in the literature, up to our knowledge. We also improved the area and delay of the inversion circuit of the lightweight and fast S-boxes in [69], by slightly modifying the exponentiation block and designing a new subfield inverter block. The improved inversion circuit leads to a more compact and a faster lightweight S-box and it yields a lower area fast S-box.

Moreover, we show that the "tech. XORs" concept proposed by Maximov et al. [54] to estimate the delay of the S-box is not accurate. We show how to use the logical effort method [74] instead to estimate and compare the delay of previous and improved S-boxes, regardless of the CMOS technology library used for the implementation.

We verified all the codes at the RTL level using Mentor Graphics Modelsim®, by comparing against the legitimate S-box outputs. We synthesized the designs using STM 65nm CMOS standard cell library and we used VHDL coding as the design entry method to Synopsys Design Compiler®. The synthesis results confirm the lower area and/or delay of the improved S-box designs and match our space and timing analyses.

Keywords: AES, S-box, Finite field $GF(2^8)$, Multiplicative inverse in $GF(2^8)$, Polynomial basis, Normal basis, Redundant normal basis, Composite field $GF((2^4)^2)$, Tower field $GF(((2^2)^2)^2)$, Logic-minimization heuristics, VHDL, ASIC, Gate Equivalents

Summary for Lay Audience

The Advanced Encryption Standard (AES) is the most popular algorithm used in symmetric key cryptography. The efficient computation of AES is essential for many computing platforms. The S-box is the only nonlinear transformation step of the AES algorithm. Efficient implementation of the AES S-box is very crucial for AES hardware. The AES S-box can be implemented efficiently in hardware using finite field arithmetic. The S-box input mapping is a linear transformation matrix that is used to map all the bytes at the input of the S-box to a different field, where the S-box computations will be done more efficiently. The output mapping is the linear transformation matrix that combines the inverse isomorphic mapping used to re-map the computations back to the S-box original field and the affine transformation step of the S-box.

In this thesis, we evaluated, improved and compared several previous forward, inverse and combined AES S-boxes in terms of implementation area and/or critical path delay. We improved the implementation area of some S-boxes by optimizing the aforementioned input and output mappings. We improved other designs by modifying the architecture of the S-box to reduce the delay. Matlab® is used to search for the most compact linear mappings. We verified all the codes using Computer-Aided Design (CAD) tools and we used Very high speed integrated circuit Hardware Description Language (VHDL) coding as a design entry method to the CAD tool in order to obtain the simulation results. The simulation results obtained from the CAD tool confirm the improved performance of the proposed S-boxes and match our space and timing analyses.

Acknowledgements

First, I would like to thank my supervisor Dr. Arash Reyhani-Masoleh for his continuous guidance, encouragement and advice during the course of this thesis. Moreover, I would like to thak Dr. Mostafa Taha for lending me his experience in CAD tools and MATLAB$^{®}$ simulations. Finally, I would like to thank my family for their unwavering support and patience throughout my degree.

Table of Contents

Abstract **i**

Acknowledgements **iii**

Table of Contents **iv**

List of Tables **vii**

List of Figures **ix**

List of Appendices **xii**

List of Abbreviations **xiii**

1 Introduction **1**

1.1 Advanced Encryption Standard 1

1.1.1 What is AES 1

1.1.2 Literature Review of AES Hardware Implementations 3

1.2 AES S-box Algorithm 5

1.2.1 Forward AES S-box 5

1.2.2 Inverse AES S-box 7

1.2.3 Combined Forward/Inverse AES S-box 7

1.3 Logic Minimization Algorithms 7

1.4 Motivation and Significance 9

1.5 Thesis Outline 10

2 Preliminaries **12**

2.1 Reyhani et al. Composite Field $GF((2^4)^2)$ S-box 12

2.1.1 Composite Field $GF((2^4)^2)$ Construction 12

2.1.2 Inversion over Composite Field $GF((2^4)^2)$ 13

2.2 Reyhani et al. Tower Field $GF(((2^2)^2)^2)$ S-box 14

2.2.1 Tower Field $GF(((2^2)^2)^2)$ Construction 14

2.2.2 Inversion over Tower Field $GF(((2^2)^2)^2)$ 15

2.3 Conclusion 16

3 Implementation Results and Comparisons of Forward, Inverse and Combined AES S-boxes **17**

3.1 Forward, Inverse and Combined AES S-boxes 17

3.2 Behavioural and Structural VHDL Coding 18

3.3 Implementation Results and Comparisons of Forward AES S-boxes 18

3.3.1 Comparisons of the Space and Time Complexities 18

3.3.2 ASIC Synthesis Results and Comparisons 20

3.4 Implementation Results and Comparisons of Inverse AES S-boxes 22

3.4.1 Complexity Analysis of Inverse S-boxes 22
3.4.2 ASIC Synthesis Results and Comparisons 22
3.5 Implementation Results and Comparisons of Combined AES S-boxes 23
3.5.1 Complexity Analysis of Combined S-boxes 23
3.5.2 ASIC Implementation Results 23
3.6 Conclusion 24

4 Improved $GF((2^4)^2)$ Forward AES S-box 25
4.1 Introduction 25
4.2 Improved Lightweight and Fast AES S-boxes 25
4.2.1 Improved Exponentiation 25
4.2.2 New Subfield Inverter 27
4.2.3 Space and Time Complexity Analyses 28
4.2.4 ASIC Synthesis Results and Comparisons 28
4.3 Additional Transformation Matrices 30
4.3.1 Multiplying by a Binary Field Element η 30
4.3.2 Modified AES S-box Architecture 33
4.3.3 Matlab® Results 33
4.4 Improved Fast AES S-box Architecture 34
4.4.1 The Modified Fast S-box, Fast 1 35
4.4.2 The Modified Fast S-box, Fast 2 39
4.4.3 Complexity Analysis 42
4.4.4 Comparisons of the Space and Time Complexities 42
4.4.5 Calculating Delay in CMOS Circuits 43
4.4.6 ASIC Synthesis Results and Comparison 46
4.5 Additional Extended Transformation Matrices 47
4.5.1 Complexity Analysis 51
4.5.2 Comparisons of the Space and Time Complexities 51
4.5.3 ASIC Synthesis Results and Comparison 51
4.6 Conclusion 53

5 Improved $GF(((2^2)^2)^2)$ Forward AES S-box 55
5.1 Introduction 55
5.2 Architecture of the Improved AES S-box 55
5.3 Input and Output Mappings of the Improved AES S-box 56
5.3.1 Input Mapping of the Improved AES S-box 57
5.3.2 Output Mapping of the Improved AES S-box 58
5.4 Modified Inversion over $GF(((2^2)^2)^2)$ 59
5.4.1 Multiplication over $GF((2^2)^2)$ 59
5.4.2 Squaring with Scaling 60
5.4.3 Modified Exponentiation Computation 60
5.4.4 Modified Subfield Inversion over $GF((2^2)^2)$ 62
5.4.5 Modified Output Multipliers 63
5.5 Implementation Results and Comparisons 65
5.5.1 Complexity Analysis of the Improved AES S-box 65
5.5.2 ASIC Implementation Results 65
5.5.2.1 Individual Blocks 65
5.5.2.2 Overall Design 66
5.6 Conclusion 67

6 Summary and Future Work **68**

6.1 Thesis Summary 68

6.2 Future Work 68

Appendix A Additional Figures for Chapter 3 **77**

Appendix B Additional Figures for Chapter 4 **86**

Curriculum Vitae **95**

List of Tables

3.1 Space complexity comparison of some forward S-boxes. 20
3.2 Time complexity comparison for different S-boxes. 21
3.3 ASIC comparisons of the S-Box architectures. 21
3.4 Gate count of some inverse S-box circuits. 22
3.5 ASIC Implementation results for some Inverse S-box. 23
3.6 Complexity comparison of different combined S-box designs. 23
3.7 ASIC Implementation results of different combined S-boxes. 24

4.1 The truth table of the inverter over $GF(2^4)$ generated by the ONB-I [69]. 28
4.2 Space and time complexities for different blocks and the $GF((2^4)^2)$ inversion of the entire improved S-boxes. 29
4.3 ASIC synthesis results for the improved exponentiation and the improved subfield inverter. 29
4.4 ASIC synthesis results for the two S-boxes in [69] using the improved exponentiation and improved subfield inverter. 29
4.5 The number of XOR2 gates and the maximum delay of specific $\mathbf{T}_{out}$ matrices. . . . 35
4.6 Fast 1 implementation of transformations. 38
4.7 Fast 2 implementation of transformations. 42
4.8 Space and time complexities of the modified S-box architecture Fast 1 and Fast 2. . 43
4.9 Space complexity comparison of different fast S-boxes. 43
4.10 Time complexity comparison for different fast S-boxes. 43
4.11 The input capacitance, C_{in} of some basic gates in Samsung's STD90/MDL90 0.35 μm library normalized to SL [47]. 44
4.12 The parasitic delay of some basic gates normalized to p_{inv} [74]. 45
4.13 The logical effort of some basic gates [74]. 45
4.14 Normalizd delay values for different fast S-boxes using the method of logical effort. 46
4.15 ASIC comparisons of the fast S-Box architectures. 47
4.16 Testing $\mathbf{T}_{out}$ entries for 1's in order to determine the 4 signals that will be used to extend $\mathbf{T}_{in}$. i is the row number, $1 \leq i \leq 8$. 50
4.17 Fast implementation of transformations in (4.41) for $\nu = 1000$, Generator #1 and $\eta = 48$ (Fast 3 design). 51
4.18 Space and time complexities of the modified fast S-box architecture, Fast 3. 52
4.19 Space complexity comparison of different fast S-boxes. 52
4.20 Time complexity comparison for different fast S-boxes. 52
4.21 ASIC comparisons of the S-Box architectures 53

5.1 The number of XOR2 gates and the maximum delay results for some $\mathbf{T}_{out_{new}}$ matrices. 57
5.2 Equations used to implement the input mappings of the improved forward S-box. . 58
5.3 Equations used to implement the proposed output mappings of the improved forward-only S-box. 59
5.4 The truth table of the inverter over $GF((2^2)^2)$ using the field represented by $\{\nu = [0010] = \omega\alpha^4, N = [10] = \omega\}$. 62

5.5 Complexity comparison of the improved and the original forward S-box designs. . 65
5.6 ASIC synthesis results for the three blocks of the inversion and the entire inversion over the tower field at STM 65nm technology. 66
5.7 ASIC Implementation results for the improved forward S-box. 66

List of Figures

1.1 The AES cipher. 2
1.2 The encryption process of the AES-128 algorithm. 3
1.3 The AES S-box transformation step. 6
1.4 The forward AES S-box using composite/tower field inversion. 7
1.5 The inverse AES S-box using composite/tower field inversion. 7
1.6 The combined forward/inverse AES S-box. 8

2.1 The architecture of the S-box using $GF(((2^4)^2)$ in the NB representation. 13
2.2 Reyhani et al. composite field S-box. 14
2.3 The architecture of the S-box using $GF(((2^2)^2)^2)$ in the NB representation. 14
2.4 Reyhani et al. tower field S-box. 16

4.1 The improved exponentiation computation block for the S-box architecture. 26
4.2 The modified subfield inverter circuit. 28
4.3 Input and output transformations used in the S-box of [69]. 30
4.4 The modified input and output transformations. 33
4.5 The S-box architecture in [69]. 35
4.6 The modified fast S-box architectures, Fast 1. 36
4.7 The modified fast S-box architectures, Fast 2. 39
4.8 Influence of the load on the normalized delay values of the different fast S-boxes. . 46
4.9 Area in GEs of the proposed Fast 2 S-box as compared to previous work [69, 54] at different input delay constraints for the STM 65nm library. 47
4.10 Improved fast S-box architecture. 49
4.11 Area in GEs of the proposed Fast 3 and Fast 2 S-boxes as compared to previous work [69, 54] at different input delay constraints for the STM 65nm library. 54

5.1 The overall architecture of the improved forward S-box using $GF(((2^2)^2)^2)$ in the NB representation. 55
5.2 The overall architecture of the improved forward S-box using $GF(((2^2)^2)^2)$ in the NB representation. 56
5.3 The architecture of the modified exponentiation block. 61
5.4 The modified subfield inverter block. 62
5.5 Area in GEs of the improved forward S-box circuit as compared to previous work [70] at different input delay constraints targeting the STM 65nm library. 66

A.1 (a) The number of XOR2 gates required to implement $\mathbf{T}_{in}$ as a function of η for the case of $\nu = 1000$, Generator #1.(b) The maximum delay of $\mathbf{T}_{in}$ as a function of η for the case of $\nu = 1000$, Generator #1. 78
A.2 (a) The number of XOR2 gates required to implement $\mathbf{T}_{in}$ as a function of η for the case of $\nu = 1000$, Generator #2.(b) The maximum delay of $\mathbf{T}_{in}$ as a function of η for the case of $\nu = 1000$, Generator #2. 78

A.3 (a) The number of XOR2 gates required to implement $\mathbf{T}_{in}$ as a function of η for the case of $v = 1000$, Generator #3.(b) The maximum delay of $\mathbf{T}_{in}$ as a function of η for the case of $v = 1000$, Generator #3. 79

A.4 (a) The number of XOR2 gates required to implement $\mathbf{T}_{in}$ as a function of η for the case of $v = 1000$, Generator #4.(b) The maximum delay of $\mathbf{T}_{in}$ as a function of η for the case of $v = 1000$, Generator #4. 79

A.5 (a) The number of XOR2 gates required to implement $\mathbf{T}_{in}$ as a function of η for the case of $v = 0111$, Generator #1.(b) The maximum delay of $\mathbf{T}_{in}$ as a function of η for the case of $v = 0111$, Generator #1. 80

A.6 (a) The number of XOR2 gates required to implement $\mathbf{T}_{in}$ as a function of η for the case of $v = 0111$, Generator #2.(b) The maximum delay of $\mathbf{T}_{in}$ as a function of η for the case of $v = 0111$, Generator #2. 80

A.7 (a) The number of XOR2 gates required to implement $\mathbf{T}_{in}$ as a function of η for the case of $v = 0111$, Generator #3.(b) The maximum delay of $\mathbf{T}_{in}$ as a function of η for the case of $v = 0111$, Generator #3. 81

A.8 (a) The number of XOR2 gates required to implement $\mathbf{T}_{in}$ as a function of η for the case of $v = 0111$, Generator #4.(b) The maximum delay of $\mathbf{T}_{in}$ as a function of η for the case of $v = 0111$, Generator #4. 81

A.9 (a) The number of XOR2 gates required to implement $\mathbf{T}'_{in}$ as a function of η for the case of $v = 1000$, Generator #1.(b) The maximum delay of $\mathbf{T}'_{in}$ as a function of η for the case of $v = 1000$, Generator #1. 82

A.10 (a) The number of XOR2 gates required to implement $\mathbf{T}'_{in}$ as a function of η for the case of $v = 1000$, Generator #2.(b) The maximum delay of $\mathbf{T}'_{in}$ as a function of η for the case of $v = 1000$, Generator #2. 82

A.11 (a) The number of XOR2 gates required to implement $\mathbf{T}'_{in}$ as a function of η for the case of $v = 1000$, Generator #3.(b) The maximum delay of $\mathbf{T}'_{in}$ as a function of η for the case of $v = 1000$, Generator #3. 83

A.12 (a) The number of XOR2 gates required to implement $\mathbf{T}'_{in}$ as a function of η for the case of $v = 1000$, Generator #4.(b) The maximum delay of $\mathbf{T}'_{in}$ as a function of η for the case of $v = 1000$, Generator #4. 83

A.13 (a) The number of XOR2 gates required to implement $\mathbf{T}'_{in}$ as a function of η for the case of $v = 0111$, Generator #1.(b) The maximum delay of $\mathbf{T}'_{in}$ as a function of η for the case of $v = 0111$, Generator #1. 84

A.14 (a) The number of XOR2 gates required to implement $\mathbf{T}'_{in}$ as a function of η for the case of $v = 0111$, Generator #2.(b) The maximum delay of $\mathbf{T}'_{in}$ as a function of η for the case of $v = 0111$, Generator #2. 84

A.15 (a) The number of XOR2 gates required to implement $\mathbf{T}'_{in}$ as a function of η for the case of $v = 0111$, Generator #3.(b) The maximum delay of $\mathbf{T}'_{in}$ as a function of η for the case of $v = 0111$, Generator #3. 85

A.16 (a) The number of XOR2 gates required to implement $\mathbf{T}'_{in}$ as a function of η for the case of $v = 0111$, Generator #4.(b) The maximum delay of $\mathbf{T}'_{in}$ as a function of η for the case of $v = 0111$, Generator #4. 85

B.1 (a) The number of XOR2 gates required to implement $\mathbf{T}_{in_{new}}$ as a function of η for the case of $\{v = [0010], N = [01]\}$, Generator #1.(b) The maximum delay of $\mathbf{T}_{in_{new}}$ as a function of η for the case of $\{v = [0010], N = [01]\}$, Generator #1. 87

B.2 (a)The number of XOR2 gates required to implement $\mathbf{T}_{in_{new}}$ as a function of η for the case of $\{v = [0010], N = [01]\}$, Generator #2.(b) The maximum delay of $\mathbf{T}_{in_{new}}$ as a function of η for the case of $\{v = [0010], N = [01]\}$, Generator #2. 87
B.3 (a) The number of XOR2 gates required to implement $\mathbf{T}_{in_{new}}$ as a function of η for the case of $\{v = [0010], N = [01]\}$, Generator #3.(b) The maximum delay of $\mathbf{T}_{in_{new}}$ as a function of η for the case of $\{v = [0010], N = [01]\}$, Generator #3. 88
B.4 (a) The number of XOR2 gates required to implement $\mathbf{T}_{in_{new}}$ as a function of η for the case of $\{v = [0010], N = [01]\}$, Generator #4.(b) The maximum delay of $\mathbf{T}_{in_{new}}$ as a function of η for the case of $\{v = [0010], N = [01]\}$, Generator #4. 88
B.5 (a) The number of XOR2 gates required to implement $\mathbf{T}_{in_{new}}$ as a function of η for the case of $\{v = [0010], N = [10]\}$, Generator #1.(b) The maximum delay of $\mathbf{T}_{in_{new}}$ as a function of η for the case of $\{v = [0010], N = [10]\}$, Generator #1. 89
B.6 (a) The number of XOR2 gates required to implement $\mathbf{T}_{in_{new}}$ as a function of η for the case of $\{v = [0010], N = [10]\}$, Generator #2.(b) The maximum delay of $\mathbf{T}_{in_{new}}$ as a function of η for the case of $\{v = [0010], N = [10]\}$, Generator #2. 89
B.7 (a) The number of XOR2 gates required to implement $\mathbf{T}_{in_{new}}$ as a function of η for the case of $\{v = [0010], N = [10]\}$, Generator #3.(b) The maximum delay of $\mathbf{T}_{in_{new}}$ as a function of η for the case of $\{v = [0010], N = [10]\}$, Generator #3. 90
B.8 (a) The number of XOR2 gates required to implement $\mathbf{T}_{in_{new}}$ as a function of η for the case of $\{v = [0010], N = [10]\}$, Generator #4.(b) The maximum delay of $\mathbf{T}_{in_{new}}$ as a function of η for the case of $\{v = [0010], N = [10]\}$, Generator #4. 90
B.9 (a) The number of XOR2 gates required to implement $\mathbf{T}_{in_{new}}$ as a function of η for the case of $\{v = [0111], N = [01]\}$, Generator #1.(b) The maximum delay of $\mathbf{T}_{in_{new}}$ as a function of η for the case of $\{v = [0111], N = [01]\}$, Generator #1. 91
B.10 (a) The number of XOR2 gates required to implement $\mathbf{T}_{in_{new}}$ as a function of η for the case of $\{v = [0111], N = [01]\}$, Generator #2.(b) The maximum delay of $\mathbf{T}_{in_{new}}$ as a function of η for the case of $\{v = [0111], N = [01]\}$, Generator #2. 91
B.11 (a) The number of XOR2 gates required to implement $\mathbf{T}_{in_{new}}$ as a function of η for the case of $\{v = [0111], N = [01]\}$, Generator #3.(b) The maximum delay of $\mathbf{T}_{in_{new}}$ as a function of η for the case of $\{v = [0111], N = [01]\}$, Generator #3. 92
B.12 (a) The number of XOR2 gates required to implement $\mathbf{T}_{in_{new}}$ as a function of η for the case of $\{v = [0111], N = [01]\}$, Generator #4.(b) The maximum delay of $\mathbf{T}_{in_{new}}$ as a function of η for the case of $\{v = [0111], N = [01]\}$, Generator #4. 92
B.13 (a) The number of XOR2 gates required to implement $\mathbf{T}_{in_{new}}$ as a function of η for the case of $\{v = [0111], N = [10]\}$, Generator #1.(b) The maximum delay of $\mathbf{T}_{in_{new}}$ as a function of η for the case of $\{v = [0111], N = [10]\}$, Generator #1. 93
B.14 (a) The number of XOR2 gates required to implement $\mathbf{T}_{in_{new}}$ as a function of η for the case of $\{v = [0111], N = [10]\}$, Generator #2.(b) The maximum delay of $\mathbf{T}_{in_{new}}$ as a function of η for the case of $\{v = [0111], N = [10]\}$, Generator #2. 93
B.15 (a) The number of XOR2 gates required to implement $\mathbf{T}_{in_{new}}$ as a function of η for the case of $\{v = [0111], N = [10]\}$, Generator #3.(b) The maximum delay of $\mathbf{T}_{in_{new}}$ as a function of η for the case of $\{v = [0111], N = [10]\}$, Generator #3. 94
B.16 (a) The number of XOR2 gates required to implement $\mathbf{T}_{in_{new}}$ as a function of η for the case of $\{v = [0111], N = [10]\}$, Generator #4.(b) The maximum delay of $\mathbf{T}_{in_{new}}$ as a function of η for the case of $\{v = [0111], N = [10]\}$, Generator #4. 94

List of Appendices

Appendix A Additional Figures for Chapter 3 77
Appendix B Additional Figures for Chapter 4 86

List of Abbreviations

AES	Advanced Encryption Standard
ASIC	Application-Specific Integrated Circuit
CAD	Computer-Aided Design
CBC	Cipher Block Chaining
CPD	Critical Path Delay
DES	Data Encryption Standard
ECB	Electronic Code Book
GEs	Gate Equivalents
IoT	Internet of Things
ISE	Instruction Set Extension
LUT	Look-Up Table
LWC	Lightweight Cryptography
MSB	Most Significant Bit
NB	Normal Basis
NIST	National Institute of Standards and Technology
PB	Polynomial Basis
PQC	Post-quantum Cryptogrphy
RFID	Radio-Frequency IDentification
RNB	Redundant Normal Basis
ROM	Read-Only-Memory
RTL	Register-Transfer Level
SPN	Substitution-Permutation Network
VHDL	Very-high-speed integrated circuit Hardware Description Language

Chapter 1

Introduction

The Advanced Encryption Standard (AES) is the most popular algorithm used in symmetric key cryptography. The efficient computation of AES is essential for many computing platforms. The S-box is the only nonlinear transformation step of the AES algorithm. Efficient implementation of the AES S-box is very crucial for AES hardware. The AES S-box could be implemented by using Look-Up Table (LUT) method which stores all 256 predefined 8-bit values of the S-box in a Read-Only-Memory (ROM). The S-box could also be implemented using composite/tower field arithmetic. The finite field arithmetic design approach to implement the AES S-box is superior to LUTs in saving the hardware resources. The main objective of this research is to evaluate, compare and improve the composite/tower field hardware implementations of the AES S-box.

In this Chapter, we will give a brief introduction to the AES algorithm and its only non-linear step, the S-box. We will also review the logic minimization algorithms used to minimize the implementation area of linear mapping circuits used at the input and output of the S-box. This is an essential step before stating the motivation and significance of the thesis.

1.1 Advanced Encryption Standard

In this section, we will explain the AES algorithm used for encryption/decryption. A literature review of AES hardware implementations is also provided.

1.1.1 What is AES

The Advanced Encryption Standard (AES) [31], also called the Rijndael algorithm, is a symmetric block cipher algorithm that was adopted by the National Institute of Standards and Technology (NIST) as a replacement of the Data Encryption Standard (DES) algorithm back in 2001. AES is essentially a subset of the Rijndael algorithm [25] which was the winner of a five-year competition among fifteen block cipher algorithms. The AES can be implemented in software and/or hardware to encrypt/decrypt electronic data. It works as a Substitution-Permutation Network (SPN) with four main operations: SubBytes, ShiftRows, MixColumns, and AddRoundKey. The SubBytes operation uses the Rijndael S-box which is the main non-linear substitution step of AES. The AES algorithm is a symmetric encryption algorithm, meaning encryption and decryption are performed by mainly the same steps. It is a block cipher, where the data is encrypted/decrypted in blocks of 128 bits. The original Rijndael algorithm allows other block sizes, but the AES standard only permits 128-bit blocks. The AES cipher is shown in Figure 1.1. Here, the input data block is arranged as 4×4 matrix of bytes, called the state. Each data block is processed by several transformation rounds, where each round involves four steps. Three different key sizes are allowed according to the level of security needed: 128 bits, 192 bits, or 256 bits, and the corresponding number of transformation rounds for each is 10 rounds, 12 rounds, or 14 rounds, respectively. The "AES-128",

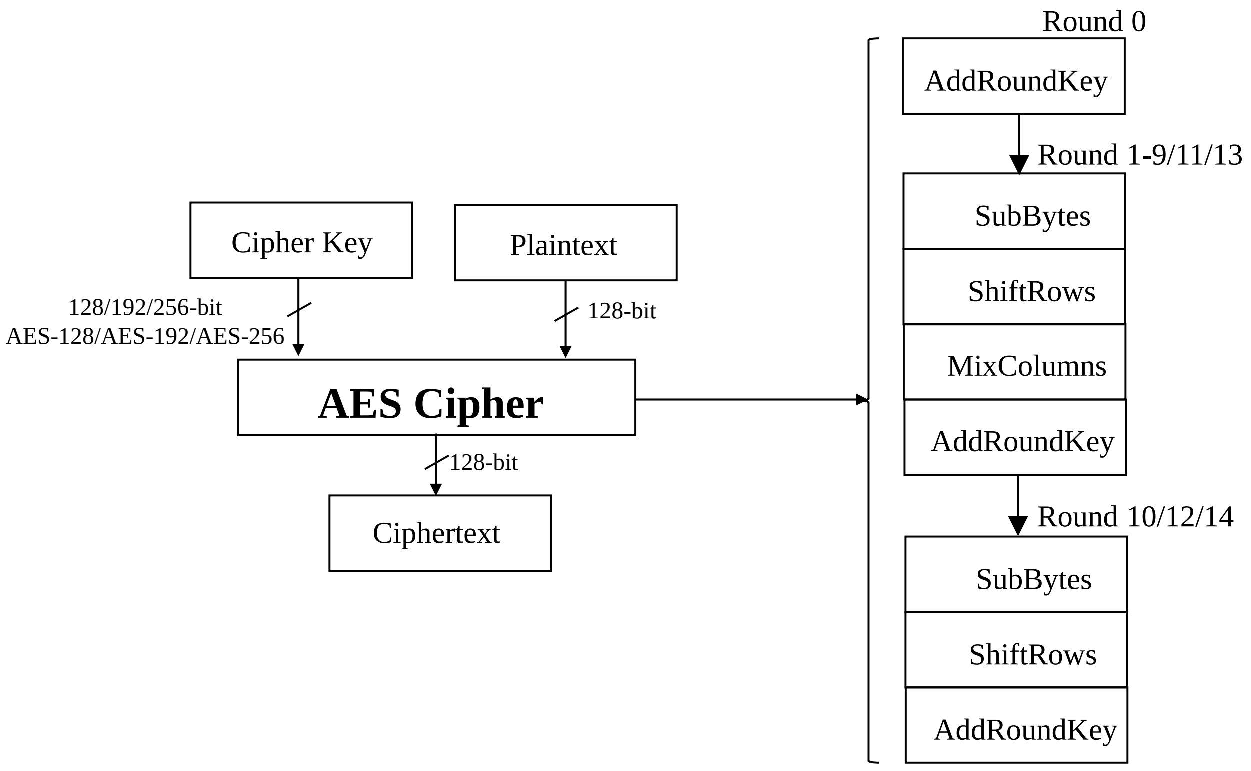


Figure 1.1: The AES cipher.

"AES-192" and "AES-256" refer to the 128 bits, 192 bits and 256 bits cryptographic key lengths, respectively. The original key is used to compute a series of Round Keys, one for each of these rounds in a routine called key expansion.

Figure 1.2 shows the encryption process of the AES-128 algorithm. The length of the input block, the output block and the state is 128 bits. The length of the key is 128 bits and the number of round functions is 10. The state is a 4×4 matrix of bytes, or 4 columns. Each column is a 32-bit word or alternatively 4 bytes. The three steps, namely ShiftRows, MixColumns, and AddRoundKey, are linear such that the output 128-bit block for such steps is a linear combination (bit-wise, modulo 2) of the outputs for each separate input bit. The SubBytes (S-box) operation is the only nonlinear step of the AES algorithm, where each input byte is replaced by the result of applying the S-box function to that byte. In the key schedule routine, 10 sets of round keys are calculated from the initial key.

In the beginning of the encryption process, the AES-128 cipher performs an initial round, where the initial key is added to the state. After the initial round, there are 9 identical rounds then a slightly different final round. For rounds 1 to 9, the round function is identical and is implemented as follows:

1. **AddRoundKey**: A bit-wise XOR operation is used to add the round key to the state.

2. **SybBytes**: Two steps, the first step involves finding the multiplicative inverse of the input byte in $GF(2^8)$, except for the element {00}, which is mapped to itself. We use two hexadecimal numbers to represent a byte, e.g. the element {00} refers an input byte of value zero. The second step is applying an affine transformation, which is a multiplication by a matrix **M** followed by the addition of a constant vector **h**.

3. **ShiftRows**: Cyclically shift the rows of the state to the left by a different number of bytes, except for the first row which is not shifted.

4. **MixColumns**: Independently mix the columns of the state into new columns. This is done by considering each column as a four-term polynomial over $GF(2^8)$ then multiplying it mod $(x^4 + 1)$ with the fixed polynomial $a(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\}$.

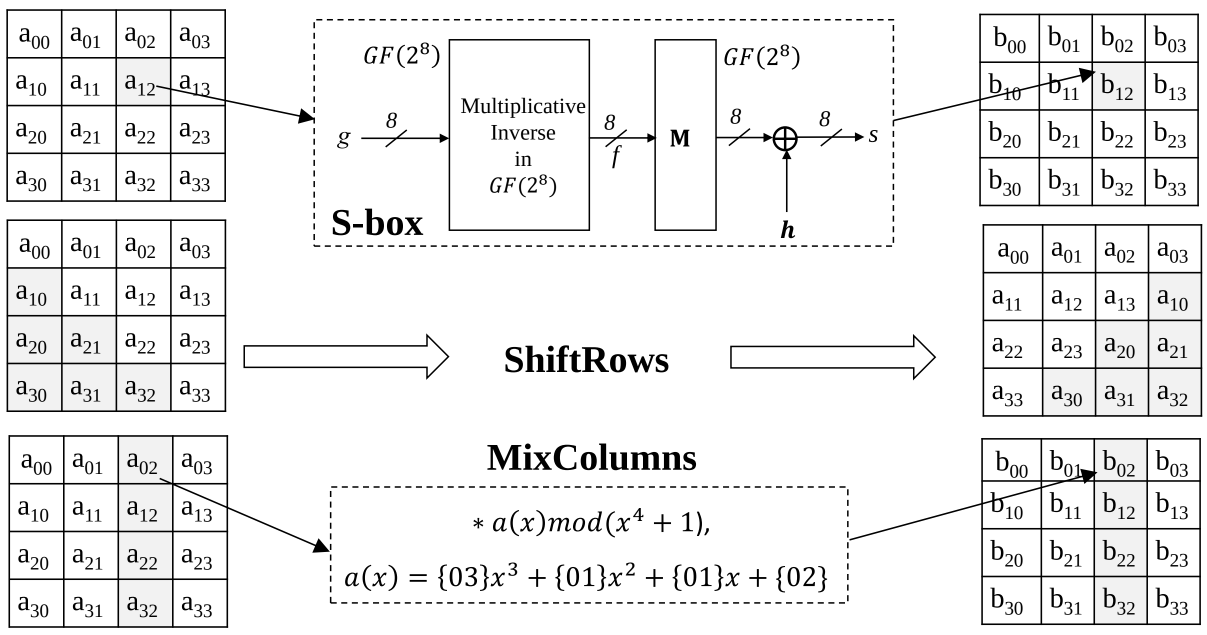

S-box
GF(2⁸)
Multiplicative Inverse in GF(2⁸)
M
g
f
h
s
8
ShiftRows
MixColumns
∗ a(x)mod(x⁴ + 1),
a(x) = {03}x³ + {01}x² + {01}x + {02}

Figure 1.2: The encryption process of the AES-128 algorithm.

For the AES-128 inverse cipher, the AES-128 cipher operations are inverted and implemented in reverse order. The AddRoundKey is its own inverse and the round keys are used in reverse order from the encryption process. The three other transformations needed for decryption in the inverse cipher are as follows:

1. **InvSubBytes**: Two steps, first apply the inverse affine transformation then find the multiplicative inverse in $GF(2^8)$.

2. **InvShiftRows**: Shift the last three rows of the state in the opposite direction (i.e. to the right) from the ShiftRows. The first row stays the same.

3. **InvMixColumns**: Each column is considered as a four-term polynomial over $GF(2^8)$ and multiplied mod $(x^4 + 1)$ with the fixed polynomial $a^{-1}(x) = \{0b\}x^3 + \{0d\}x^2 + \{09\}x + \{0e\}$.

1.1.2 Literature Review of AES Hardware Implementations

The implementations of the AES algorithm are based on the use of hardware only, software only or a hybrid approach using an Instruction Set Extension (ISE). In a hardware only AES implementation, the AES algorithm is executed by using a dedicated hardware module (e.g. an ASIC chip). In a software only implementation, the execution of the AES is performed by software instructions (e.g. an assembly language AES implementation on an 8-bit microcontroller). The hybrid ISE approach uses both the hardware and software to implement the AES algorithm (e.g. use both hardware on the host core and new instructions added to the base Instruction Set Architecture (ISA) to execute AES) [49]. The focus of this research will be the hardware only AES implementations.

Since its introduction in 2001, many articles about AES hardware implementations that target area [57, 7, 8, 37, 3], speed [52, 51, 80] or power consumption [58, 29, 34, 77] have been published. In 2001, Satoh introduced a compact AES encrypt/decrypt hardware architecture while optimizing the S-box [72]. The S-box is optimized for low area by using the tower field $GF(((2^2)^2)^2)$ over polynomial basis to compute the multiplicative inverse of the S-box input byte. Different throughputs are achieved by varying the number of S-boxes used per round. The highest throughput is achieved by using 20 S-boxes per round, where 16 S-boxes are used for calculating the round function and 4 S-boxes are deployed by the key expansion routine to determine the separate round keys. Same as Satoh article, the article [56] also used the tower field $GF(((2^2)^2)^2)$ to perform the Galois field inverse of the S-box.

To reduce the area of implementation, 8-bit AES designs that uses either a single S-box or two S-boxes are deployed. In the first case, where only one S-box is adopted, the S-box operation is interleaved between the round function and the key expansion, thus increasing the number of clock cycles/block and decreasing the throughput. To increase the throughput, two S-boxes could be utilized, the first S-box would be used for the normal round operation and the other S-box for the key expansion routine. The grain of sand 8-bit AES encrypt/decrypt compact core by Feldhofer et al. [30] introduced a low-resource (i.e. low area and low power consumption) hardware implementation. It occupies an area of 3400 Gate Equivalents (GEs) and supports both encryption and decryption. The number of clock cycles that are required to encrypt one 128 bits block of data is 1032 cycles and the maximum clock frequency is 80 MHz. The throughput can be computed as $128\,bits \div \frac{1032}{80\times10^6}\,s = 9.9$ Mbps, which is relatively low. The article mentioned that the decryption performs nearly the same as encryption. A single S-box is used for SubBytes and InvSuBytes operations as well as for the key expansion. The S-box used in the implementation is based on the finite field $GF(2^8)$ arithmetic operations rather than Look-Up Table (LUT) method and is adopted from [85], where a pipeline register is added to reduce the critical path of the S-box therefore increasing the clock frequency. This comes with the adverse effect of an increased number of clock cycles necessary for the encryption process.

Canright introduced his compact combined forward/inverse AES S-box in 2005 [20, 21]. It uses subfields of 4 bits and of 2 bits to calculate the multiplicative inverse of the S-box input byte. He also examined several choices of bases for each subfield, including the normal basis one as [72]. A very compact encrypt/decrypt AES algorithm based on Canright combined S-box was later published in [22]. It reduces the number of bit operations for encryption, by combining the affine transformation of the forward S-box and the Galois multiplications by constants of the MixColumns step. For decryption, it integrates the inverse affine transformation of the inverse S-box and the InvMixColumns Galois scalings step.

The 8-bit AES encryption circuit in [34] adopted two S-boxes, one S-box is used for round computation and the other S-box is used for key expansion. As a result, no interleaving of operation is required and the number of clock cycles required for ciphering one block of data is set theoretically to 160 cycles. Because of its low energy and relatively small area (3100 GEs), [86] used the AES design in [34] to build an energy efficient 8-bit AES encryption core that can be used to secure Internet of Things (IoT) networks.

The AES Encrypt/Decrypt design in [51] minimized the circuit area by optimizing the encryption and decryption composite field polynomials, separately. The AES computations are performed in the $GF((2^4)^2)$ composite field, where equivalent composite field representations are derived for the Affine, InvAffine, MixColumns and InvMixColumns steps.

The compact 8-bit AES encryption core in [57] used only one S-box for both round computation and key expansion, leading to a reduced area of 2400 GEs. As a result of sharing the S-box between the round function and the key schedule module, the number of clock cycles needed to encrypt one 128 bits block of data is increased to 226 clock cycles. Due to the increased number of clock cycles/block, the energy efficiency of this design is not optimum. Both area and energy are important criteria in determining the energy efficiency of a certain implementation [76]. Despite having a larger area, Hamalainen design [34] is more energy efficient than Moradi core [57] as it is more than 40% faster [86].

The AES encryption only core in [57] was later refined to provide both encryption and decryption functionalities in Atomic AES [7], which was followed by Atomic AES v2.0 [8] to further refine the area of implementation. In 2017, the same architecture was modified into a serialized one-bit Encryption/decryption AES circuit in [37]. It has a very compact area, however it needs 1776 clock cycles for encryption and 2512 clock cycles for decryption, which drastically increase the latency.

A very compact 8-bit both encryption and decryption AES-128/192/256 hardware core was introduced in [3]. The serialized (i.e. 8-bit) AES-192 and AES-256 circuits are of greater importance for lightweight cryptography applications in the next era of quantum computers due to their increased security levels.

Recently, [26] introduced an energy-efficient 8-bit AES architecture suitable for resource-constrained applications. The circuit reduces the data movement, hence the energy, by directly multiplexing the bytes into the data path and updating the state register only once per round using a technique called register renaming. The energy efficiency is increased because there is no need to shift the bytes through the state register, i.e. it reduced the data movement. Also there is no storage for intermediate results between the AES rounds, which saves the area. The energy-efficient S-box in [12] was utilized in this architecture due to its lower glitching, hence lower energy [5].

An important aspect of the AES is the mode of operation. There are several different modes in which AES can be used [28]. e.g. the Cipher Block Chaining (CBC) works in a feedback mode of operation, where the result of encrypting one block is used to encrypt the next block. The CBC mode can not be pipelined for improved speed because of the feedback nature of operation. The Electronic Code Book (ECB) and the Counter (CTR) modes can be pipelined as they do not require feedback blocks from previous ciphers.

In the following section, we will explain the S-box step of the AES algorithm.

1.2 AES S-box Algorithm

The S-box is a nonlinear function that involves finding the multiplicative inverse of the input byte in the finite field $GF(2^8)$, except for the input zero which is mapped to itself [31]. The S-box works independently on each byte of the input. The S-box could be implemented in hardware using the Look-Up Table (LUT) method or by using composite field arithmetic. In the LUT method, the input byte is used as an index to a table (256 bytes table) stored in memory to find the output byte. This method is fast but allocates a large amount of hardware resources to store, address and fetch the output bytes. A more efficient hardware implementation could be realized by using composite/tower field arithmetic to compute the inverse of the S-box input byte [72, 85, 20]. By utilizing the finite field arithmetic to compute the inverse of the S-box input byte, the inversion circuit could be shared for decryption such that only the inverse affine transformation has to be computed for the inverse S-box.

The S-boxes implemented using the finite field arithmetic will be evaluated, compared and improved in this thesis.

According to the type of data processing needed, either encryption or decryption, a forward AES S-box or an inverse AES S-box will be used. A combined forward/inverse AES S-box is also used for AES encryption/decryption engines. Each will be explained in the following sections.

1.2.1 Forward AES S-box

The forward S-box depends on performing an inversion over the $GF(2^8)$ field of AES, as defined over the irreducible polynomial $q(x) = x^8 + x^4 + x^3 + x + 1$, followed by an affine transformation and addition with a constant as shown in Figure 1.3. If the input to the S-box is g, where g is an element in $GF(2^8)$ generated by the irreducible polynomial $q(x)$. Let α be a root of $q(x)$, then $g = (g_7, \cdots, g_1, g_0) \in GF(2^8)$ is represented in polynomial basis representation as $g = \sum_{i=0}^{7} g_i\alpha^i$, $g_i \in GF(2)$, $0 \leq i \leq 7$, where g_is are the coordinates of $g \in GF(2^8)$. For convenience, these coordinates will be denoted in vector notation as $\mathbf{g} = [g_7, \cdots, g_1, g_0]^{tr}$, where tr denotes the

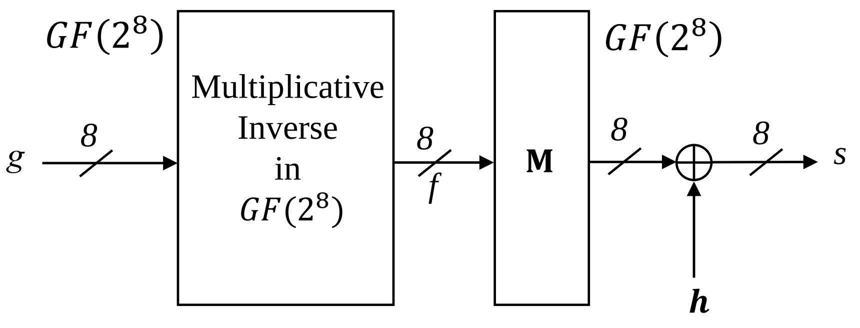


Figure 1.3: The AES S-box transformation step.

transposition.

Let $f = (f_7, \cdots, f_1, f_0) \in GF(2^8)$ be the multiplicative inverse (or inverse in short) of the non-zero S-box input, i.e., $f \times g = 1$, where $g \neq 0$. Then, the first step in the S-box computation is to find the inverse $f = g^{-1}$ for $g \neq 0$. For $g = 0$, $f = 0$. Let $s = (s_7, \cdots, s_1, s_0) \in GF(2^8)$ be the S-box output. Then in the second step of the S-box computation, the *affine transformation* is calculated as:

$$\mathbf{s} = \mathbf{Mf} \oplus \mathbf{h} \tag{1.1}$$

$$\mathbf{s} = \begin{bmatrix} s_7 \\ s_6 \\ s_5 \\ s_4 \\ s_3 \\ s_2 \\ s_1 \\ s_0 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} f_7 \\ f_6 \\ f_5 \\ f_4 \\ f_3 \\ f_2 \\ f_1 \\ f_0 \end{bmatrix} \oplus \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix} \tag{1.2}$$

where $\oplus$ is the modulo 2 addition (XOR), bit #7 is the most significant bit, and $\mathbf{M}$ is the affine transformation matrix.

An efficient hardware implementation of the S-box can be realized by using finite field arithmetic as shown in Figure 1.4. The finite field arithmetic is used to compute the multiplicative inverse of the S-box input byte. To lower the cost of finding the multiplicative inverse in $GF(2^8)$, the following 3 steps are often adopted:

1. Map all elements of $GF(2^8)$ at the input of the S-box to a composite field $GF((2^4)^2)$ or to a tower field $GF(((2^2)^2)^2)$, using an isomorphism function $\mathbf{X}^{-1}$.
2. Compute the multiplicative inverse over $GF((2^4)^2)$ or over $GF(((2^2)^2)^2)$.
3. Re-map the computation results to $GF(2^8)$, using the inverse isomorphism function $\mathbf{X}$.

As shown in Figure 1.4, we denote the matrix that is used to map the 256 Galois field elements at the input of the S-box, i.e used for input isomorphic mapping of the S-box, as $\mathbf{X}^{-1}$. The matrix that is used to convert in the other way around is denoted as $\mathbf{X}$. Therefore, the S-box input vector $\mathbf{g} = [g_7, \cdots, g_1, g_0]^{tr}$ is converted to the corresponding vector representation in the corresponding field as $\mathbf{i} = \mathbf{X}^{-1} \times \mathbf{g}$. As a result, the output of the forward S-box can be computed using (1.1) as

$$\mathbf{s} = \mathbf{MX} \times \mathbf{o} \oplus \mathbf{h} \tag{1.3}$$

where $\mathbf{f} = \mathbf{X} \times \mathbf{o}$ is used to convert the output of the inversion circuit, $o = i^{-1} \in GF((2^4)^2)/GF(((2^2)^2)^2)$ back to the binary field $GF(2^8)$.

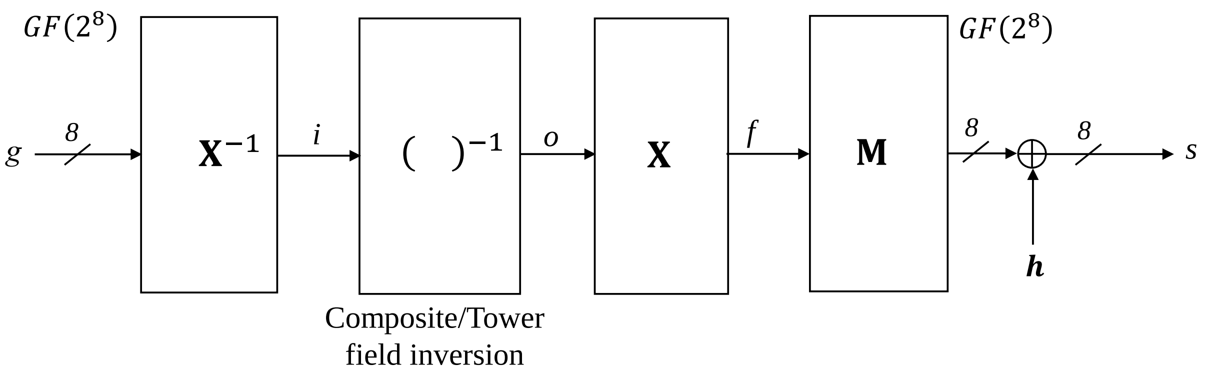


Figure 1.4: The forward AES S-box using composite/tower field inversion.

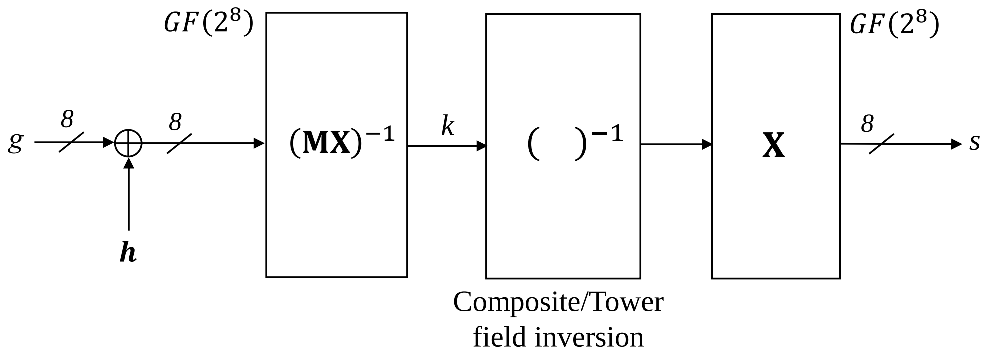


Figure 1.5: The inverse AES S-box using composite/tower field inversion.

1.2.2 Inverse AES S-box

The inverse S-box performs the reverse of the forward S-box operation. Let **g** be the input of the inverse S-box. One can find out that the input mapping of the inverse S-box is the reverse operation of the output mapping of the forward S-box as shown in Figure 1.5. In other words the input mapping of the inverse S-box generates:

$$\mathbf{k} = (\mathbf{MX})^{-1} \times (\mathbf{g} \oplus \mathbf{h}) = (\mathbf{MX})^{-1} \times \mathbf{g} \oplus \mathbf{h}', \tag{1.4}$$

where $\mathbf{h}' = (\mathbf{MX})^{-1} \times \mathbf{h}$ is a fixed binary 8-bit vector.

It is noted that adding with fixed vectors can easily be implemented by changing the corresponding XOR gate to XNOR gate.

1.2.3 Combined Forward/Inverse AES S-box

In the combined forward/inverse S-box shown in Figure 1.6, the inversion over the $GF(2^8)$ field of AES is shared between the forward and the inverse S-boxes. Based on the operation needed, either encryption or decryption, the input and output mappings of the forward or the inverse S-box, respectively are chosen using a set of multiplexers as shown in Figure . We use the notation $\mathbf{X}_e^{-1}$ to refer to the input mapping used in the encryption path and $\mathbf{X}_d^{-1}$ to refer to the decryption one.

1.3 Logic Minimization Algorithms

In this section, we will review the logic minimization algorithms used to minimize the input and output linear mappings of the AES S-box.

If $\mathbb{A} = \mathbf{Tr}\times\mathbb{B}$, where $\mathbb{A}$ and $\mathbb{B}$ are elements in two different isomorphic fields and **Tr** is a generic transformation matrix, then logic minimization algorithms are used to find the smallest number of gates that is required to implement the transformation matrix **Tr**. The best solution could only be

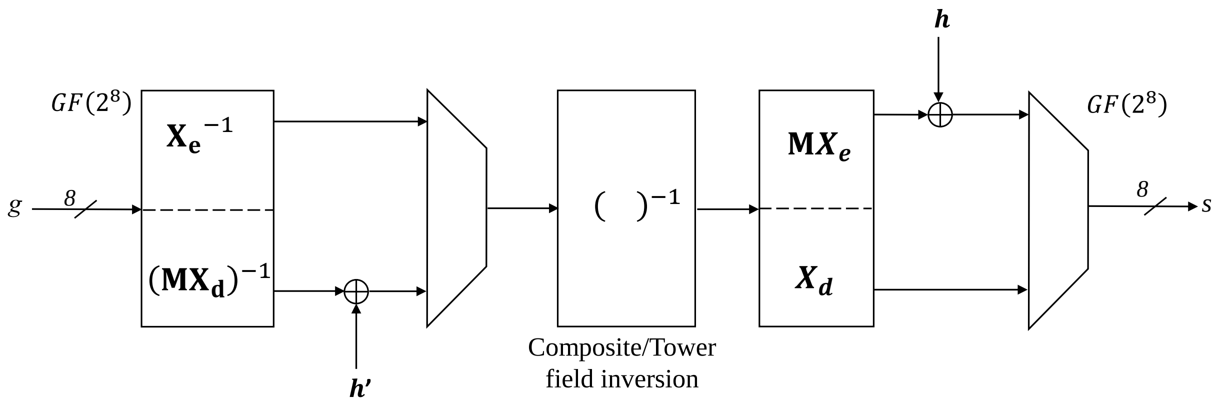


Figure 1.6: The combined forward/inverse AES S-box.

found by an exhaustive search over all possible circuits, which quickly becomes infeasible at large input transformation matrices.

Paar proposed a heuristic that works by iteratively selecting the XOR gate which is the most common in the target equations [61, 62]. The matrix is then appended by adding this gate, and the target equations are updated accordingly. This gate-selection and matrix-updating continues until there is a single '1' in each row of the matrix.

Canright used an exhaustive search over all possible solutions [21], as the target matrices were relatively small (8×8). In order to manage this time-consuming search, he searched within a small subset of the tower fields that he studied (only 27 cases out of the 432 fields). The smallest circuit that was found required 13 XOR gates at the input matrix $\mathbf{X}^{-1}$ and 11 XOR gates at the output matrix **MX** of the AES S-box.

Boyar and Peralta showed that the algorithms used by both Canright and Paar are cancellation-free, where the selected gates must always result in an output with a Hamming weight that is the addition of the Hamming weights of the inputs [18]. i.e. Canright and Paar did not use the intermediate values that were calculated once for other computation steps. Hence, in these solutions, XOR gates are never used to cancel-out common terms. This means that the exhaustive search conducted by Canright did not exhaust all the actually possible solutions. Hence, they proposed a new heuristic algorithm, the Normal-BP heuristic, that is not cancellation-free.

Using the Normal-BP heuristic, Boyar and Peralta did not find any smaller implementation for the matrices used by Canright. As a work around, and being supported by fast searching algorithms, they included all the linear equations in the inner circuit of Canright S-box (parts of the tower field inversion circuit) that directly followed the input matrix $\mathbf{X}^{-1}$ and those that preceded the output matrix **MX** into the input and output transformation matrices, respectively. The new extended-input matrix became 22×8 and the new extended-output matrix became 8×18. These new matrices are computationally intractable for the exhaustive search algorithms, and interesting targets for the fast heuristic-based algorithms. The extended-input matrix directly computes 22 equations at a cost of 23 XOR gates. For comparison, the original Canright circuit would need 27 gates to solve these equations (13 for the 8×8 input matrix $\mathbf{X}^{-1}$ followed by 14 extra gates to solve the 14 extra equations). Similarly, the extended-output matrix required 30 XOR gates.

Visconti et al. proposed a tweaked algorithm that sometimes works better for dense matrices [81]. It depends on computing the common path of the target matrix using its boolean complement, which has a lower density, before applying the regular Normal-BP heuristic algorithm.

Reyhani et al. proposed several improvements to the Normal-BP algorithm [69]. They proposed three alternatives, namely Improved-BP, Shortest-Dist-First and Focused-Search logic minimization algorithms. They also added a delay-controlled option to the Normal-BP algorithm so that the improved algorithm could find the best circuit at a certain maximum delay. The focused-search algorithm is a speed-optimized variant of exhaustive search and is not cancellation-free.

The focused-search logic-minimization algorithm outperforms all previous logic-minimization algorithms, as demonstrated by [69]. The focused-search algorithm is slow as it slightly moves toward being an exhaustive search.

Maximov et al. proposed a new technique called the floating multiplexers, where the shared part between two linear inputs to the multiplexer is pushed after the multiplexer in order to save the hardware [54]. By using floating multiplexers, it is possible to save the circuit area as the shared part will only need to be computed once instead of twice in the case that we compute it for both inputs to the multiplexer. They used this idea to design a subfiled inverter with only 9 gates and depth 3. The floating multiplexers are also used to get a very compact combined S-box, by including the multiplexers in the minimization algorithm and sharing parts of the input and output mappings of the forward and the inverse S-box to further reduce the area.

Recently, Tan et al [75] proposed two heuristics that improves the Normal-BP algorithm, namely A1 and A2, to minimize the number of XOR gates in linear circuits. This is done by implementing the outputs (targets) that have the minimal number of XOR gates first. A1 and A2 algorithms were used on the MixColumns diffusion matrix of the AES block cipher, the resulting circuit requires only 94 XOR gates, which was a new record. A new record of 92 XOR gates was recorded in [53], which is currently the smallest number of XOR2 gates implementation of the MixColumns transformation step available in the literature, up to our knowledge. The previous implementation record of 97 XOR gates was reported in [44] and improved to 95 XOR gates in [11]. Since the implementation area of a 3-input XOR gate is less than the sum of the areas of two 2-input XOR gates in most technology libraries, the authors of [11] explained how to use a tree based heuristics to convert a circuit from only using 2-input XOR gates to using both 2-input XOR gates and 3-input XOR gates. As a result, they were able to provide more compact results for some linear circuits than previous work.

It is worth mentioning that there are some available tools that can handle both linear or non-linear layers in a circuit. Examples are [18, 84, 73, 38].

1.4 Motivation and Significance

The AES S-box is the main component of the AES algorithm and plays an important role in determining the overall efficiency of the hardware implementation. Faster and/or more efficient S-boxes were proposed in [19, 15, 78, 69], while faster and/or more efficient combined forward/inverse S-box were proposed in [45, 52, 59, 79]. In addition, an automated search for a field representation in $GF((2^4)^2)$ was conducted in [51, 33] to optimize the area of high speed AES cores to be suitable for memory encryption engines.

The research for S-box designs that are more compact than Canright was explored in a line of work by Boyar et al. [16, 18, 17, 14], Reyhani-Masoleh et al. [69, 68, 70] and Maximov et al. [54]. Generally speaking, for the forward S-boxes proposed in these contributions and the inverse S-boxes proposed in [14, 70, 54], the authors moved all the linear operations in the input and the output of the composite/tower field inversion circuit to the input and output isomorphic mappings, respectively. Then, logic-minimization heuristics were used to reduce the number of XOR gates that is required to implement the extended isomorphic mappings. The currently smallest design of the AES forward S-box can be found in [54]. Note that, although the number of gates of the inverse S-box design in [14] is smaller than Canright [21], the actual implementation area of Canright [21] is smaller as tested in both STM 65nm and NanGate 15nm technologies. [69] used the composite field representation $GF((2^4)^2)$, whereas [21], [14] and [68, 70] used the tower field representation $GF(((2^2)^2)^2)$.

In this thesis, we evaluated some of the most prominent forward, inverse and combined AES

S-boxes available in the literature. We improved some previous forward S-boxes for Application-Specific Integrated Circuit (ASIC) implementation. We analyzed the hardware complexity and compared the implementation results of different S-boxes. Very-high-speed integrated circuit Hardware Description Language (VHDL) coding was used as a design entry method to Synopsys Design Compiler® and STM 65nm CMOS standard cell library was used for logic synthesis. For each and every code, we verified the codes using the S-box testbenches and Modelsim® and by comparing against the legitimate S-box outputs.

We improved the area and the delay of the exponentiation block and the area of the subfield inversion block of the S-box inversion circuit in [69]. As a result, the area and the delay of the lightweight S-box improved and the area of the fast S-box reduced too while maintaining the same critical path delay.

For the lightweight S-box proposed in [69], we conducted a search over all input transformation matrices of the S-box that are modified by multiplying the S-box input by an 8-bit binary field element in order to find a lower area one than [69]. We did not find such a matrix.

We also improved the speed of the fast S-box proposed in [69], by modifying its architecture so that no output mapping is needed as suggested in [54]. We searched over all extended input transformation matrices that are multiplied by an 8-bit element in order to find a lower delay and a lower area one. This resulted in the fastest S-box to date, up to our knowledge.

We explained and used the method of logical effort to linearly model the critical path delay of some previous and improved S-boxes. This is done in order to standardize the calculation of the S-box delay among the different technology libraries used in the synthesis process.

Using the same technique of modifying the input and output transformation matrices of the S-box by multiplying the input byte by a binary field element that ranges from 1-255, we improved the area of the low area forward S-box proposed in [68]. Due to the different field representation selected by the logic minimization algorithm, we redesigned the inversion circuit of the improved S-box correspondingly.

1.5 Thesis Outline

The rest of the thesis is organized as follows. In Chapter 2, we reviewed some formulations from Reyhani et al. composite field $GF((2^4)^2)$ S-box [69] and Reyhani et al tower field $GF(((2^2)^2)^2)$ S-box [70]. This is an essential step to understand the underlying S-box architectures before introducing our improvements in Chapter 4 and Chapter 5. In Chapter 3, different AES forward, inverse and combined AES S-boxes are evaluated and compared in terms of area and delay. Some previous AES S-boxes are improved by using more area and delay efficient gates.

The starting point of Chapter 4 is Reyhani et al. composite field S-box proposed in [69]. In Chapter 4, we improved the lightweight and the fast S-boxes by slightly modifying the exponentiation stage and designing a new subfield inverter stage. Using the improved exponentiation and new subfield inverter blocks, the improved lightweight S-box has a smaller area and delay than [69] and the improved fast S-box has a smaller area and same delay as [69]. We derived the matrix form of multiplying the S-box input by an 8-bit field element, then we use this matrix to modify the input and output transformation matrices of the lightweight S-box. We conducted a search over all the distinct field representations and all the values of the 8-bit field element, using the focused-search logic minimization algorithm [69], in order to find a lower complexity input mapping of the lightweight S-box. We also modified the fast S-box architecture by removing the output mapping as suggested in [54] and extending the input transformation matrix. We searched over all the extended input transformation matrices that are modified by multiplying the S-box input by a binary field element that ranges from (1-255). The resulting improved fast S-box is currently the fastest and most efficient (as measured by area $\times$ delay) S-box available in the literature, up to our knowledge.

We also used the method of logical effort [74] to linearly model the critical path delay of the S-box in order to standardize the calculation of the delay among different technology libraries used in the synthesis process. We used this method to evaluate and compare the delay of previous and improved fast S-box architectures. The results obtained from the logical effort method match those obtained from Synopsys Design Compiler® for the delay.

In Chapter 5, we started with the low area S-box proposed in [70]. We multiplied the input of the S-box by an 8-bit binary field element and then we did a search over all input mappings of the S-box for a lower complexity input matrix. The logic minimization algorithm is then applied to the output transformation matrices corresponding to the lowest complexity input matrices in order to select the improved mappings. As a result of the different field representation selected by the logic-minimization algorithm for the improved mappings, the inversion circuit is re-designed. Our analysis and the ASIC implementation results show that the improved S-box outperforms the original scheme in terms of area.

Finally, Chapter 6 highlights the contributions of this thesis and lists the future work.

Chapter 2

Preliminaries

In this Chapter, we will review the underlying architecture and necessary formulations of both Reyhani et al. composite field $GF((2^4)^2)$ S-box proposed in [69] and Reyhani et al. tower field $GF(((2^2)^2)^2)$ S-box introduced in [70]. Both structures will be improved later in Chapter 4 and Chapter 5.

2.1 Reyhani et al. Composite Field $GF((2^4)^2)$ S-box

In [69], Reyhani et al. used the composite field implementation for the S-box. The matrix used for the input isomorphic mapping of the S-box is denoted as as $\mathbf{X}^{-1}$ [21]. Therefore, the S-box input vector $\mathbf{g} = [g_7, \cdots, g_1, g_0]^{tr}$ represented in the binary field is converted to the corresponding vector representation in the used composite field, i.e., $\mathbf{i} = [a_0, a_1, a_2, a_3, b_0, b_1, b_2, b_3]^{tr}$ as $\mathbf{i} = \mathbf{X}^{-1} \times \mathbf{g}$. The corresponding composite field S-box is shown in Figure 2.1. The internal dashed block in this figure performs the inversion of $i = (a_0, a_1, a_2, a_3, b_0, b_1, b_2, b_3)$ over the composite field $GF((2^4)^2)$ as $o = (w_0, w_1, w_2, w_3, z_0, z_1, z_2, z_3) = i^{-1} \in GF((2^4)^2)$ which can be converted back to the binary field as $\mathbf{f} = \mathbf{X} \times \mathbf{o} = \mathbf{X} \times [w_0, w_1, w_2, w_3, z_0, z_1, z_2, z_3]^{tr}$. As a result the output of the composite field S-box can be computed using (1.2) as $\mathbf{s} = \mathbf{MX} \times \mathbf{o} \oplus \mathbf{h}$.

2.1.1 Composite Field $GF((2^4)^2)$ Construction

The elements are converted to an isomorphic composite field $GF((2^4)^2)$. The irreducible polynomial over $GF(2^4)$, namely

$$p(y) = y^2 + \mu y + \nu = (y + \gamma)(y + \gamma^{16}), \tag{2.1}$$

is used to construct the composite field. In (2.1), γ is its root and the subfield elements $\mu, \nu \in GF(2^4)$ should be chosen so that this polynomial is irreducible over $GF(2^4)$. Then, $\{\gamma, \gamma^{16}\}$ is the Normal Basis (NB) and every element g in $GF(2^8)$ can be mapped to its composite field representation over $GF(2^4)$ as $g = A\gamma + B\gamma^{16}$, where $A = (a_0a_1a_2a_3)$ and $B = (b_0b_1b_2b_3)$ are subfield elements in $GF(2^4)$ and a_i and b_i are their binary coordinates, respectively.

The subfield $GF(2^4)$ is generated using the irreducible all-one-polynomial (AOP) with degree four, i.e.,

$$r(t) = t^4 + t^3 + t^2 + t + 1. \tag{2.2}$$

If β is a root of $r(t)$, i.e., $r(\beta) = 0$. Then, the type-I optimal normal basis (ONB-I) $\{\beta, \beta^2, \beta^{2^2}, \beta^{2^3}\}$ is used for representing field elements and their efficient computations over $GF(2^4)$ [66]. Then, any field element $A = (a_0a_1a_2a_3) \in GF(2^4)$ can be represented as $A = a_0\beta + a_1\beta^2 + a_2\beta^{2^2} + a_3\beta^{2^3}$.

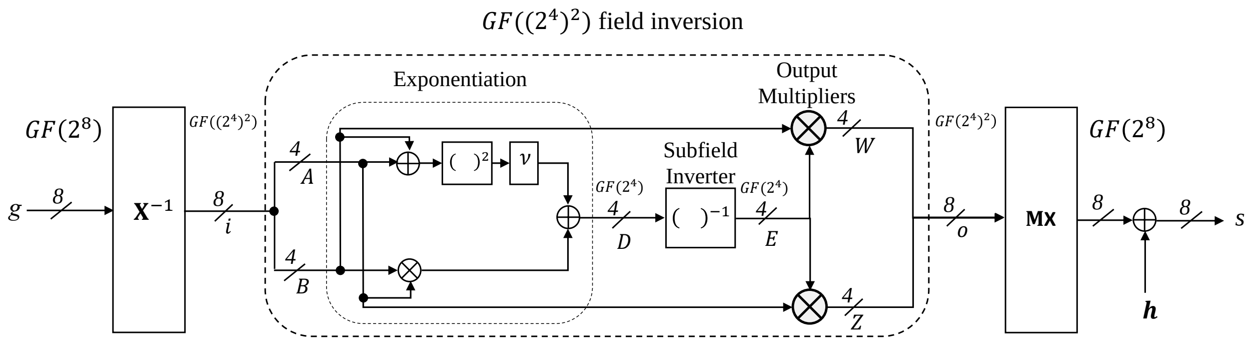


Figure 2.1: The architecture of the S-box using $GF(((2^4)^2)$ in the NB representation.

2.1.2 Inversion over Composite Field $GF((2^4)^2)$

The inverse of $g = A\gamma + B\gamma^{16}$ in the composite field $GF((2^4)^2)$ can be written as [36, 61], $g^{-1} = (g^r)^{-1}g^{r-1}$, where $r = \frac{2^{4\times2}-1}{2^4-1} = 17$. Then, $g^{-1} = (g^{17})^{-1}g^{16}$ can be computed as

$$\begin{aligned} g^{-1} &= (gg^{16})^{-1}g^{16} \\ &= [(A\gamma + B\gamma^{16})(B\gamma + A\gamma^{16})]^{-1}(B\gamma + A\gamma^{16}) \\ &= [AB(\gamma + \gamma^{16})^2 + (A + B)^2\gamma\gamma^{16}]^{-1}(B\gamma + A\gamma^{16}) \\ &= EB\gamma + EA\gamma^{16} = W\gamma + Z\gamma^{16}, \end{aligned} \tag{2.3}$$

where

$$E = D^{-1} = [g^{17}]^{-1} = [AB(\gamma + \gamma^{16})^2 + (A + B)^2\gamma\gamma^{16}]^{-1}, \tag{2.4}$$

$W = EB$ and $Z = EA$. Since γ and γ^{16} are roots of (2.1), then $\gamma + \gamma^{16} = \mu$ and $\gamma\gamma^{16} = \nu$. Thus, $D \in GF(2^4)$ in (2.4) is simplified to

$$D = AB\mu^2 + (A + B)^2\nu. \tag{2.5}$$

By taking $\mu = 1 \in GF(2^4)$, then (2.1) and (2.5) would respectively become

$$p(y) = y^2 + y + \nu = (y + \gamma)(y + \gamma^{16}), \tag{2.6}$$

and

$$D = AB + (A + B)^2\nu. \tag{2.7}$$

The underlying arithmetic operations for the $GF((2^4)^2)$ inversion are shown in Figure 2.1. The inversion over $GF((2^4)^2)$ requires an addition, a squaring, three multiplications and multiplication by the constant ν (scaling), which are implemented over subfield $GF(2^4)$. The $GF((2^4)^2)$ inversion consists of three main blocks, namely exponentiation computation, subfield inverter, and output multipliers.

The overall architecture of Reyhani et al. composite field S-box is highlighted in Figure 2.2. The input and output transformation blocks are responsible for converting elements between the AES binary field $GF(2^8)$ and the corresponding elements in the composite field $GF((2^4)^2)$. The input transformation block accepts an 8-bit element g from the $GF(2^8)$ field, and generates two 4-bit field elements A and B, and the mod-2 addition between every two bits in each of the two elements of A and B, i.e., A_{jk} and B_{jk} , where A_{jk} is a set that contains $a_j \oplus a_k$, and B_{jk} contains $b_j \oplus b_k$, both for $0 \leq j, k \leq 3$, and $j \neq k$. The output transformation block accepts the result of the two subfield multipliers; W and Z (5 bits each), and generates the corresponding element s, i.e., the S-box output, in the AES $GF(2^8)$ field.

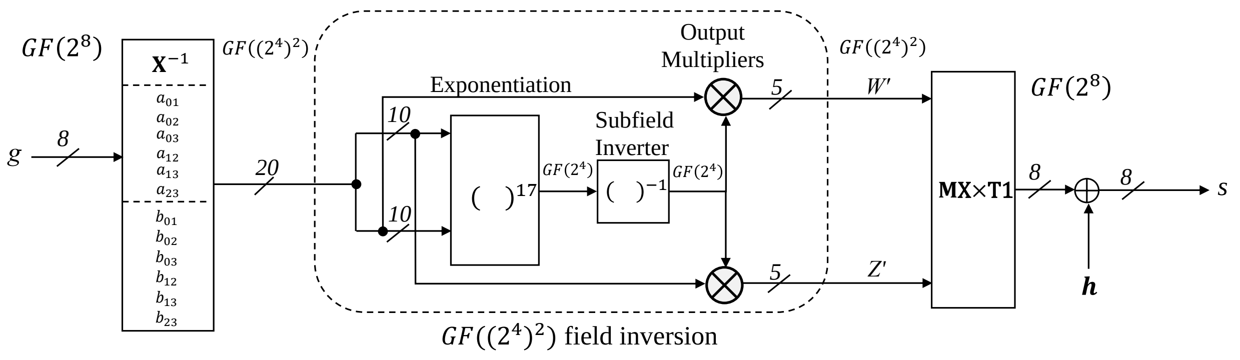


Figure 2.2: Reyhani et al. composite field S-box.

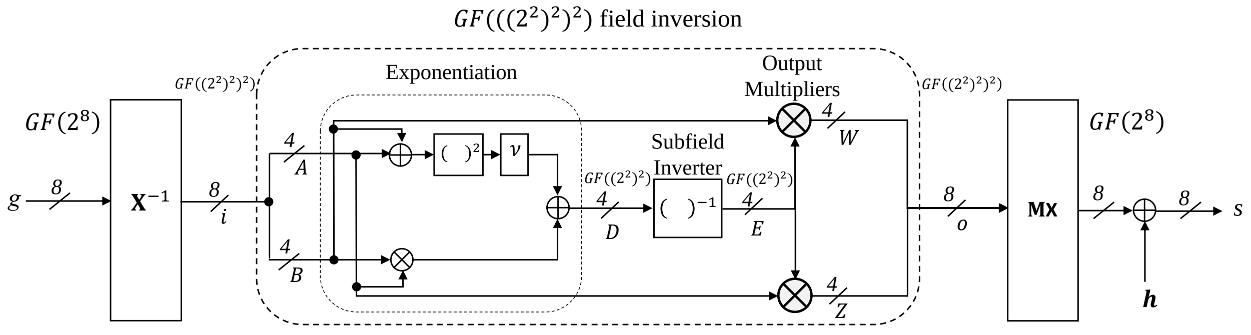


Figure 2.3: The architecture of the S-box using $GF(((2^2)^2)^2)$ in the NB representation.

2.2 Reyhani et al. Tower Field $GF(((2^2)^2)^2)$ S-box

In [70], the authors used the tower field implementation for the S-box. The matrix used for input isomorphic mapping of the S-box is denoted as as $\mathbf{X}^{-1}$ [21]. The S-box input vector $\mathbf{g} = [g_7, \cdots, g_1, g_0]^{tr}$ is converted to the corresponding vector representation in the used tower field, i.e., $\mathbf{i} = [a_0, a_1, a_2, a_3, b_0, b_1, b_2, b_3]^{tr}$ as $\mathbf{i} = \mathbf{X^{-1}} \times \mathbf{g}$. The corresponding tower field S-box architecture is shown in Figure 2.3. The inversion of $i = (a_0, a_1, a_2, a_3, b_0, b_1, b_2, b_3)$ is performed over the tower field $GF(((2^2)^2)^2)$ as $o = (w_0, w_1, w_2, w_3, z_0, z_1, z_2, z_3) = i^{-1} \in GF(((2^2)^2)^2)$ which can be converted back to the binary field as $\mathbf{f} = \mathbf{X} \times \mathbf{o} = \mathbf{X} \times [w_0, w_1, w_2, w_3, z_0, z_1, z_2, z_3]^{tr}$. As before, the output of the tower field S-box can be computed using (1.2) as $\mathbf{s} = \mathbf{MX} \times \mathbf{o} \oplus \mathbf{h}$.

2.2.1 Tower Field $GF(((2^2)^2)^2)$ Construction

Here, a field element $g = (g_7, ..., g_0) \in GF(2^8)$ is converted to an isomorphic tower field $i = (a_0, a_1, a_2, a_3, b_0, b_1, b_2, b_3) \in GF(((2^2)^2)^2)$ defined using the irreducible polynomial over $GF((2^2)^2)$:

$$p(y) = y^2 + y + \nu = (y + \gamma)(y + \gamma^{16}), \tag{2.8}$$

where γ (its root) and ν are subfield elements in $GF((2^2)^2)$ that should be chosen so that this polynomial is irreducible over $GF((2^2)^2)$. Then, $\{\gamma, \gamma^{16}\}$ is the NB over $GF((2^2)^2)$ and every element g in $GF(2^8)$ can be mapped to its tower field representation over $GF((2^2)^2)$ as $i = (a_0, a_1, a_2, a_3, b_0, b_1, b_2, b_3) = A\gamma + B\gamma^{16}$, where $A = (a_0, a_1, a_2, a_3)$ and $B = (b_0, b_1, b_2, b_3)$ are subfield elements in $GF((2^2)^2)$.

Similarly, the subfield $GF((2^2)^2)$ is generated using the irreducible polynomial over $GF(2^2)$ of

$$q(z) = z^2 + z + N = (z + \alpha)(z + \alpha^4), \tag{2.9}$$

with its root α that generates the NB $\{\alpha, \alpha^4\}$ over $GF(2^2)$. In (2.9), $N \in GF(2^2)$ should be selected so that $q(z)$ is irreducible over $GF(2^2)$. Therefore, any subfield element $A = (a_0, a_1, a_2, a_3) \in$

$GF((2^2)^2)$ can be represented with respect to the NB $\{\alpha, \alpha^4\}$ by $A = A_0\alpha + A_1\alpha^4$ where $A_0 = (a_0, a_1)$, $A_1 = (a_2, a_3) \in GF(2^2)$. The RNB $\{\alpha, \alpha^4, 1\}$ is used to represent field elements over $GF(2^2)$, where $\alpha + \alpha^4 = 1 \in GF(2^2)$. The hat notation is used to represent the coordinates with respect to the RNB. Therefore, the element A can also be represented by $A = \hat{A}_0\alpha + \hat{A}_1\alpha^4 + \hat{A}_2$, where $\hat{A}_i \in GF(2^2)$, and $A_i = \hat{A}_i + \hat{A}_2$ for $i = 0, 1$ as $\alpha + \alpha^4 = 1$.
To construct the binary field $GF(2^2)$, the irreducible all-one-polynomial (AOP) with degree 2,

$$r(t) = t^2 + t + 1 = (t + \omega)(t + \omega^2), \tag{2.10}$$

is used to generate the NB $\{\omega, \omega^2\}$ over $GF(2)$. Therefore, the field elements $A_0, A_1 \in GF(2^2)$ can be represented as $A_0 = (a_0, a_1) = a_0\omega + a_1\omega^2$ and $A_1 = (a_2, a_3) = a_2\omega + a_3\omega^2$, respectively. As a result, any subfield element $A = (a_0a_1a_2a_3) \in GF((2^2)^2)$ can be represented as

$$A = (a_0\omega + a_1\omega^2)\alpha + (a_2\omega + a_3\omega^2)\alpha^4, \tag{2.11}$$

where a_i are the binary coordinates. The RNB $\{\omega, \omega^2, 1\}$ is also used to represent an element over $GF(2^2)$, where $\omega + \omega^2 = 1 \in GF(2)$. Similarly, the hat notation is used for the coordinates with respect to the RNB. Therefore, A_0 can also be represented as $A_0 = \hat{a}_0\omega + \hat{a}_1\omega^2 + \hat{a}_2$ where $\hat{a}_i \in GF(2)$, $i \in [0, 2]$ are the coordinates of A_0 with respect to the RNB. Hence $a_i = \hat{a}_i \oplus \hat{a}_2$ for $i = 0, 1$ as $\omega + \omega^2 = 1$.

To make $p(y)$ in (2.8) irreducible, the constant v in (2.8) can take any of the following 8 different values: {[1000],[0100],[0010],[0001],[1110],[1101],[1011],[0111]}. The 4-bits of each value represent the coefficients of $v_i, i \in [0, 3]$ in $v = (v_0\omega + v_1\omega^2)\alpha + (v_2\omega + v_3\omega^2)\alpha^4$. Similarly, the constant N in (2.9) can take any of the following 2 values: {[10],[01]}, where the 2-bits represent the coefficients of $N = N_0\omega + N_1\omega^2$.

2.2.2 Inversion over Tower Field $GF(((2^2)^2)^2)$

The multiplicative inverse of $g = A\gamma + B\gamma^{16}$ in the tower field $GF(((2^2)^2)^2)$ can be written as $g^{-1} = (g^{17})^{-1}g^{16}$ [36, 61]. Since $g^{16} = B\gamma + A\gamma^{16}$, one can define $D = g^{17} = (A\gamma + B\gamma^{16})(B\gamma + A\gamma^{16})$, which can be simplified to

$$D = A \times B + (A + B)^2 v. \tag{2.12}$$

Then, $g^{-1} = D^{-1}(B\gamma + A\gamma^{16})$. Assuming that $g^{-1} = W\gamma + Z\gamma^{16}$, where $W, Z \in GF((2^2)^2)$, the outputs of the inversion $g^{-1} \in GF(((2^2)^2)^2)$ can be found by computing

$$\begin{aligned} W &= B \times D^{-1} = B \times E \\ Z &= A \times D^{-1} = A \times E, \end{aligned} \tag{2.13}$$

where $E = D^{-1}$.

The inversion over tower field in Figure 2.3 consists of the following steps:

1. Convert the input g to an equivalent element in the tower field representation A and B.
2. Use A and B to compute D in (2.12).
3. Find E, the multiplicative inverse of D.
4. Find $W = B \times E$ and $Z = A \times E$, which represent the output in tower field representation.
5. Convert the outputs W and Z (which corresponds to inversion output o in the tower field) to an equivalent element in the binary field f, which is the inversion output.

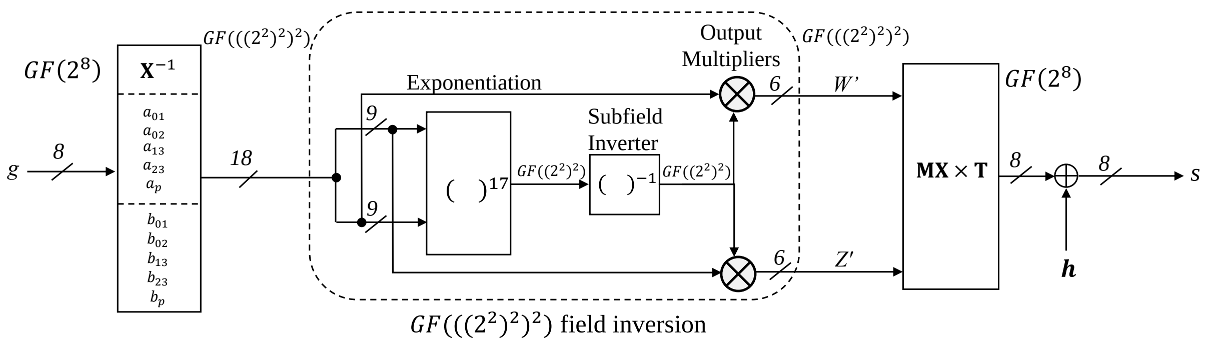


Figure 2.4: Reyhani et al. tower field S-box.

The overall architecture of Reyhani et al. tower field S-box is highlighted in Figure 2.4. In the beginning, the 8-bit input g is processed through the input isomorphic mapping to find the equivalent elements A and B, of 4-bit each, in the tower field representation. The $GF((2^2)^2)$ elements A and B represent the inputs of the inversion circuit. Some XOR gates from the tower field inversion are included into the input and output mappings of the S-box. The input mapping generates A and B, of 4-bit each, and also $\{a_{01}, a_{02}, a_{13}, a_{23}, a_p\}$ and $\{b_{01}, b_{02}, b_{13}, b_{23}, b_p\}$, where $a_{jk} = a_j \oplus a_k$ and $b_{jk} = b_j \oplus b_k$ for $0 \leq j, k \leq 3$, and $j \neq k$. Also, $a_p = a_{02} \oplus a_{13}$ and $b_p = b_{02} \oplus b_{13}$ which are the parities of A and B, respectively.

Similarly, the output mapping implements the last stage of the output multipliers. The $GF((2^2)^2)$ multiplier output is represented as six terms (6-bit output) denoted by W' and Z' as RNB6 (6-bit redundant normal basis) representation, before reduced to four terms (4-bit output). Therefore, the output mapping of the forward S-box accepts two redundant RNB6 elements with a total of 12 bits, and generates an 8-bit output in the binary field of $GF(2^8)$ as shown in Figure 2.4.

2.3 Conclusion

In this Chapter, we reviewed the preliminaries of the two recent S-boxes proposed in [69, 70]. This is an essential step before introducing our area and delay improvements to the aforementioned S-boxes in Chapter 4 and Chapter 5.

Chapter 3

Implementation Results and Comparisons of Forward, Inverse and Combined AES S-boxes

In this Chapter, we will evaluate and compare some previous and improved forward, inverse and combined AES S-boxes. For every code, we first verify the code at the Register-Transfer Level (RTL) using Mentor Graphics Modelsim®, by comparing against the legitimate S-box outputs. The design is then synthesized using STM 65nm standard cell library where VHDL coding is utilized as the design entry method to Synopsys Design Compiler®.

3.1 Forward, Inverse and Combined AES S-boxes

In this section, we will provide a review about some AES S-boxes introduced in the literature. The AES S-boxes that represent the most significant designs, in terms of area and/or delay, up to our knowledge, will be implemented and compared in this Chapter.

Canright S-box has been known as the most compact S-box design since its introduction back in CHES'05. Canright implemented tower field inversion using normal basis, instead of polynomial basis [21]. In Addition, he conducted an exhaustive search through all the subfield representations (a total of 432 fields) to reduce the overall implementation area. Kasper et al. [40] used Canright S-box in a bit-sliced software implementation of AES. The authors converted each logic gate of Canright S-box to its equivalent CPU instruction. As a result of using Canright optimized S-box, the instruction count of [40] improved by 15%. Boyar-Peralta proposed logic-minimization heuristics that could reduce the gate count of Canright S-box from 120 gates to 113 gates, however synthesis results did not reflect much improvement. In the track of lightweight S-boxes, Boyar, Peralta, along with others proposed several logic-minimization heuristics to reduce the gate count of Canright S-box [18, 17]. They did not explore substantially different subfields, but they focused on reducing the number of gates that are needed to implement Canright circuit itself. They proposed a reduction in the gate count of the S-box circuit from 120 gates in Canright circuit to 115 gates in [18, 17], to 114 gates in [81], and to 113 gates in [14]. In CHES'15, Ueno et al. proposed an S-box that has a slightly higher area, but significantly faster than the previous designs, hence it was the most efficient (measured by area $\times$ delay) S-box implementation to that date. Ueno et al. used their fast S-box later to design an AES architecture with a very short critical path delay in [80].

The design of Canright S-box did not target fast application. Hence, the track of high-speed/higher-efficiency S-boxes received more research focus. [71, 39, 60, 59, 78] are some papers that fall in this latter category. Boyar et al. also proposed low-depth circuits for Canright S-box using delay-controlled logic-minimization heuristics [19, 15].

In 2018, Reyhani et al. proposed two new S-box designs, based on a new composite field $GF((2^4)^2)$ [69]. The first design, the lightweight S-box, was the smallest S-box, in terms of the actual ASIC implementation area, as compared to previous work. The second design, the fast S-box, was the fastest and most efficient S-box design then. They also proposed new logic-minimization heuristics that give smaller and faster circuits than Boyar-Peralta. Later in 2019, Reyhani et al. introduced three new lightweight designs for the forward, inverse and combined AES S-boxes [70], based on a new tower field $GF(((2^2)^2)^2)$. Each design had the smallest area compared to previous designs in its respective category.

Maximov et al. recently proposed new circuit minimization techniques for smaller and faster S-boxes [54]. It is worth mentioning that Maximov fast design is currently the fastest and most-efficient S-box design. Also, Maximov bonus S-box is currently the smallest S-box design, up to our knowledge.

3.2 Behavioural and Structural VHDL Coding

The S-box circuit can be coded in VHDL using either behavioral or structural coding. This is done so we can reach to the optimum circuit, in terms of area and/or delay for each block of the S-box. In behavioral modeling, we define the circuits based on their input-output relationship (their behavior), and let the Computer-Aided Design (CAD) tool selects the best implementation based on the available gates in the target library. In the structural modeling, we define the exact gates and circuit structure based on formulations or circuit diagram very precisely, with no optimization in the gate selection by the CAD tool. By modeling the circuit structurally, we are enforcing the desired synthesis results on the CAD tool. We also compared the ASIC implementation results of the behavioral modeling against the structural modeling in order to select the optimum implementation. By observing the synthesis results of both structural and behavioural VHDL coding of the circuit, we are able to select a coding style that is capable of meeting the required design constraint, either area, delay or both.

3.3 Implementation Results and Comparisons of Forward AES S-boxes

In this section, we will evaluate and compare the implementation results of some previous and improved forward AES S-box circuits. All the implementation areas in Gate Equivalents (GEs) estimated in this section are computed based on the area of individual gates in the STM 65nm technology library.

3.3.1 Comparisons of the Space and Time Complexities

In this section, we will list the space and time complexities of the fastest and most compact forward AES S-box architectures available in the literature, up to our knowledge. In addition, we will list the space and time complexities of the improved S-box architectures proposed in this thesis.

Tables 3.1 and 3.2 provide the hardware complexity and time delay of 19 previous and improved S-boxes. We chose five of the most lightweight schemes, namely Canright [21, 20], and the 113-gate design in [14], which was found using the Boyar-Peralta heuristic [18] while exhaustively searching through all the ties, Reyhani et al. lightweight S-box [69], Reyhani et al. low area S-box [70] and Maximov et al. bonus S-box [54]. In addition, we chose five of the fastest schemes, namely the two Boyar et al. designs in [19] and [15], along with the Ueno et al. design [78], Reyhani et al. fast S-box [69] and Maximov et al. fast S-box [54]. The space complexity comparison of fast designs helps in detecting where the speed improvements came from and also in validating the CAD tool results.

The gate count for Canright S-box is obtained from [21] and [20]. Let us denote the 113-gate design in [14] as Boyar-Op113, the 16-depth circuit, with 128 gates in [19] as Boyar-Dp16-1, and the new 16-depth circuit, with 125 gates in [15] (which is available in [14]) as Boyar-Dp16-2.

The Boyar-Op113 scheme requires 113 gates (81 XOR with 32 AND operations). The Boyar-Dp16-1 scheme uses 94 XOR2/XNOR2 with 34 AND operations, while the Boyar- Dp16-2 scheme uses 91 XOR2/XNOR2 with 34 AND operations. We improved the 113-gate S-box of [14] to use NAND gates instead of AND gates, resulting in the first S-box circuit to have an actual lower implementation area than the one proposed by Canright (not just the gate count). We also improved the low-depth circuits of [19, 15]. It is cheaper and faster to use NAND gates instead of AND gates in the ASIC implementations. Therefore, the improved schemes is obtained by changing the AND gates to NAND gates and tracing the results of such changes by replacing some XOR to XNOR and some XNOR to XOR. Boolean algebra are used whenever the input of an AND gate is complemented to change the AND gate to a NOR gate. To adjust the complement of the other AND input, its complement is moved to the gate that generates it and the complement of this gate is traced to all the other gates. At the end, the correctness of the improved schemes is verified by testbenches and using the CAD tool.

The gate count of the original Boyar-Op113 [14], the Boyar-Dp16-1 [19], and the Boyar-Dp16-2 [15] schemes as well as those of the improved schemes are presented in Table 3.1. In Table 3.1, we also provided the GEs for all S-box architectures using the corresponding GEs of the gates from the STM CMOS 65nm technology. In our GEs calculations, we used XOR2/XNOR2 (2 GEs), AND2 (1.25 GEs), OR2 (1.25 GEs), NAND2 (1 GE), NAND3 (1.25 GEs), NOR2 (1 GE), and NOT (0.75 GEs).

The gate count of the $GF((2^4)^2)$ inversion of the original scheme proposed by Ueno et al. is provided in [78] as (51 XOR2 + 38 AND2 + 16 OR2 + 4 NOT) gates. However, no information regarding the gate count of the input matrix (denoted by Δ_f in their paper) and output matrix (denoted by Δ_l in their paper) is provided in the paper. We drove the gate counts of the input and output matrices (14 and 26) to minimize the total number of XOR2/XNOR2 gates needed in these blocks. Most importantly, we made sure the delays of these transformation matrices are the same as the ones proposed in [78], namely $2D_X$ and $3D_X$, respectively. As a result, the delay of the entire S-box remains the same as the one proposed in [78], i.e., $11D_X + 3D_A + 1D_O$. The same Critical Path Delay (CPD) was obtained by the CAD tool after coding the original Ueno et al. [78] S-box architecture in VHDL. Since the original Ueno et al. S-box used several AND gates, we improved the design using NAND gates by changing the corresponding formulations of all stages and blocks in their architecture.

The CPD of those schemes are also compared in Table 3.2. These delays are obtained from the corresponding papers and are also verified by the codes. More importantly, we used the CAD tool to provide the CPD in terms of the number and type of gates. The results from the CAD tool match the ones provided in the corresponding papers except for the Boyar-Op113 scheme which is reduced to the depth of 27 from the depth of 28 mentioned in [15]. The CPD and the gate counts for the improved schemes are derived by analysis of the improved formulations, and verified using the CAD tool. As seen from Table 3.1, Maximov bonus S-box has the smallest area among all previous and improved lightweight schemes, namely Canright, Boyar-Op113, improved Boyar-Op113, Reyhani lightweight, improved lightweight, Reyhani low area and improved low Area. Also, from Table 3.2, Maximov fast S-box is faster than the previous four fast schemes, however

Table 3.1: Space complexity comparison of some forward S-boxes.

Design	Hardware Complexity	GEs*
Lightweight		
Canright [20]	80X+34ND+6NR	200
Boyar-Op113 [14]	81X+32AD	202
Improved Boyar_Op113	81X+32ND	194
Reyhani Lightweight [69]	63X+3X3+27ND+7NR+4O3+4NT	182.25
Reyhani Low Area [70]	48X+7X3+1XN3+32ND+6NR+2O3+2O4+6NT	177.75
Maximov Bonus [54]	64X+27ND+5NR+6MX	172
Imp. Lightweight (Proposed)	63X+30ND+6NR+4AOI12+4OAI212+4NT	178
Imp. Low Area (Proposed)	44X + 8X3 + 1XN3 + 32ND + 6NR + 2O3+2O4 + 6NT	173.5
Fast		
Boyar-Dp16-1 [19]	94X+34AD	230.5
Imp. Boyar-Dp16-1	94X+30ND+4NR	222
Boyar-Dp16-2[15]	91X+34AD	224.5
Imp. Boyar-Dp16-2	91X+30ND+4NR	216
Ueno et al. [78]	91X+38AD+16OR+4NT	256.5
Imp. Ueno	91X+35ND+4N3+13NR+4NT	238
Reyhani Fast [69]	79X+39ND+4N3+3NR+4NT	208
Maximov Fast [54]	78X+4AD+37ND+5NR+6MX	215
Fast 1 (Proposed)	93X+51ND+4N3+3NR+4NT	248
Fast 2 (Proposed)	88X+51ND+4N3+3NR+4NT	238
Fast 3 (Proposed)	77X+42ND+6NR+4AOI12+4OAI212+4NT	218

*All GEs values are estimated using STM 65nm technology where X is XOR2/XNOR2 = 2GEs, X3 is XOR3 = 3.75GEs, XN3 is XNOR3 = 4GEs, AD is AND2 = 1.25GEs, OR is OR2 = 1.5GEs, ND is NAND2 = 1GE, N3 is NAND3 = 1.25GE, NR is NOR2 = 1GE, O3 is OAI32 = 2GEs, O4 is OAI222 = 2.5GEs, AOI12 is AND2 into NOR2 = 1.25GEs, OAI212 is 2 OR2 into NAND3 = 2GEs, NT is NOT = 0.75GEs and MX is MUX21 = 2GEs.

our proposed Fast 3 S-box has the same number of gates on the critical path. Both Maximov fast S-box and the proposed Fast 3 S-box has a depth of 12 gates.

3.3.2 ASIC Synthesis Results and Comparisons

We coded all the above-mentioned S-boxes (original and improved ones) in VHDL and present their ASIC results in Table 3.3. For each and every code, we verified the codes using the S-box testbenches and Modelsim®. The results are collected at conservative wire load models and the critical path delays are reported by the CAD tool when there is no load at the external output. The power consumptions are included as reported by the CAD tool at relaxed constraints using a clock frequency of 100 MHz.

The original code for the Canright scheme was obtained from [21], where they provide a behavioral modeling (in Verilog) for the combined S-box/inverse S-box core. We revised it, with minimal changes, to a behavioral modeling of an S-box-only core, leading to a hardware cost of 208.5 GEs (denoted Canright_Beh. in the table). Then, we wrote a structural code following the formulations written in the paper, in order to exactly match the hardware complexity provided in the report, leading to a hardware cost of 200 GEs (denoted Canright_Str. in the table). Note that in [41], the authors mention that the Boyar-Peralta S-box [18] can be implemented in 193.8 GEs by converting AND gates to NAND gates, but they did not detail on the exact equations used.

The results listed in Table 3.3 show that Maximov bonus S-box is the smallest area S-box, while our proposed improved lightweight S-box is the fastest and most efficient (measured by area $\times$ delay) S-box design among other lightweight S-boxes considered.

Similarly, our proposed Fast 3 S-box is the fastest and most efficient S-box design in the fast S-box category to date. Reyhani fast S-box is the lowest in implementation area in the fast S-box category.

Table 3.2: Time complexity comparison for different S-boxes.

Design	CPD*
Lightweight	
Canright [20]	$19D_X + 3D_{ND} + 1D_{NR}$
Boyar-Op113 [14]	$21D_X + 6D_A$
Improved Boyar_Op113	$21D_X + 6D_{ND}$
Reyhani Lightweight [69]	$15D_X + 1D_{X3} + 1D_{ND} + 1D_{O3} + 1D_{NT}$
Reyhani Low Area [70]	$12D_X + 3D_{X3} + 2D_{ND} + 1D_{O4} + 1D_{NT}$
Maximov Bonus [54]	$20D_X + 1D_{ND} + 2D_{NR} + 1D_{MX}$
Imp. Lightweight (Proposed)	$15D_X + 1D_{NR} + 1D_{ND} +$ $1D_{AOI12} + 1D_{OAI212} + 1D_{NT}$
Imp. Low Area (Proposed)	$13D_X + 3D_{X3} + 2D_{ND} + 1D_{O4} + 1D_{NT}$
Fast	
Boyar-Dp16-1 [19]	$14D_X + 2D_A$
Imp. Boyar-Dp16-1	$14D_X + 1D_{ND} + 1D_{NR}$
Boyar-Dp16-2[15]	$13D_X + 3D_A$
Imp. Boyar-Dp16-2	$14D_X + 2D_{ND}$
Ueno et al. [78]	$11D_X + 3D_A + 1D_O$
Imp. Ueno	$10D_X + 4D_{ND} + 1D_{N3} + 1D_{NT}$
Reyhani Fast [69]	$11D_X + 5D_{ND} + 1D_{NT}$
Maximov Fast [54]	$8D_X + 1D_A + 1D_{ND} + 1D_{NR} + 1D_{MX}$
Fast 1 (Proposed)	$8D_X + 5D_{ND} + 1D_{NT}$
Fast 2 (Proposed)	$8D_X + 5D_{ND} + 1D_{NT}$
Fast 3 (Proposed)	$7D_X + 1D_{ND} + 1D_{NR} +$ $1D_{AOI12} + 1D_{OAI212} + 1D_{NT}$

*X = XOR2/XNOR2, X3 = XOR3, A = AND2, ND = NAND2, N3 = NAND3, NR = NOR2, O3 = OAI32, O4 = OAI222, AOI12 = AND2 into NOR2, OAI212 = 2 OR2 into NAND3, NT = NOT and MX = MUX21.

Table 3.3: ASIC comparisons of the S-Box architectures.

Design	Area		Delay	Power	Area-Time
	μm^2	GE	ns	μW	product
Lightweight					
Canright_Beh. [21]	433.68	208.5	1.287395	42.125	268.422
Canright_Sr.	416	200	1.252811	41.023	250.562
Boyar-Op113 [14]	420.16	202	1.522775	39.365	307.601
Improved Boyar-Op113	403.52	194	1.34606	40.471	261.136
Reyhani Lightweight [69]	379.08	182.25	1.197698	38.085	218.28
Reyhani Low Area [70]	369.72	177.75	1.168615	34.731	207.721
Maximov Bonus [54]	**357.76**	**172**	1.382479	36.437	237.786
Imp. Lightweight (Proposed)	370.24	178	**1.102665**	37.549	**196.274**
Imp. Low Area (Proposed)	360.88	173.5	1.2450	34.57	216.001
Fast					
Boyar-Dp16-1 [19]	479.44	230.5	0.960458	44.020	221.386
Improved Boyar-Dp16-1	461.76	222	0.905652	44.797	201.055
Boyar-Dp16-2 [15]	466.96	224.5	0.956535	42.724	214.742
Improved Boyar-Dp16-2	449.28	216	0.911743	43.645	196.936
Ueno et al. [78]	533.52	256.5	0.831007	48.178	213.153
Imp. Ueno	495.04	238	0.772424	49.609	183.837
Reyhani Fast [69]	**432.64**	**208**	0.779697	42.750	162.177
Maximov Fast [54]	447.2	215	0.686869	41.398	147.677
Fast 1 (Proposed)	515.84	248	0.636572	49.310	157.870
Fast 2 (Proposed)	495.04	238	0.627886	47.807	149.437
Fast 3 (Proposed)	453.44	218	**0.626698**	46.580	**136.62**

3.4 Implementation Results and Comparisons of Inverse AES S-boxes

In this section, we will evaluate and compare the implementation results of some previous and improved inverse AES S-box circuits. All the estimated implementation areas (in GEs) in this section are computed based on the area of individual gates in the STM 65nm technology library.

3.4.1 Complexity Analysis of Inverse S-boxes

In this section, we will compare the hardware complexity of 6 inverse AES S-box architectures, namely Canright [21], the 121-gates circuit, denoted Boyar_I121, as proposed in [14], along with our improved version of the Boyar_I121 circuit where we replaced all the AND gates by NAND gates with necessary changes for correct operation. We also will include the inverse S-box proposed in [70], denoted Reyhani inverse, Maximov et al. bonus inverse S-box and Maximov et al. fast inverse S-box proposed in [54]. The gate count of each inverse S-box design is listed in Table 3.4. It is noted that Maximov bonus inverse S-box is the smallest inverse S-box among other considered inverse S-boxes.

Table 3.4: Gate count of some inverse S-box circuits.

Design	Hardware Complexity	GEs*
Canright [21]	81X + 34ND + 6NR	202
Boyar_I121 [14]	87X + 34AD	216.5
Imp. Boyar_I121	87X + 34ND	208
Reyhani Inverse [70]	47X + 10X3 + 32ND + 6NR + 2O3 + 2O4 + 6NT	183
Maximov Bonus [54]	63X+27ND+5NR+6MX+1NT	170.75
Maximov Fast [54]	78X+41ND+5NR+6MX	214

*All GE values are estimated using STM 65nm technology where X is XOR2/XNOR2 = 2GEs, X3 is XOR3 = 3.75GEs, AD is AND2 = 1.25GEs, ND is NAND2 = 1GE, NR is NOR2 = 1GE, O3 is OAI32 = 2GEs, O4 is OAI222 = 2.5GEs, NT is NOT = 0.75GEs and MX is MUX21 = 2GEs.

3.4.2 ASIC Synthesis Results and Comparisons

For the inverse S-box, we have written the VHDL codes for the designs of the inverse S-boxes proposed by Canright [21], and Boyar et al. [14], denoted as Boyar_I112, in addition to improved Boyar_I112, Reyhani inverse S-box [70], Maximov bonus inverse S-box and Maximov fast inverse S-box [54]. Implementation results using STM 65nm technology are presented in Table 3.5. The results are collected at conservative wire load models and the critical path delays are reported by the CAD tool when there is no load at the external output. The power consumption values are included as reported by the CAD tool at relaxed constraints using a clock frequency of 100 MHz.

It is noted that the reported area from the CAD tool for the Canright design is lower than the area complexity of the inverse S-box mentioned in [21]. The formulations for original Boyar et. al inverse S-box design are obtained from [14]. We improved the inverse S-box designed by Boyar et al. in [14]. For the improved Boyar et. al inverse S-box design, we used Boolean Algebra to obtain the equivalent formulations. Our improved design uses NAND, NOR, XOR and XNOR gates, whereas the original Boyar et al. [14] inverse S-box uses AND, XOR, and XNOR gates. As shown in Table 3.5, the improved Boyar_I121 design is more compact and faster than the original Boyar_I121 inverse S-box. Moreover, Maximov bonus inverse S-box [54] is the most compact inverse S-box and Maximov fast inverse S-box [54] is the fastest inverse S-box up to date.

Table 3.5: ASIC Implementation results for some Inverse S-box.

Inverse S-box	Area		CPD	Power
	μm^2	GE	ns	μW
Canright [21]	420.16	202	1.168	41.320
Boyar_I121 [14]	450.32	216.5	1.231	42.22
Improved Boyar_I121	432.64	208	1.182	43.06
Reyhani [70]	380.64	183	1.387	35.41
Maximov Bonus [54]	**355.16**	**170.75**	1.388	35.99
Maximov Fast [54]	445.12	214	**0.648**	41.23

Table 3.6: Complexity comparison of different combined S-box designs.

Design	Hardware Complexity	GEs*
Canright [21]	94X+34ND+6NR+2NT+16MXI	257.5
Reyhani [70]	57X+12X3+32ND+6NR+2O3+2O4+6NT+14MXI+2MX	239
Maximov Bonus [54]	79X+27ND+5NR+16MX	222
Maximov Fast [54]	104X+41ND+6NR+13MX+12MXI	302

*All GE values are estimated using STM 65nm technology where X is XOR2/XNOR2 = 2GEs, X3 is XOR3 = 3.75GEs, ND is NAND2 = 1GE, NR is NOR2 = 1GE, O3 is OAI32 = 2GEs, O4 is OAI222 = 2.5GEs, NT is NOT = 0.75GEs, MX is MUX21 = 2GEs and MXI is MUXI21 = 1.75GEs.

3.5 Implementation Results and Comparisons of Combined AES S-boxes

In this section, we will evaluate and compare the implementation results of 4 combined AES S-box circuits. All the estimate implementation areas (in GEs) in this section are computed based on the area of individual gates in the STM 65nm technology library.

3.5.1 Complexity Analysis of Combined S-boxes

Table 3.6 compares the hardware complexity analysis of four combined forward/inverse S-box schemes proposed in the contributions [21] , [70] and [54]. The estimated implementation areas (in GEs) are computed based on the area of individual gates in the STM 65nm technology library. Table 3.6 shows that Maximov bonus combined S-box is smaller than both Canright and Reyhani.

3.5.2 ASIC Implementation Results

We used VHDL coding as a design entry to the Synopsys Design Vision® for logic synthesis. All the combined S-boxes are evaluated using the STM 65nm CMOS standard-cell library. The results are collected at conservative wire load models and the CPDs are reported by the CAD tool when there is no load to the external output. The power consumption values are included as reported by the CAD tool at relaxed constraints using a clock frequency of 100 MHz.

Table 3.7 highlights the ASIC reported results for four combined S-boxes, namely Canright [21], Reyhani [70], Maximov bonus and Maximov fast [54]. The table also shows the area in terms of GEs, CPD and power consumption as reported by the CAD tool. As shown in Table 3.7, Maximov bonus combined S-box is more compact than both Canright [21]) and Reyhani [70]. Also Maximov fast combined S-box is currently the fastest combined S-box available in the literature, up to our knowledge.

Table 3.7: ASIC Implementation results of different combined S-boxes.

Combined S-box	Area		CPD	Power
	μm^2	GE	ns	μW
Canright [21]	535.6	257.5	1.344	54.41
Reyhani [70]	497.12	239	1.355	48.72
Maximov Bonus [54]	**461.76**	**222**	1.363	48.81
Maximov Fast [54]	628.16	302	**0.758**	59.07

3.6 Conclusion

In this chapter, we evaluated and compared some of of the best designs for the forward, inverse and combined AES S-boxes as well as our improved versions. The designs were verified by extensive simulation codes. The space and timing complexity analyses as well as the ASIC implementation results were used to compare the various circuits. We improved the area and delay of some previous S-boxes, by using more area efficient logic gates as NAND and NOR instead of the less area efficient AND gates, and making the necessary changes in order to keep the correct operation of the S-box. We also included the simulation results of the 5 proposed improved S-boxes, namely the proposed lightweight, the proposed low area, the proposed Fast 1, the proposed Fast 2 and the proposed Fast 3. The proposed improved S-boxes will be discussed later in more detail in Chapter 4 and Chapter 5.

Chapter 4

Improved $GF((2^4)^2)$ Forward AES S-box

4.1 Introduction

In 2018, Reyhani et al. [69] proposed two AES S-box architectures, lightweight S-box and fast S-box. The lightweight S-box was the most compact design, establishing a new milestone in the area of AES S-box design since Canright [21] introduced his S-box in 2005. The fast design was the fastest and most efficient design as compared to previous work. In this Chapter, we will suggest some modifications to the lightweight and fast S-boxes in [69] in order to improve the area and/or delay. Our improved fast S-box, Fast 3 is currently the fastest and most efficient (as measured by area $\times$ delay) S-box to date, up to our knowledge.

In section 4.2, improved blocks of the inversion circuit of the lightweight and fast S-boxes are derived. In section 4.2.1, an improved lower area and higher speed exponentiation block is derived. In section 4.2.2, a new smaller area subfield inverter will be derived and used in the two S-boxes proposed in [69]. Multiplication by a binary field element in matrix form is derived in section 4.3, and used to modify the input and output transformation matrices of the lightweight S-box. The focused-search logic minimization algorithm [69] is then used to search for a minimal circuit for the transformation matrices, in order to reduce the area of the lightweight S-box. In section 4.4, two new fast S-box architectures, Fast 1 and Fast 2, are derived and the ASIC synthesis results are obtained. The synthesis results confirm an improvement in the speed of the two new architectures. In section 4.5, the extended input transformation matrix for the fast S-box is derived and a search is conducted to find a lower area architecture, Fast 3. All the improved S-boxes are implemented in hardware and the ASIC simulation results are compared to previous work.

4.2 Improved Lightweight and Fast AES S-boxes

In this section, we will improve the area and delay of the exponentiation block and the subfield inverter proposed in [69]. The improved blocks will be used to reduce the area and delay of both the lightweight and the fast S-boxes proposed in [69].

4.2.1 Improved Exponentiation

In this section, we will derive an improved exponentiation stage for the S-box. The new exponentiation stage will have a lower complexity and a lower delay than the corresponding one in [69]. Using equation (21) from [69], we can write the output D of the exponentiation block as

$$\begin{aligned} d_0 &= (a_0b_0) \oplus (a_{12}b_{12}) \oplus a_1 \oplus b_1 \oplus (a_{02}b_{02}) \oplus (a_{13}b_{13}) \\ d_1 &= (a_1b_1) \oplus (a_{23}b_{23}) \oplus a_{13} \oplus b_{13} \oplus (a_{02}b_{02}) \oplus (a_{13}b_{13}) \\ d_2 &= (a_2b_2) \oplus (a_{03}b_{03}) \oplus a_{12} \oplus b_{12} \oplus (a_{02}b_{02}) \oplus (a_{13}b_{13}) \\ d_3 &= (a_3b_3) \oplus (a_{01}b_{01}) \oplus a_{01} \oplus b_{01} \oplus (a_{02}b_{02}) \oplus (a_{13}b_{13}) \end{aligned} \tag{4.1}$$

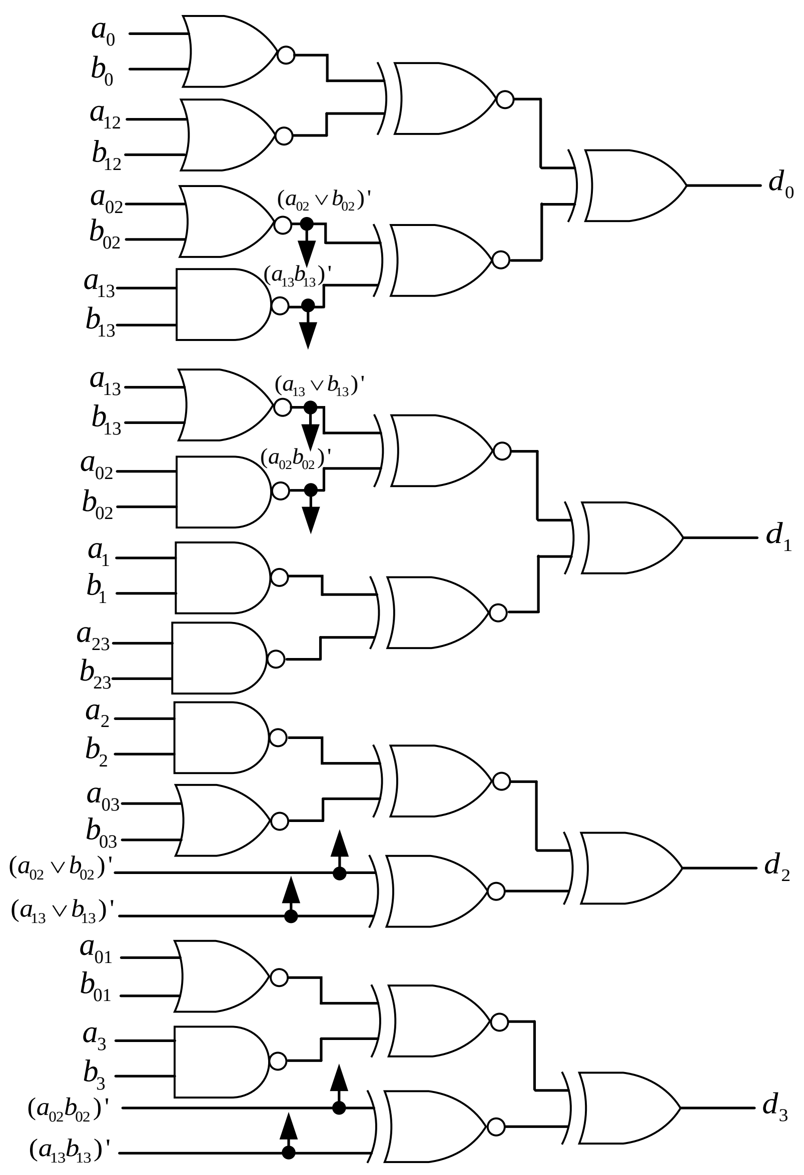


Figure 4.1: The improved exponentiation computation block for the S-box architecture.

To simplify d_0 in (4.1), we will use $(a_1 \oplus b_1) = (a_{12} \oplus b_{12} \oplus a_{02} \oplus b_{02} \oplus a_0 \oplus b_0)$, then $(a_0b_0 \oplus a_0 \oplus b_0) = (a_0 \vee b_0)$, $(a_{12}b_{12} \oplus a_{12} \oplus b_{12}) = (a_{12} \vee b_{12})$ and $(a_{02}b_{02} \oplus a_{02} \oplus b_{02}) = (a_{02} \vee b_{02})$, where $\vee$ represents an OR operation. Similarly, to simplify d_1, we will use $(a_{13}b_{13} \oplus a_{13} \oplus b_{13}) = (a_{13} \vee b_{13})$. For d_2, we will use $(a_{12} \oplus b_{12}) = (a_{13} \oplus b_{13} \oplus a_{03} \oplus b_{03} \oplus a_{02} \oplus b_{02})$. then $(a_{03}b_{03} \oplus a_{03} \oplus b_{03}) = (a_{03} \vee b_{03})$, $(a_{02}b_{02} \oplus a_{02} \oplus b_{02}) = (a_{02} \vee b_{02})$ and $(a_{13}b_3 \oplus a_{13} \oplus b_{13}) = (a_{13} \vee b_{13})$. Similar property $(a_{01}b_{01} \oplus a_{01} \oplus b_{01}) = (a_{01} \vee b_{01})$ is used for d_3. Therefore, the following formulations can be obtained for D

$$\begin{aligned} d_0 &= (a_0 \vee b_0) \oplus (a_{12} \vee b_{12}) \oplus (a_{02} \vee b_{02}) \oplus (a_{13}b_{13}) \\ d_1 &= (a_1b_1) \oplus (a_{23}b_{23}) \oplus (a_{02}b_{02}) \oplus (a_{13} \vee b_{13}) \\ d_2 &= (a_2b_2) \oplus (a_{03} \vee b_{03}) \oplus (a_{02} \vee b_{02}) \oplus (a_{13} \vee b_{13}) \\ d_3 &= (a_3b_3) \oplus (a_{01} \vee b_{01}) \oplus (a_{02}b_{02}) \oplus (a_{13}b_{13}) \end{aligned} \tag{4.2}$$

To minimize the chip area in our ASIC implementations, we use NAND and NOR gates instead of AND and OR gates, respectively. This is because NAND and NOR gates have lower chip area and delay as compared with AND and OR gates, respectively [83]. Therefore, the following formulations can be obtained from (4.2)

$$\begin{aligned} d_0 &= (a_0 \vee b_0)' \oplus (a_{12} \vee b_{12})' \oplus (a_{02} \vee b_{02})' \oplus (a_{13}b_{13})' \\ d_1 &= (a_1b_1)' \oplus (a_{23}b_{23})' \oplus (a_{02}b_{02})' \oplus (a_{13} \vee b_{13})' \\ d_2 &= (a_2b_2)' \oplus (a_{03} \vee b_{03})' \oplus (a_{02} \vee b_{02})' \oplus (a_{13} \vee b_{13})' \\ d_3 &= (a_3b_3)' \oplus (a_{01} \vee b_{01})' \oplus (a_{02}b_{02})' \oplus (a_{13}b_{13})' \end{aligned} \tag{4.3}$$

Figure 4.1 shows the circuit used for implementation of (4.3). In the first level of XOR gates, the XOR gates are changed to XNOR gates to reduce the delay. Since the inverting of both inputs of an XOR gate does not affect its operation, the XOR gates used to get $d_i, 0 \le i \le 3$, does not need to be changed to XNOR. Figure 4.1 shows that the highest delay of the improved exponentiation is $2D_X + 1D_{NOR}$, taking into account that the delay of NOR gate is slightly higher than NAND gate. Based on Figure 4.1 and equation (4.3), one can find the space and time complexities of the improved exponentiation computation block as follows. Note that the two NAND gates that generate $(a_{02}b_{02})'$ and $(a_{13}b_{13})'$ are shared. The two NOR gates that generate $(a_{02} \vee b_{02})'$ and $(a_{13} \vee b_{13})'$ are shared as well.

Proposition 4.2.1 *The improved exponentiation block consists of 12 XOR2/XNOR2 (2-input XOR/XNOR), 6 NAND2 (2-input NAND), and 6 NOR2 (2-input NOR) gates with the critical path delay of $2D_X + 1D_{NR}$, where D_X is the delay of one XOR2/XNOR2 gate and D_{NR} is the delay of one NOR2 gate.*

4.2.2 New Subfield Inverter

In this section, we will design a new subfield inverter block, in order to decrease its depth to only 3 gates and to decrease its implementation area as well. The subfield inverter block generates the inverse of its input $D = (d_0d_1d_2d_3) = \sum_{i=0}^{3} d_i\beta^{2^i}$ over $GF(2^4)$ and its output is $E = D^{-1} \in GF(2^4)$. If $e_i \in GF(2)$, is the ith, $0 \le i \le 3$, binary coordinate of E represented with respect to the ONB-I, then, $E = (e_0e_1e_2e_3) = \sum_{i=0}^{3} e_i\beta^{2^i}$. We can obtain e_i as follows.

Lemma 4.2.2

$$e_i = [(d_i'd_{i+2})'((d_id_{i+2}')' \vee (d_{i+1}d_{i+3}')')(d_{i+3}' \vee (d_{i+2} \vee d_i'd_{i+1})')]', \;\; 0 \le i \le 3 \tag{4.4}$$

Proof:

The subfield inverter is a combinational circuit which can be explained by its truth table. The truth table of the inverter is shown in Table 4.1. Then, from the e_0 column, one can represent e_0 as an OR operation ($\vee$) of the corresponding minterms, i.e.,

$$\begin{aligned} e_0 &= m_2 \vee m_3 \vee m_5 \vee m_6 \vee m_7 \vee m_{11} \vee m_{12} \vee m_{15} \\ &= (m_2 \vee m_3 \vee m_6 \vee m_7) \vee (m_3 \vee m_7 \vee m_{11} \vee m_{15}) \vee (m_5 \vee m_7) \vee m_{12}. \end{aligned} \tag{4.5}$$

One can simplify the first term of (4.5) as $m_2 \vee m_3 \vee m_6 \vee m_7 = d_0'd_2$. Also, the second terms of (4.5) can be simplified as $m_3 \vee m_7 \vee m_{11} \vee m_{15} = d_2d_3$. The third term of (4.5) $m_5 \vee m_7 = d_0'd_1d_3$. The fourth term is $m_{12} = d_0d_1d_2'd_3'$. Therefore,

$$\begin{aligned} e_0 &= d_0'd_2 \vee d_2d_3 \vee d_0'd_1d_3 \vee d_0d_1d_2'd_3'. \\ &= d_0'd_2 \vee d_0d_1d_2'd_3' \vee d_3(d_2 \vee d_0'd_1). \\ &= [(d_0'd_2 \vee d_0d_1d_2'd_3')'(d_3(d_2 \vee d_0'd_1))']' \\ &= [(d_0'd_2)'((d_0d_2')' \vee (d_1d_3')')(d_3' \vee (d_2 \vee d_0'd_1)')]' \end{aligned} \tag{4.6}$$

If the input is changed from $D = (d_0d_1d_2d_3) = \sum_{i=0}^{3} d_i\beta^{2^i}$ to its left cyclic shift $D' = D^{2^{-1}} = \sum_{i=0}^{3} d_i\beta^{2^{i-1}} = (d_1d_2d_3d_0)$, then its inverse at the output will be changed from $E = (e_0e_1e_2e_3) = D^{-1}$ to $E' = (D')^{-1} = (D^{2^{-1}})^{-1} = (D^{-1})^{2^{-1}} = (e_1e_2e_3e_0)$. In other words, the cyclic shifts at the inverse input will result in the same cyclic shifts at its output. Let us denote the 0th coordinate of $E = (e_0e_1e_2e_3)$ as the function $e_0 = I(d_0d_1d_2d_3)$ in terms of its input $D = (d_0d_1d_2d_3)$ (see the formulation presented in (4.6) for the detail of the Boolean function I). Then, the bit 0 of $E' = (e_1e_2e_3e_0)$, i.e., e_1, is obtained when the input is $D' = (d_1d_2d_3d_0)$. Therefore, one can find $e_1 = I(d_1d_2d_3d_0)$ and so other coordinates of $E = D^{-1}$, say e_i, for $1 \le i \le 3$, can be found similarly by adding (i modulo 4) with D, i.e., $e_i = I(d_id_{i+1}d_{i+2}d_{i+3})$. This is similar to to the idea proposed in [50, 82, 65] for multiplication operation. Replacing $D = (d_0d_1d_2d_3)$ by $D = (d_id_{i+1}d_{i+2}d_{i+3})$ concludes the proof.

Table 4.1: The truth table of the inverter over $GF(2^4)$ generated by the ONB-I [69].

$d_0d_1d_2d_3$	$e_0e_1e_2e_3$	$d_0d_1d_2d_3$	$e_0e_1e_2e_3$
0000	0000	1000	0010
0001	0100	1001	0111
0010	1000	1010	0101
0011	1110	1011	1100
0100	0001	1100	1011
0101	1010	1101	0110
0110	1101	1110	0011
0111	1001	1111	1111

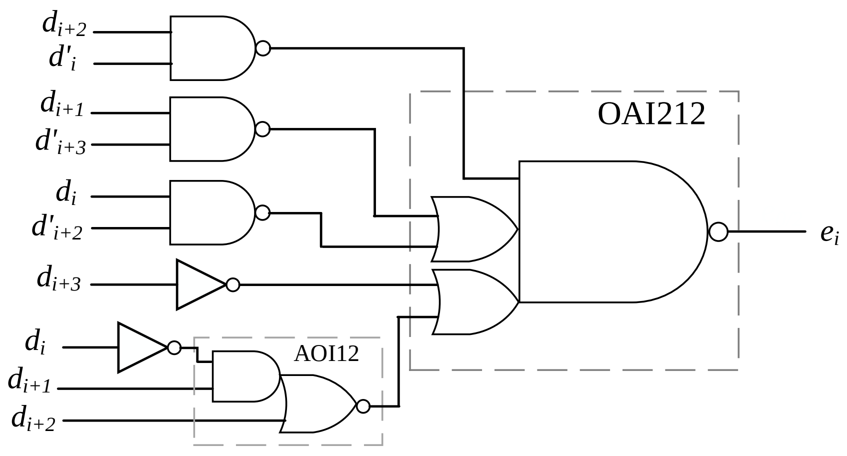


Figure 4.2: The modified subfield inverter circuit.

As shown in Figure 4.2, e_0 in (4.6) can be implemented using an OAI212 gate (two 2-input OR into 3-input NAND), which has five inputs. The first input is from the NAND2 gate $(d'_0d_2)'$, the second and third inputs from the two NAND2 gates that generates $(d_0d'_2)'$ and $(d_1d'_3)'$. The fourth one comes from d'_3, and the last one from the AOI12 gate (2-input AND into 2-input NOR) that generates $(d_2 \vee d'_0d_1)'$. By sharing the NAND2 gates between the different e_i's, only 4 NAND2 gates are needed. Those NAND2 gates will generate $(d'_0d_2)', (d_0d'_2)', (d_1d'_3)', (d'_1d_3)'$. Hence, we can conclude the following for the subfield inverter.

Proposition 4.2.3 *The improved subfield inverter block consists of 4 OAI212 (two 2-input OR into 3-input NAND), 4 AOI12 (2-input AND into 2-input NOR), 4 NAND2 (2-input NAND), and 4 NOT gates with the critical path delay of $D_{AOI12} + D_{OAI212} + D_{NOT}$ where D_{AOI12} is the delay of one AOI12 gate, D_{OAI212} is the delay of one OAI212 gate, and D_{NOT} is the delay of a NOT gate.*

4.2.3 Space and Time Complexity Analyses

The space and time complexities of the improved exponentiation block and the improved subfield inverter, along with the input and output transformation blocks and the output multipliers block as presented in [69], are summarized in Tables 4.2. This table also shows the space and time complexities of the improved $GF((2^4)^2)$ inversion and the entire improved AES lightweight and fast S-box architectures. The corresponding GEs of all the blocks are also presented. GE is the chip area in terms of an equivalent of 2-input NAND gates. The provided GEs are based on the used STM 65nm CMOS technology.

4.2.4 ASIC Synthesis Results and Comparisons

The ASIC synthesis results are obtained using the Synopsys Design Vision®. The ASIC synthesis results for the improved exponentiation and the improved subfield inverter presented in section

Table 4.2: Space and time complexities for different blocks and the $GF((2^4)^2)$ inversion of the entire improved S-boxes.

Block/ Target	Space Complexity[*]							Time Complexity
	X	ND	NR	AOI12	OAI212	NT	GE	
Input Transformation Block								
Light.	19						38	$5D_X$
Fast	24						48	$3D_X$
$GF((2^4)^2)$ Inversion								
Imp. Exp.	12	6	6				36	$2D_X + 1D_{NR}$
Imp. Inv.		4		4	4	4	20	$1D_{AOI12} + 1D_{OAI212} + 1D_{NT}$
Mult.	16	20					52	$2D_X + 1D_{ND}$
Output Transformation Block								
Light.	16						32	$6D_X$
Fast	21						42	$3D_X$
Total Complexity of Improved S-Box								
Light.	63	30	6	4	4	4	178	$15D_X + 1D_{NR} + 1D_{ND} + 1D_{AOI12} + 1D_{OAI212} + 1D_{NT}$
Fast	73	30	6	4	4	4	198	$10D_X + 1D_{NR} + 1D_{ND} + 1D_{AOI12} + 1D_{OAI212} + 1D_{NT}$

[*]All GEs values are estimated using STM 65nm technology where X is XOR2/XNOR2 = 2GEs, ND is NAND2 = 1GE, NR is NOR2 = 1GE, AOI12 is AND2 into NOR2 = 1.25GEs, OAI212 = 2 OR2 into NAND3 = 2GEs, NT is NOT = 0.75GEs.

Table 4.3: ASIC synthesis results for the improved exponentiation and the improved subfield inverter.

Block	Area		Delay	Power
	μm^2	GE	ns	μW
Improved Exp.	74.88	36	0.106154	3.1677
New Subfield Inv.	41.6	20	0.082485	1.5242

4.2.1 and 4.2.2, respectively are shown in Table 4.3. The technology library used was STM 65nm CMOS standard library and the CORE65LPSVT standard cell library which is optimized for low power applications. The area, delay and power for all the considered blocks are generated by the CAD tool with relaxed constraints at a clock frequency of 100 MHz.

We used the improved exponentiation and the improved subfield inverter for both the lightweight and the fast S-boxes proposed in [69] and the ASIC synthesis results are summarized in Table 4.4. From Table 4.4, the improved lightweight S-box has 4.25 GEs less area than [69], an improvement of 2%. It also leads to a reduction in the critical path delay of the improved lightweight S-box. For the fast S-box design, the area is 10 GE less than [69], an improvement of 5%, with approximately the same delay. The efficiency, measured as area × delay of the improved fast S-box is increased by 5%.

Table 4.4: ASIC synthesis results for the two S-boxes in [69] using the improved exponentiation and improved subfield inverter.

Architecture	Area		Delay	Power
	μm^2	GE	ns	μW
Lightweight S-box [69]	379.08	182.25	1.197698	38.085
Improved Lightweight S-box	**370.24**	**178**	**1.102665**	37.549
Fast S-box [69]	432.64	208	0.779697	42.750
Improved Fast S-box	**411.84**	**198**	**0.776748**	40.476

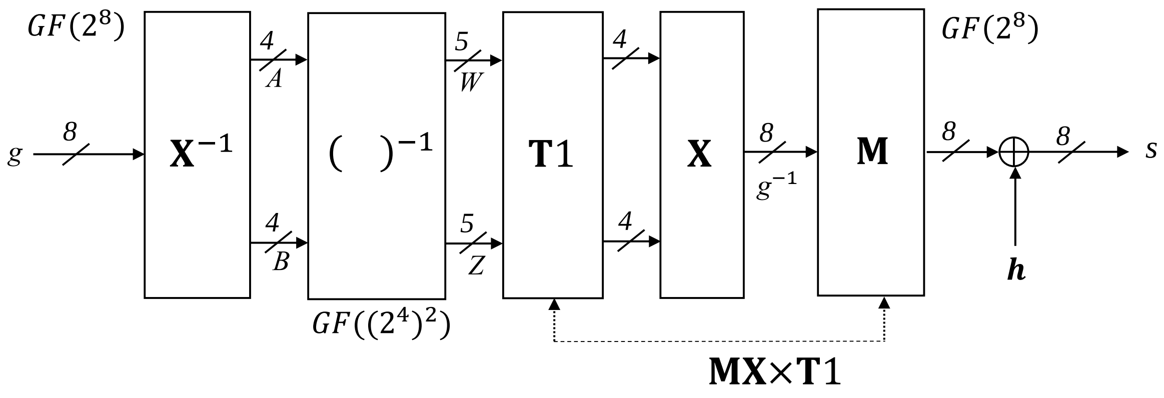


Figure 4.3: Input and output transformations used in the S-box of [69].

4.3 Additional Transformation Matrices

In this section, we will modify the input to the lightweight S-box in [69], by multiplying it by a binary field element η that ranges from $(1-255)$ and then check if there will be any improvement in the implementation area of the input and output transformation matrices of the S-box. By multiplying by η, we are increasing the variety of transformation matrices by 255 times, which might lead to a better area and/or delay mappings. This idea was first proposed in [54], however, we will derive the necessary formulations and modify the S-box architecture in [69] based on those formulas. We will also analyze, using Matlab®, all the results of applying the focused-search logic minimization algorithm [69] in order to find a better architecture for the fast S-box.

4.3.1 Multiplying by a Binary Field Element η

In this section, we will derive the matrix form of multiplying the input to the S-box g by an 8-bit binary field element η.

As show in Figure 4.3, the input to the S-box in [69] is g, where g is an element in $GF(2^8)$ generated by the irreducible polynomial $q(x) = x^8 + x^4 + x^3 + x + 1$. If α is a root of $q(x)$, then $g = (g_7, \cdots, g_1, g_0) \in GF(2^8)$ is represented in Polynomial Basis (PB) representation as $g = \sum_{i=0}^{7} g_i\alpha^i = \boldsymbol{\alpha}^{tr}\mathbf{g}$, where $\boldsymbol{\alpha} = [\alpha^7, \cdots, \alpha^1, 1]^{tr}$, $\mathbf{g} = [g_7, \cdots, g_1, g_0]^{tr}$and tr denotes the transposition.

We will modify the input to the S-box, by multiplying it by an 8-bit binary field element η that ranges from $(1-255)$, and then study its effect on the logic minimization results of the new input and output transformation matrices. To find the matrix form of this multiplication, the binary field element, $\eta = (\eta_7, \cdots, \eta_1, \eta_0) \in GF(2^8)$ is represented in polynomial basis as $\eta = \sum_{i=0}^{7} \eta_i\alpha^i = \boldsymbol{\alpha}^{tr}\boldsymbol{\eta}$, $\boldsymbol{\eta} = [\eta_7, \cdots, \eta_1, \eta_0]^{tr}$. We will use **Theorem 1** from [67] to find the matrix form of this multiplication. In [67], they used the polynomial basis representation $\boldsymbol{\alpha''} = [\alpha^0, \cdots, \alpha^6, \alpha^7]^{tr}$, but we will use the notation $\boldsymbol{\alpha} = [\alpha^7, \cdots, \alpha^1, 1]^{tr}$. i.e. we are starting with the Most Significant Bit (MSB). For example, in our notation {01100011} identifies the specific finite field element $\alpha^6 + \alpha^5 + \alpha + 1$, while in [67] it identifies the element $\alpha + \alpha^2 + \alpha^6 + \alpha^7$. As a result of the different notation adopted for the field elements, our matrices will be slightly different than [67].

Theorem 4.3.1 *[67]*

For AES irreducible polynomial $q(x) = x^8 + x^4 + x^3 + x + 1$, let $c = \sum_{i=0}^{7} c_i\alpha^i = \boldsymbol{\alpha}^{tr}\mathbf{c}$ be the multiplication of g and $\eta \in GF(2^8)$. Then, the coordinates of c can be obtained from

$$\mathbf{c} = [c_7, , \cdots, c_1, c_0]^{tr} = (\mathbf{Q}^{tr}\mathbf{L} + \mathbf{U})\mathbf{g} = \mathbf{C}\mathbf{g}$$

where $\mathbf{g} = [g_7, \cdots, g_1, g_0]^{tr}$,

$$\mathbf{L} = \begin{bmatrix} \eta_7 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \eta_6 & \eta_7 & 0 & 0 & 0 & 0 & 0 & 0 \\ \eta_5 & \eta_6 & \eta_7 & 0 & 0 & 0 & 0 & 0 \\ \eta_4 & \eta_5 & \eta_6 & \eta_7 & 0 & 0 & 0 & 0 \\ \eta_3 & \eta_4 & \eta_5 & \eta_6 & \eta_7 & 0 & 0 & 0 \\ \eta_2 & \eta_3 & \eta_4 & \eta_5 & \eta_6 & \eta_7 & 0 & 0 \\ \eta_1 & \eta_2 & \eta_3 & \eta_4 & \eta_5 & \eta_6 & \eta_7 & 0 \end{bmatrix},$$

$$\mathbf{U} = \begin{bmatrix} \eta_0 & \eta_1 & \eta_2 & \eta_3 & \eta_4 & \eta_5 & \eta_6 & \eta_7 \\ 0 & \eta_0 & \eta_1 & \eta_2 & \eta_3 & \eta_4 & \eta_5 & \eta_6 \\ 0 & 0 & \eta_0 & \eta_1 & \eta_2 & \eta_3 & \eta_4 & \eta_5 \\ 0 & 0 & 0 & \eta_0 & \eta_1 & \eta_2 & \eta_3 & \eta_4 \\ 0 & 0 & 0 & 0 & \eta_0 & \eta_1 & \eta_2 & \eta_3 \\ 0 & 0 & 0 & 0 & 0 & \eta_0 & \eta_1 & \eta_2 \\ 0 & 0 & 0 & 0 & 0 & 0 & \eta_0 & \eta_1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \eta_0 \end{bmatrix},$$

and the 7×8 *binary reduction matrix,* $\mathbf{Q}$ *is obtained from*

$$[\alpha^{14}, \cdots, \alpha^9, \alpha^8]^{tr} \equiv \mathbf{Q}[\alpha^7, \cdots, \alpha^1, 1]^{tr} (mod\ q(x)).$$

Proof:

First, the reduction matrix $\mathbf{Q}$ is the 7×8 binary matrix obtained from the following equation

$$\boldsymbol{\alpha}^{\uparrow} \equiv \mathbf{Q}\boldsymbol{\alpha}(mod\ q(x)) \tag{4.7}$$

where $\boldsymbol{\alpha}^{\uparrow} = [\alpha^{14}, \cdots, \alpha^9, \alpha^8]^{tr}$

The reduction matrix can be obtained by writing the representation of $\alpha^{14}, \cdots, \alpha^9, \alpha^8$ in polynomial basis using the AES irreducible polynomial $q(x)$ as

$$\mathbf{Q} = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \end{bmatrix} \tag{4.8}$$

The multiplication of g and η can be represented in PB as

$$\begin{gathered} p = (\textstyle\sum_{j=0}^{7} \eta_i \alpha^j)(\sum_{i=0}^{7} g_i \alpha^i) = \sum_{k=0}^{14} p_k \alpha^k \\ p = (\eta_7\alpha^7 + \eta_6\alpha^6 + \eta_5\alpha^5 + \eta_4\alpha^4 + \eta_3\alpha^3 + \eta_2\alpha^2 + \eta_1\alpha^1 + \eta_0\alpha^0) \\ \times(g_7\alpha^7 + g_6\alpha^6 + g_5\alpha^5 + g_4\alpha^4 + g_3\alpha^3 + g_2\alpha^2 + g_1\alpha^1 + g_0\alpha^0) \end{gathered} \tag{4.9}$$

where $p_k = \sum_{i+j=k} \eta_i g_j,\ 0 \le i, j \le 7,\ 0 \le k \le 14$

Then the multiplication, c, of η and g can be written as $c = \sum_{i=0}^{7} c_i \alpha^i = p\ mod\ (q(x))$, where p can be written as

$$p = \boldsymbol{\alpha}^{\Uparrow^{tr}} \mathbf{p} \tag{4.10}$$

where $\boldsymbol{\alpha}^{\Uparrow} = [\alpha^{14}, \cdots, \alpha, 1]^{tr} = \begin{bmatrix} \boldsymbol{\alpha}^{\uparrow} \\ \boldsymbol{\alpha} \end{bmatrix}$ and $\mathbf{p} = [p_{14}, \cdots, p_{1,}, p_0]^{tr}$

Using (4.7), we can write

$$\boldsymbol{\alpha}^{\Uparrow} = \begin{bmatrix} \mathbf{Q} \\ \mathbf{I}_8 \end{bmatrix} \boldsymbol{\alpha} \tag{4.11}$$

, where $\mathbf{I}_8$ is the 8×8 unity matrix.

To simplify the expression in (4.9), we can define the following two Toeplitz matrices **L** and **U**, where **L** is an 7×8 lower triangular matrix and **U** is an 8×8 upper triangular matrix

$$\mathbf{L} = \begin{bmatrix} \eta_7 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \eta_6 & \eta_7 & 0 & 0 & 0 & 0 & 0 & 0 \\ \eta_5 & \eta_6 & \eta_7 & 0 & 0 & 0 & 0 & 0 \\ \eta_4 & \eta_5 & \eta_6 & \eta_7 & 0 & 0 & 0 & 0 \\ \eta_3 & \eta_4 & \eta_5 & \eta_6 & \eta_7 & 0 & 0 & 0 \\ \eta_2 & \eta_3 & \eta_4 & \eta_5 & \eta_6 & \eta_7 & 0 & 0 \\ \eta_1 & \eta_2 & \eta_3 & \eta_4 & \eta_5 & \eta_6 & \eta_7 & 0 \end{bmatrix} \tag{4.12}$$

$$\mathbf{U} = \begin{bmatrix} \eta_0 & \eta_1 & \eta_2 & \eta_3 & \eta_4 & \eta_5 & \eta_6 & \eta_7 \\ 0 & \eta_0 & \eta_1 & \eta_2 & \eta_3 & \eta_4 & \eta_5 & \eta_6 \\ 0 & 0 & \eta_0 & \eta_1 & \eta_2 & \eta_3 & \eta_4 & \eta_5 \\ 0 & 0 & 0 & \eta_0 & \eta_1 & \eta_2 & \eta_3 & \eta_4 \\ 0 & 0 & 0 & 0 & \eta_0 & \eta_1 & \eta_2 & \eta_3 \\ 0 & 0 & 0 & 0 & 0 & \eta_0 & \eta_1 & \eta_2 \\ 0 & 0 & 0 & 0 & 0 & 0 & \eta_0 & \eta_1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \eta_0 \end{bmatrix} \tag{4.13}$$

Using **L** and **U**, we can define the following two vectors, which are functions of g and η

$\mathbf{d} = \mathbf{Lg}$

$\mathbf{e} = \mathbf{Ug}$

Since $p_k = \sum_{i+j=k} \eta_i g_j,\ 0 \leq i, j \leq 7,\ 0 \leq k \leq 14$, then $d_k = p_k,\ 0 \leq k \leq 7$, and $e_l = p_{l+m},\ 0 \leq l \leq 7$ then

$\mathbf{p} = \begin{bmatrix} \mathbf{d} \\ \mathbf{e} \end{bmatrix} = \begin{bmatrix} \mathbf{L} \\ \mathbf{U} \end{bmatrix} \mathbf{g}$

and using (4.10), (4.11), we can write that

$c = p \; mod \; (q(x)) = \left(\begin{bmatrix} \mathbf{Q} \\ \mathbf{I}_8 \end{bmatrix} \boldsymbol{\alpha} \right)^{tr} \begin{bmatrix} \mathbf{L} \\ \mathbf{U} \end{bmatrix} \mathbf{g} = \boldsymbol{\alpha}^{tr} \left(\begin{bmatrix} \mathbf{Q}^{tr} & \mathbf{I}_8 \end{bmatrix} \begin{bmatrix} \mathbf{L} \\ \mathbf{U} \end{bmatrix} \right) \mathbf{g}$

Then $c = \boldsymbol{\alpha}^{tr}(\mathbf{Q}^{tr}\mathbf{L} + \mathbf{U})\mathbf{g}$

Using (4.8), (4.12)

$$\mathbf{Q}^{tr}\mathbf{L} = \begin{bmatrix} \eta_{7,5,4} & \eta_{6,5} & \eta_{7,6} & \eta_7 & 0 & 0 & 0 & 0 \\ \eta_{6,4,3} & \eta_{7,5,4} & \eta_{6,5} & \eta_{7,6} & \eta_7 & 0 & 0 & 0 \\ \eta_{5,3,2} & \eta_{6,4,3} & \eta_{7,5,4} & \eta_{6,5} & \eta_{7,6} & \eta_7 & 0 & 0 \\ \eta_{7,4,2,1} & \eta_{5,3,2} & \eta_{6,4,3} & \eta_{7,5,4} & \eta_{6,5} & \eta_{7,6} & \eta_7 & 0 \\ \eta_{7,6,5,4,3,1} & \eta_{7,6,5,4,2} & \eta_{7,6,5,3} & \eta_{7,6,4} & \eta_{7,5} & \eta_6 & \eta_7 & 0 \\ \eta_{6,3,2} & \eta_{7,4,3} & \eta_{5,4} & \eta_{6,5} & \eta_{7,6} & \eta_7 & 0 & 0 \\ \eta_{7,5,2,1} & \eta_{6,3,2} & \eta_{7,4,3} & \eta_{5,4} & \eta_{6,5} & \eta_{7,6} & \eta_7 & 0 \\ \eta_{6,5,1} & \eta_{7,6,2} & \eta_{7,3} & \eta_4 & \eta_5 & \eta_6 & \eta_7 & 0 \end{bmatrix} \tag{4.14}$$

From (4.13) and (4.14), the multiplication $\mathbf{c} = (\mathbf{Q}^{tr}\mathbf{L} + \mathbf{U})\mathbf{g}$, of a binary field element η and the input to the S-box g can be written in matrix form as

$$\mathbf{c} \quad = \quad \begin{bmatrix} \eta_{7,5,4,0} & \eta_{6,5,1} & \eta_{7,6,2} & \eta_{7,3} & \eta_4 & \eta_5 & \eta_6 & \eta_7 \\ \eta_{6,4,3} & \eta_{7,5,4,0} & \eta_{6,5,1} & \eta_{7,6,2} & \eta_{7,3} & \eta_4 & \eta_5 & \eta_6 \\ \eta_{5,3,2} & \eta_{6,4,3} & \eta_{7,5,4,0} & \eta_{6,5,1} & \eta_{7,6,2} & \eta_{7,3} & \eta_4 & \eta_5 \\ \eta_{7,4,2,1} & \eta_{5,3,2} & \eta_{6,4,3} & \eta_{7,5,4,0} & \eta_{6,5,1} & \eta_{7,6,2} & \eta_{7,3} & \eta_4 \\ \eta_{7,6,5,4,3,1} & \eta_{7,6,5,4,2} & \eta_{7,6,5,3} & \eta_{7,6,4} & \eta_{7,5,0} & \eta_{6,1} & \eta_{7,2} & \eta_3 \\ \eta_{6,3,2} & \eta_{7,4,3} & \eta_{5,4} & \eta_{6,5} & \eta_{7,6} & \eta_{7,0} & \eta_1 & \eta_2 \\ \eta_{7,5,2,1} & \eta_{6,3,2} & \eta_{7,4,3} & \eta_{5,4} & \eta_{6,5} & \eta_{7,6} & \eta_{7,0} & \eta_1 \\ \eta_{6,5,1} & \eta_{7,6,2} & \eta_{7,3} & \eta_4 & \eta_5 & \eta_6 & \eta_7 & \eta_0 \end{bmatrix} \mathbf{g} = \mathbf{Cg} \tag{4.15}$$

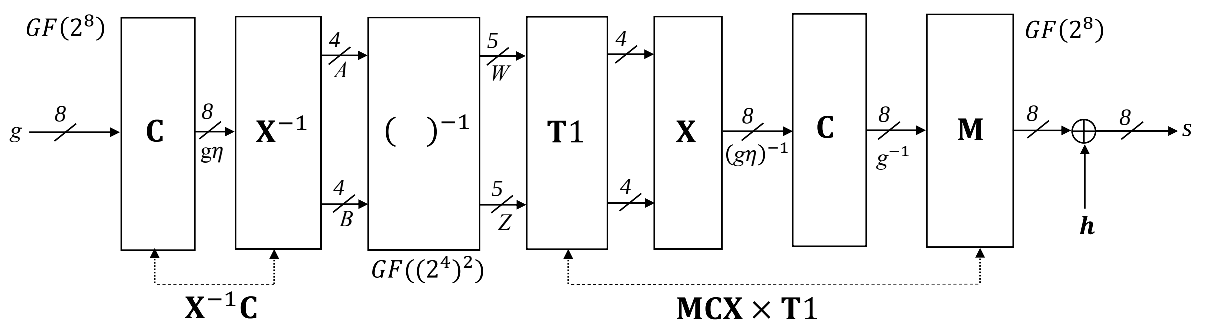


Figure 4.4: The modified input and output transformations.

4.3.2 Modified AES S-box Architecture

In this section, we will include the matrix **C** derived in (4.15) in the S-box architecture shown in Figure 4.3. To study the effect of multiplying by **C**, we will modify Figure 4.3 by adding the matrix **C** before the isomorphic mapping block $\mathbf{X}^{-1}$ as shown in Figure 4.4. To reflect the effect of adding the block **C** to the input, we add the block **C** at the output between the inverse isomorphic mapping block **X** and the affine transformation matrix **M**. The input transformation block accepts an 8-bit element g from the $GF(2^8)$ field, and generates two 4-bit field elements A and B, and the mod-2 addition between every two bits in each of the two elements of A and B, i.e., A_{jk} and B_{jk} , where A_{jk} is a set that contains $a_j \oplus a_k$, and B_{jk} contains $b_j \oplus b_k$, both for $0 \le j, k \le 3$, and $j \neq k$. The output transformation block accepts the result of the two subfield multipliers; W and Z and generates the corresponding element s, i.e., the S-box output, in the AES $GF(2^8)$ field.

As a result, the modified 20×8 input transformation matrix, $\mathbf{T}_{in}$ can be defined as

$$\mathbf{T}_{in} = \begin{bmatrix} \mathbf{X}^{-1}\mathbf{C} \\ \mathbf{a}_{ij}^{tr} \\ \mathbf{b}_{ij}^{tr} \end{bmatrix} \text{for } 0 \le i, j \le 3, i \neq j \tag{4.16}$$

where the rows of $(\mathbf{X}^{-1}\mathbf{C})$ generate the coordinates of A and B as $a_i = \mathbf{a}_i^{tr}\mathbf{g}$ and $b_i = \mathbf{b}_i^{tr}\mathbf{g}$, respectively. The next 12 rows of $\mathbf{T}_{in}$ generate $a_{ij} = a_i \oplus a_j = (\mathbf{a}_i^{tr} \oplus \mathbf{a}_j^{tr})\mathbf{g}$ and $b_{ij} = b_i \oplus b_j = (\mathbf{b}_i^{tr} \oplus \mathbf{b}_j^{tr})\mathbf{g}$. At the output of the S-box, the modified output transformation matrix, $\mathbf{T}_{out}$ is used to get the output, s of the S-box as

$$\mathbf{s} = \mathbf{T}_{out} \times \mathbf{f}_{10} \oplus \mathbf{h} \tag{4.17}$$

where $\mathbf{f}_{10} = [w_0w_1w_2w_3w_4z_0z_1z_2z_3z_4]^{tr}$ is the output of composite field inversion, represented in the RNB, $\mathbf{h} = [01100011]^{tr}$and

$$\mathbf{T}_{out} = \mathbf{MCX} \times \mathbf{T1} \tag{4.18}$$

where **T1** is the 8×10 matrix that converts from the Redundant Normal Basis (RNB) representations of $W = \sum_{i=0}^{3} w_i\beta^{2^i} + w_4$ and $Z = \sum_{i=0}^{3} z_i\beta^{2^i} + z_4$ at the outputs of the $GF((2^4)^2)$ inverter back to the non-redundant NB representations.

4.3.3 Matlab® Results

In this section, we will code equations (4.16) and (4.18) in Matlab®. We will use 255 value for η to derive 255 new **C** matrices according to (4.15). Note that the case $\eta = 1$ is already done in [69] and corresponds to Figure 4.3 . The 255 new **C** matrices will be used along with 8 different $\mathbf{X}^{-1}$ matrices. That means a total of 2040 $\mathbf{T}_{in}$ and 2040 $\mathbf{T}_{out}$ transformation matrices will be derived. We will use the focused-search logic minimization algorithm [69] to check the number of XOR2 gates required for the 2040 $\mathbf{T}_{in}$ matrices and the maximum delay associated with each one. For the

2040 $\mathbf{T}_{out}$, we will only apply the logic minimization algorithm to certain matrices (19 matrices), that correspond to a low complexity and low delay $\mathbf{T}_{in}$.

The input transformation matrix $\mathbf{T}_{in}$ in (4.16) depends on the binary transformation matrix $\mathbf{X}^{-1}$ that is used for isomorphic mapping between the fields $GF(2^8)$ and $GF((2^4)^2)$. In Table 1 [69], 32 binary transformation matrices $\mathbf{X}^{-1}$ can be used to convert from $GF(2^8)$ to $GF((2^4)^2)$. Those 32 matrices were optimized by the focused-search logic minimization algorithm and they selected the case that results in the minimum gate-count for their lightweight S-box. For the input transformation matrix, they chose an input matrix that requires 19 XOR2 gates and a propagation delay of $5D_X$. For the output transformation, they chose an output matrix that requires 16 gates with a delay of 6 gates ($6D_X$).

We noticed that from the 32 binary transformation matrices $\mathbf{X}^{-1}$, only 8 are different. The other 24 matrices are similar, where the rows of $\mathbf{X}^{-1}$ are shuffled versions of each other. Running the logic minimization algorithm for those 28 similar $\mathbf{X}^{-1}$ matrices will give the same number of XOR gates for each matrix, i.e. the same hardware complexity. As a result, we only tested the 8 different $\mathbf{X}^{-1}$ matrices in our new architecture in Figure 4.4.

For each $\mathbf{X}^{-1}$, we derived, using Matlab®, 8 different input transformation matrices $\mathbf{T}_{in}$ in (4.16) and 8 different output transformation matrices $\mathbf{T}_{out}$ in (4.18). Then for each $\mathbf{X}^{-1}$, we multiply by $\eta = (1 - 255)$. We applied the focused-search logic minimization algorithm [69] on each $\mathbf{T}_{in}$, for all values of the arbitrary binary field element η, a total of $8 \times 255 = 2040$, in order to select the smallest area one. The results of the Matlab® simulations are shown in Figures A.1, A.2, A.3, A.4, A.5, A.6, A.7 and A.8. Those figures show the number of XOR2 gates used to implement each $\mathbf{T}_{in}$ as a function of η, as well as the maximum delay associated with each matrix. We can conclude that for the 2040 input transformation matrices, we did not find a matrix with a lower number of XOR gates than the 19 gates used in [69], however we did find some matrices with a delay less than $5D_X$.

For $\mathbf{T}_{out}$, it has 10 input signals, i.e. 10 columns, while $\mathbf{T}_{in}$ has only 8. The search time of the logic minimization algorithm depends exponentially on the number of input signals and will be much larger for $\mathbf{T}_{out}$ than $\mathbf{T}_{in}$. To find a minimal solution for the output matrix, we only evaluate $\mathbf{T}_{out}$ matrices for the specific cases of $\mathbf{X}^{-1}$ and η that corresponds to an already low complexity and low delay $\mathbf{T}_{in}$. Each $\mathbf{X}^{-1}$ is associated with a specific 4-bit $GF(2^4)$ element called v and a Generator # $(1 - 4)$. v is the 4-bit field element in $GF(2^4)$ chosen so that the polynomial $p(y) = y^2 + y + v$ is irreducible over $GF(2^4)$. The generator, can be used to generate all the non-zero field elements by raising it to a positive integer power $i \in [0, n-2]$, where n is the size of the field. By choosing η, v and Generator #, a specific $\mathbf{T}_{out}$ will be optimized. We choose the cases where the number of XOR2 gates required for $\mathbf{T}_{in}$ is equal to 19 gates, and where the maximum delay is equal to or less than $5D_X$. We applied the logic minimization algorithm to the corresponding $\mathbf{T}_{out}$, and the results are summarized in Table 4.5.

For all the 19 cases in Table 4.5, we could not find an output matrix where the number of gates is less than 16, but we could find some with a maximum delay less than $6D_X$.

We can conclude from the Matlab® simulations for $\mathbf{T}_{in}$ and $\mathbf{T}_{out}$ that multiplication by a binary field element $\eta = (1 - 255)$, did not lead to any improvement in the area of the input and output transformations of the $GF((2^4)^2)$ S-box. However it might lead to an improvement in the speed of the lightweight S-box as we will explain in section 4.4.

4.4 Improved Fast AES S-box Architecture

In this section, we will derive the necessary formulas for modifying the architecture of the fast S-box [69] in order to improve its speed. We will also use the method of logical effort [74] to find

Table 4.5: The number of XOR2 gates and the maximum delay of specific $\mathbf{T}_{out}$ matrices.

v	Generator #	η	# XOR2	Max. Delay
1000[69]	4	1	16	$6D_X$
1000	4	12	16	$8D_X$
1000	4	80	16	$8D_X$
1000	4	176	16	$8D_X$
1000	4	237	16	$8D_X$
1000	3	42	17	$5D_X$
1000	3	74	18	$6D_X$
1000	3	76	17	$5D_X$
1000	3	85	18	$6D_X$
1000	3	117	18	$6D_X$
1000	3	125	17	$5D_X$
1000	3	187	18	$6D_X$
1000	3	209	18	$6D_X$
1000	3	227	17	$5D_X$
1000	3	248	17	$5D_X$
0111	4	15	18	$5D_X$
0111	4	29	18	$5D_X$
0111	4	68	18	$5D_X$
0111	4	156	18	$5D_X$
0111	4	202	18	$5D_X$

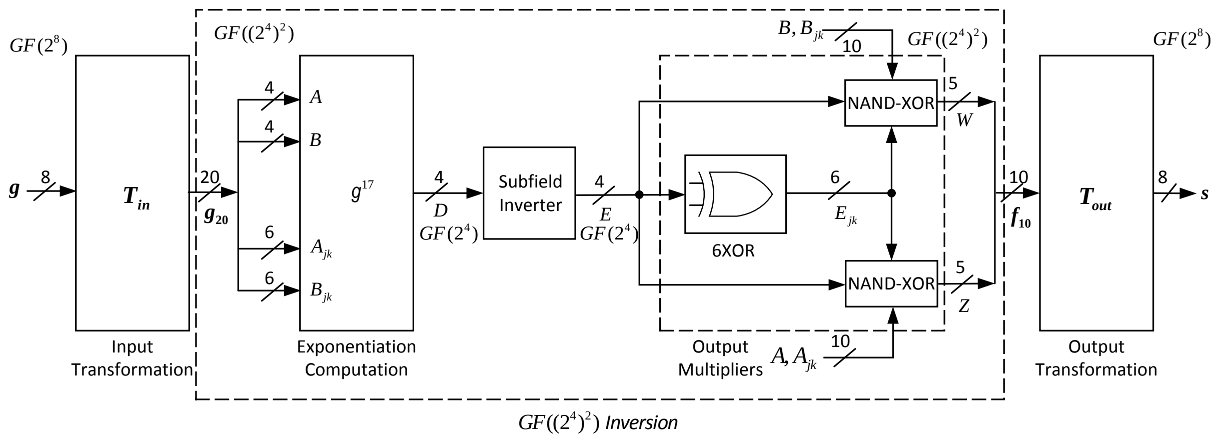


Figure 4.5: The S-box architecture in [69].

the minimum possible critical path delay of the S-box. The method of logical effort uses a linear delay model and it could provide a standard method to calculate the critical path delay of the S-box independent of the technology library used for synthesis.

Figure 4.5 shows the S-box architecture used in [69]. By eliminating the output transformation matrix $\mathbf{T}_{out}$ and the output multipliers block, we can reduce the delay of the S-box. This idea was first proposed in [54], however we will derive the formulas for a different S-box architecture.

In the following section, we will derive the formulations for the modified fast S-box for two cases. The first one uses the transformation matrices of [69], while the second one uses the result of section 4.3.3 to derive a faster and a lower area S-box, based on the reduced maximum delay values obtained from the simulations.

4.4.1 The Modified Fast S-box, Fast 1

The first modified fast S-box architecture, Fast 1 is shown in Figure 4.6. A new 28-bit signal $\mathbf{g}_{28}$ is computed in parallel to the $GF((2^4)^2)$ inversion. A 32NAND+8XOR4 block is needed, which will add a hardware complexity of 32 NAND2 and 24 XOR2 gates. The modified fast S-box is based on the S-box architecture in Figure 4.5. Fast 1 architecture corresponds to the case $v = 1000$, Generator #4 and the binary field element is simply $\eta = 1$.

The 8-bit input $\mathbf{g}$ is applied to the 20×8 input transformation matrix, $\mathbf{T}_{in}$ to produce the 20-bit signal $\mathbf{g}_{20}$ as

$$\mathbf{g}_{20} = \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ b_0 \\ b_1 \\ b_2 \\ b_3 \\ a_{01} \\ a_{02} \\ a_{03} \\ a_{12} \\ a_{13} \\ a_{23} \\ b_{01} \\ b_{02} \\ b_{03} \\ b_{12} \\ b_{13} \\ b_{23} \end{bmatrix} = \mathbf{T}_{in}\mathbf{g} = \begin{bmatrix} 0&0&0&0&1&1&0&1 \\ 0&0&0&0&0&1&0&1 \\ 1&1&1&0&0&0&1&1 \\ 0&0&1&0&0&0&0&1 \\ 0&1&1&1&1&1&0&1 \\ 1&0&1&0&0&1&1&1 \\ 0&1&0&0&1&1&0&1 \\ 1&1&1&1&1&1&0&1 \\ 0&0&0&0&1&0&0&0 \\ 1&1&1&0&1&1&1&0 \\ 0&0&1&0&1&1&0&0 \\ 1&1&1&0&0&1&1&0 \\ 0&0&1&0&0&1&0&0 \\ 1&1&0&0&0&0&1&0 \\ 1&1&0&1&1&0&1&0 \\ 0&0&1&1&0&0&0&0 \\ 1&0&0&0&0&0&0&0 \\ 1&1&1&0&1&0&1&0 \\ 0&1&0&1&1&0&1&0 \\ 1&0&1&1&0&0&0&0 \end{bmatrix} \begin{bmatrix} g_7 \\ g_6 \\ g_5 \\ g_4 \\ g_3 \\ g_2 \\ g_1 \\ g_0 \end{bmatrix}, \tag{4.19}$$

The 4-bit signal output from the exponentiation block, D is applied to the subfield inverter block. The output from the subfield inverter, E is multiplied by a set of signals produced by the input transformation matrix, $\mathbf{g}_{20}$ to derive the 10-bit signal $\mathbf{f}_{10}$. The 8×10 output transformation matrix $\mathbf{T}_{out}$ is applied to $\mathbf{f}_{10}$ to produce the final output of the S-box, $\mathbf{s}$ as in (4.20).

In the modified fast S-box architecture in Figure 4.6 , the idea is to remove the output transformation matrix, $\mathbf{T}_{out}$ to reduce the critical path delay of the fast S-box by $3D_X$. This is done by removing the output multipliers block and the output transformation block and then derive the equations needed to find the output of the S-box $\mathbf{s}$ directly from the output of the subfield inverter, E. Those equations depend on the specific output transformation matrix used. In [69], they used the following equation to derive the S-box output $\mathbf{s}$, based on the results of the logic minimization algorithm for $\mathbf{T}_{out}$, as

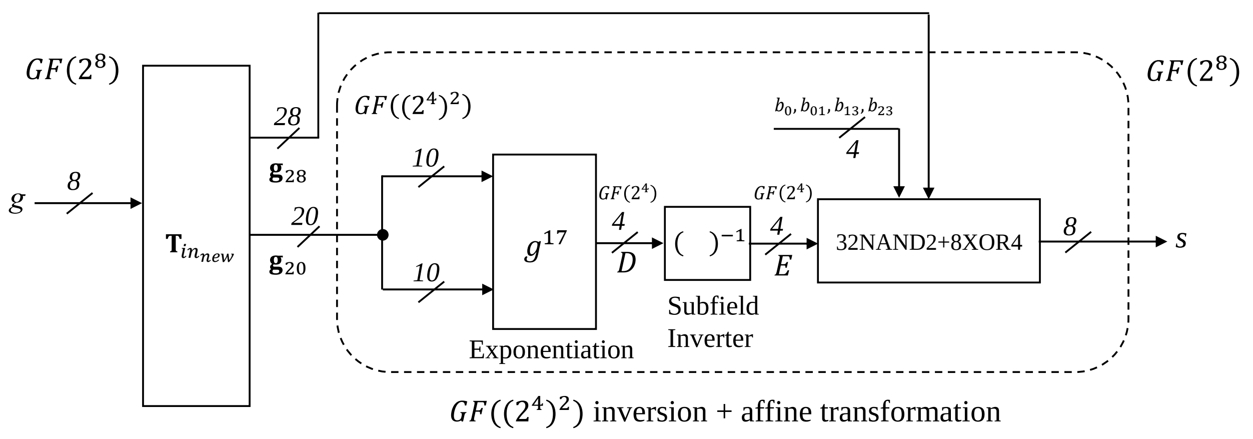


Figure 4.6: The modified fast S-box architectures, Fast 1.

$$\mathbf{s} = \begin{bmatrix} s_7 \\ s_6 \\ s_5 \\ s_4 \\ s_3 \\ s_2 \\ s_1 \\ s_0 \end{bmatrix} = \mathbf{T}_{out} \times \mathbf{f}_{10} \oplus \mathbf{h} = \begin{bmatrix} 0&1&0&0&1&0&0&0&1&1 \\ 1&1&1&0&1&1&1&1&0&1 \\ 1&1&0&1&1&0&1&1&1&1 \\ 1&1&0&0&0&0&1&1&1&1 \\ 0&0&1&1&0&0&0&1&0&1 \\ 0&1&1&1&1&0&0&0&0&0 \\ 0&0&1&0&1&0&0&0&0&0 \\ 0&1&1&0&0&0&1&1&1&1 \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ w_3 \\ w_4 \\ z_0 \\ z_1 \\ z_2 \\ z_3 \\ z_4 \end{bmatrix} \oplus \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}. \tag{4.20}$$

where $W = (w_0 w_1 w_2 w_3 w_4) = \sum_{i=0}^{3} w_i \beta^{2^i} + w_4$ and $Z = (z_0 z_1 z_2 z_3 z_4) = \sum_{i=0}^{3} z_i \beta^{2^i} + z_4$ are the 10-bit signal, $\mathbf{f}_{10}$ at the output of the multipliers block in Figure 4.5. The formulations to generate w_i and $z_i, 0 \leq i \leq 4$ are derived in [69] as

$$\begin{aligned} w_0 &= (e_0 b_0)' \oplus (e_{12} b_{12})' \\ w_1 &= (e_1 b_1)' \oplus (e_{23} b_{23})' \\ w_2 &= (e_2 b_2)' \oplus (e_{30} b_{30})' \\ w_3 &= (e_3 b_3)' \oplus (e_{01} b_{01})' \\ w_4 &= (e_{02} b_{02})' \oplus (e_{13} b_{13})' \end{aligned} \tag{4.21}$$

$$\begin{aligned} z_0 &= (e_0 a_0)' \oplus (e_{12} a_{12})' \\ z_1 &= (e_1 a_1)' \oplus (e_{23} a_{23})' \\ z_2 &= (e_2 a_2)' \oplus (e_{30} a_{30})' \\ z_3 &= (e_3 a_3)' \oplus (e_{01} a_{01})' \\ z_4 &= (e_{02} a_{02})' \oplus (e_{13} a_{13})' \end{aligned} \tag{4.22}$$

To derive architecture in Figure 4.6, we evaluate **s** directly, by substituting (4.21) and (4.22) in (4.20). We can derive the following equation for **s**

$$\begin{aligned} s_7 &= e_0(b_{02} \oplus a_{12}) \oplus e_1(b_3 \oplus a_{03}) \oplus e_2(b_{03} \oplus a_{02}) \oplus e_3(b_{12} \oplus a_1) \\ s_6 &= e_0(b_3 \oplus b_{02} \oplus a_3 \oplus a_{02}) \oplus e_1(b_2 \oplus b_{13} \oplus a_2 \oplus a_{13}) \oplus e_2(b_{13} \oplus b_0 \oplus a_{13} \oplus a_0) \\ &\quad \oplus e_3(b_{02} \oplus b_{13} \oplus a_{02} \oplus a_{13}) \oplus 1 \\ s_5 &= e_0(b_1 \oplus b_{02} \oplus a_{13} \oplus a_{02}) \oplus e_1(b_2 \oplus b_{03} \oplus a_0 \oplus a_{13}) \oplus e_2(b_{13} \oplus b_{02} \oplus a_3 \oplus a_{02}) \\ &\quad \oplus e_3(b_2 \oplus b_{13} \oplus a_{02} \oplus a_1) \oplus 1 \\ s_4 &= e_0(b_0 \oplus a_{13} \oplus a_{02}) \oplus e_1(b_2 \oplus a_0 \oplus a_{13}) \oplus e_2(b_{13} \oplus a_3 \oplus a_{02}) \oplus e_3(b_{23} \oplus a_{02} \oplus a_1) \\ s_3 &= e_0(b_{13} \oplus a_{23}) \oplus e_1(b_{01} \oplus a_{13}) \oplus e_2(b_2 \oplus a_0) \oplus e_3(b_0 \oplus a_{01}) \\ s_2 &= e_0(b_{13} \oplus b_{02}) \oplus e_1(b_0 \oplus b_{13}) \oplus e_2(b_3 \oplus b_{02}) \oplus e_3(b_{02} \oplus b_1) \\ s_1 &= e_0 b_{23} \oplus e_1 b_{13} \oplus e_2 b_0 \oplus e_3 b_{01} \oplus 1 \\ s_0 &= e_0(b_{03} \oplus a_{13} \oplus a_{02}) \oplus e_1(b_1 \oplus a_0 \oplus a_{13}) \oplus e_2(b_3 \oplus a_3 \oplus a_{02}) \\ &\quad \oplus e_3(b_{02} \oplus a_{02} \oplus a_1) \oplus 1 \end{aligned} \tag{4.23}$$

The 8-bit S-box output, **s** in (4.23) requires 32 NAND2 operations between the 4-bit output of the subfield inverter, E and 32 signals. 8 XOR4 gates are then used to sum up the 4 sub-results to derive each $s_i, 0 \leq i \leq 7$. We denote the 28-bit new signal as $\mathbf{g}_{28}$, it depends linearly on $A_{jk} = \{a_{01}, a_{02}, a_{03}, a_{12}, a_{13}, a_{23}\}$, and $B_{jk} = \{b_{01}, b_{02}, b_{03}, b_{12}, b_{13}, b_{23}\}$. In addition to $\mathbf{g}_{28}$, the 32NAND2+8XOR4 will need the signals $b_0, b_{01}, b_{13}, b_{23}$ to formulate s_1 in (4.23). Each bit of $\mathbf{g}_{28}$ is a linear combination of some terms of A_{jk} and B_{jk}. Those 28 linear combinations in (4.23) are added to the input linear mapping of the S-box to formulate a new 48×8 input transformation matrix, $\mathbf{T}_{in_{new}}$. Hence, we include 28 extra signals in the logic-minimization problem and use the focused-search logic minimization algorithm as proposed in [69] to minimize $\mathbf{T}_{in_{new}}$. By using the input transformation matrix in (4.19), we can write the new input transformation matrix as

$$\mathbf{T}_{in_{new}} = \begin{bmatrix} \mathbf{T}_{in} \\ \mathbf{T}_{in_{ext}} \end{bmatrix} \tag{4.24}$$

where $\mathbf{T}_{in}$ is given in (4.19) and $\mathbf{T}_{in_{ext}}$ is given as

$$\mathbf{g}_{28} = \mathbf{T}_{in_{ext}}\mathbf{g} = \left[\begin{array}{c} o_{20} = b_{02} \oplus a_{12} \\ o_{21} = b_3 \oplus a_{03} \\ o_{22} = b_{03} \oplus a_{02} \\ o_{23} = b_{12} \oplus a_1 \\ o_{24} = b_3 \oplus b_{02} \oplus a_3 \oplus a_{02} \\ o_{25} = b_2 \oplus b_{13} \oplus a_2 \oplus a_{13} \\ o_{26} = b_{13} \oplus b_0 \oplus a_{13} \oplus a_0 \\ o_{27} = b_{02} \oplus b_{13} \oplus a_{02} \oplus a_{13} \\ o_{28} = b_1 \oplus b_{02} \oplus a_{13} \oplus a_{02} \\ o_{29} = b_2 \oplus b_{03} \oplus a_0 \oplus a_{13} \\ o_{30} = b_{13} \oplus b_{02} \oplus a_3 \oplus a_{02} \\ o_{31} = b_2 \oplus b_{13} \oplus a_{02} \oplus a_1 \\ o_{32} = b_0 \oplus a_{13} \oplus a_{02} \\ o_{33} = b_2 \oplus a_0 \oplus a_{13} \\ o_{34} = b_{13} \oplus a_3 \oplus a_{02} \\ o_{35} = b_{23} \oplus a_{02} \oplus a_1 \\ o_{36} = b_{13} \oplus a_{23} \\ o_{37} = b_{01} \oplus a_{13} \\ o_{38} = b_2 \oplus a_0 \\ o_{39} = b_0 \oplus a_{01} \\ o_{40} = b_{13} \oplus b_{02} \\ o_{41} = b_0 \oplus b_{13} \\ o_{42} = b_3 \oplus b_{02} \\ o_{43} = b_{02} \oplus b_1 \\ o_{44} = b_{03} \oplus a_{13} \oplus a_{02} \\ o_{45} = b_1 \oplus a_0 \oplus a_{13} \\ o_{46} = b_3 \oplus a_3 \oplus a_{02} \\ o_{47} = b_{02} \oplus a_{02} \oplus a_1 \end{array}\right] \begin{bmatrix} g_7 \\ g_6 \\ g_5 \\ g_4 \\ g_3 \\ g_2 \\ g_1 \\ g_0 \end{bmatrix} \tag{4.25}$$

The theoretical minimum critical path delay of $\mathbf{T}_{in_{new}}$ is $3D_X$, as the maximum number of non-zero entries in $\mathbf{T}_{in_{new}}$ is 8. The maximum allowed delay D_{max} is set to $3D_X$, then the focused-search logic minimization algorithm is applied. The corresponding formulations for $\mathbf{T}_{in_{new}}$ are presented in Table 4.6. For the input transformation block $\mathbf{T}_{in_{new}}$, the presented implementation in Table 4.6 requires 51 XOR2 gates with a propagation delay of 3 XOR2 gates (as required).

Table 4.6: Fast 1 implementation of transformations.

$a_0 = a_1 \oplus g_3$	$(2D_X)$	$a_1 = g_2 \oplus g_0$	$(1D_X)$	$a_2 = a_3 \oplus a_{23}$	$(3D_X)$
$a_3 = g_5 \oplus g_0$	$(1D_X)$	$b_0 = b_{13} \oplus o_{41}$	$(3D_X)$	$b_1 = t_4 \oplus t_3$	$(3D_X)$
$b_2 = a_0 \oplus g_6$	$(3D_X)$	$b_3 = a_{03} \oplus o_{21}$	$(3D_X)$	$a_{01} = g_3$	$(0D_X)$
$a_{02} = a_{03} \oplus a_{23}$	$(3D_X)$	$a_{03} = a_{13} \oplus g_3$	$(2D_X)$	$a_{12} = a_{13} \oplus a_{23}$	$(3D_X)$
$a_{13} = g_5 \oplus g_2$	$(1D_X)$	$a_{23} = t_6 \oplus g_1$	$(2D_X)$	$b_{01} = t_6 \oplus t_5$	$(3D_X)$
$b_{02} = g_5 \oplus g_4$	$(1D_X)$	$b_{03} = g_7$	$(0D_X)$	$b_{12} = o_{27} \oplus o_{44}$	$(3D_X)$
$b_{13} = t_0 \oplus t_2$	$(2D_X)$	$b_{23} = b_{02} \oplus g_7$	$(2D_X)$	$o_{20} = o_{46} \oplus o_{29}$	$(3D_X)$
$o_{21} = t_6 \oplus t_1$	$(2D_X)$	$o_{22} = a_{13} \oplus o_{44}$	$(3D_X)$	$o_{23} = o_{44} \oplus o_{30}$	$(3D_X)$
$o_{24} = g_1$	$(0D_X)$	$o_{25} = t_6 \oplus g_4$	$(2D_X)$	$o_{26} = t_0 \oplus g_2$	$(2D_X)$
$o_{27} = g_7 \oplus g_5$	$(1D_X)$	$o_{28} = t_2 \oplus a_0$	$(3D_X)$	$o_{29} = a_{13} \oplus t6$	$(2D_X)$
$o_{30} = a_1 \oplus o_{27}$	$(2D_X)$	$o_{31} = a_{03} \oplus o_{25}$	$(3D_X)$	$o_{32} = o_{46} \oplus t3$	$(3D_X)$
$o_{33} = a_{13} \oplus g_6$	$(2D_X)$	$o_{34} = t_3 \oplus g_4$	$(3D_X)$	$o_{35} = t_1 \oplus o_{44}$	$(3D_X)$
$o_{36} = a_{23} \oplus b_{13}$	$(3D_X)$	$o_{37} = t_5 \oplus o_{29}$	$(3D_X)$	$o_{38} = g_6$	$(0D_X)$
$o_{39} = t_1 \oplus o_{33}$	$(3D_X)$	$o_{40} = o_{26} \oplus o_{33}$	$(3D_X)$	$o_{41} = a_1 \oplus t_4$	$(2D_X)$
$o_{42} = t_6 \oplus a_0$	$(3D_X)$	$o_{43} = b_{23} \oplus o_{41}$	$(3D_X)$	$o_{44} = t_0 \oplus g_6$	$(2D_X)$
$o_{45} = o_{26} \oplus g_7$	$(3D_X)$	$o_{46} = b_{02} \oplus g_1$	$(2D_X)$	$o_{47} = t_0 \oplus o_{21}$	$(3D_X)$
$t_0 = g_3 \oplus g_1$	$(1D_X)$	$t_1 = g_4 \oplus g_0$	$(1D_X)$	$t_2 = g_6 \oplus g_4$	$(1D_X)$
$t_3 = a_1 \oplus g_7$	$(2D_X)$	$t_4 = g_5 \oplus g_1$	$(1D_X)$	$t_5 = t_0 \oplus g_4$	$(2D_X)$
$t_6 = g_7 \oplus g_6$	$(1D_X)$				

As shown in Figure 4.6, the 28 new signals $\boldsymbol{g}_{28}$, in addition to the 4 signals $b_0, b_{01}, b_{13}, b_{23}$ are applied to the 32NAND2+8XOR4 block. This block consists of 32 NAND2, 24 XOR2/XNOR2 gates. This block is used to compute the S-box output $s = (s_7, \cdots, s_1, s_0) \in GF(2^8)$ in (4.23). This is done by performing the 32 AND operations in (4.23) using 32 NAND2 gates, and then XOR the corresponding four terms in each $s_j, 0 \leq j \leq 7$ using 8 XOR4 (an XOR4 is a two-level binary tree of 3 XOR2 gates). Note that inverting both inputs of the XOR2 gate does not change its operation. As a result, the use of 32 NAND2 instead of 32 AND2 to implement the 32 AND terms in (4.23) does not change the S-box output **s**. In the second level of the binary tree of XOR2 gates used to implement the 8 XOR4 part of the block, four XOR2 gates have been replaced with XNOR2 gates in order to incorporate the effect of adding the constant $\mathbf{h} = [01100011]^{tr}$.

4.4.2 The Modified Fast S-box, Fast 2

In the following, we will derive the architecture of the second modified fast S-box, Fast 2 shown in Figure 4.7. It has a better performance than Fast 1 dervied in section 4.4.1. Fast 2 architecture corresponds to the case $\nu = 1000$, Generator #2 and the binary field element $\eta = 3$.

We noticed from the Matlab® simulations in Figure A.2 that the focused-search logic minimization algorithm gives a very interesting result. As opposed to the fast implementation proposed in [69], where the fast input transformation matrix at $\eta = 1$ using $\nu = 1000$ and Generator #4 requires 24 XOR2 gates with a maximum delay of $3D_X$, we can notice that the point at $\eta = 3$ for the case $\nu = 1000$ and Generator #2 requires only 20 XOR2 gates with a maximum delay of $3D_X$. This is an interesting result as it will save 4 XOR2 gates from the implementation area of $\mathbf{T}_{in}$ while keeping the maximum delay at $3D_X$ as required by the fast S-box. The binary transformation matrix ($\mathbf{X}^{-1}$) corresponding to this point, where $\nu = 1000$, Generator #2 and $\eta = 3$ is derived from Matlab® as

$$\mathbf{X}^{-1} = \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \end{bmatrix}$$

Hence, using Matlab®, the inverse of $\mathbf{X}^{-1}$ is given by

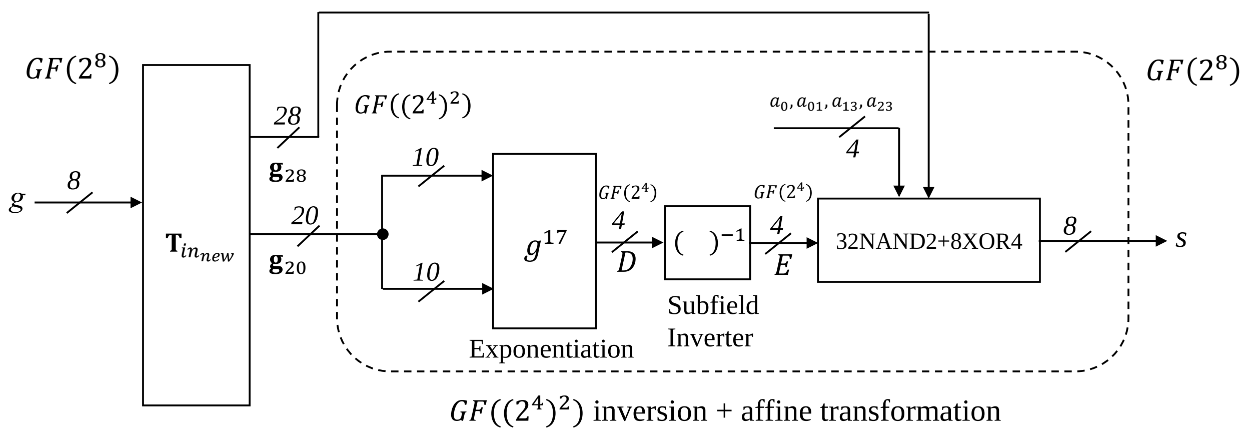


Figure 4.7: The modified fast S-box architectures, Fast 2.

$$\mathbf{X} = \begin{bmatrix} 1 & 0 & 1 & 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 \end{bmatrix}$$

From (4.15) and using Matlab®, the binary matrix used to multiply by the binary field element $\eta = 3$ is

$$\mathbf{C} = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

As a result, $\mathbf{X}_{new}^{-1}$ can be written as

$$\mathbf{X}_{\mathbf{new}}^{\mathbf{-1}} = \mathbf{X}^{-1}\mathbf{C} = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \end{bmatrix}$$

At the output of the S-box, $\mathbf{T}_{out} = \mathbf{MCX} \times \mathbf{T}1$, where

$$\mathbf{M} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Therefore

$$\mathbf{s} = \begin{bmatrix} s_7 \\ s_6 \\ s_5 \\ s_4 \\ s_3 \\ s_2 \\ s_1 \\ s_0 \end{bmatrix} = \mathbf{T}_{out} \times \mathbf{f}_{10} \oplus \mathbf{h} = \begin{bmatrix} 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ w_3 \\ w_4 \\ z_0 \\ z_1 \\ z_2 \\ z_3 \\ z_4 \end{bmatrix} \oplus \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}. \qquad (4.26)$$

Therefore, the new extended-input ($\mathbf{T}_{in_{new}}$) transformation matrix can be found as follows:

$$\mathbf{T}_{in_{new}} = \begin{bmatrix} \mathbf{T}_{in} \\ \mathbf{T}_{in_{ext}} \end{bmatrix} \tag{4.27}$$

where

$$\mathbf{g}_{20} = \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ b_0 \\ b_1 \\ b_2 \\ b_3 \\ a_{01} \\ a_{02} \\ a_{03} \\ a_{12} \\ a_{13} \\ a_{23} \\ b_{01} \\ b_{02} \\ b_{03} \\ b_{12} \\ b_{13} \\ b_{23} \end{bmatrix} = \mathbf{T}_{in}\mathbf{g} = \begin{bmatrix} 0&0&0&1&1&0&0&0 \\ 0&1&0&0&0&0&0&1 \\ 1&0&1&1&1&0&0&1 \\ 0&0&0&0&0&1&0&0 \\ 1&1&1&0&0&0&0&1 \\ 1&1&1&1&0&0&1&1 \\ 0&1&1&1&0&0&0&1 \\ 0&1&1&1&0&1&1&1 \\ 0&1&0&1&1&0&0&1 \\ 1&0&1&0&0&0&0&1 \\ 0&0&0&1&1&1&0&0 \\ 1&1&1&1&1&0&0&0 \\ 0&1&0&0&0&1&0&1 \\ 1&0&1&1&1&1&0&1 \\ 0&0&0&1&0&0&1&0 \\ 1&0&0&1&0&0&0&0 \\ 1&0&0&1&0&1&1&0 \\ 1&0&0&0&0&0&1&0 \\ 1&0&0&0&0&1&0&0 \\ 0&0&0&0&0&1&1&0 \end{bmatrix} \begin{bmatrix} g_7 \\ g_6 \\ g_5 \\ g_4 \\ g_3 \\ g_2 \\ g_1 \\ g_0 \end{bmatrix}, \tag{4.28}$$

and

$$\mathbf{g}_{28} = \mathbf{T}_{in_{ext}}\mathbf{g} = \begin{bmatrix} o_{20} = b_3 \oplus b_{12} \oplus a_3 \oplus a_{02} \\ o_{21} = b_{02} \oplus b_{13} \oplus a_2 \oplus a_{13} \\ o_{22} = b_1 \oplus b_{02} \oplus a_{13} \oplus a_0 \\ o_{23} = b_0 \oplus b_{13} \oplus a_{02} \oplus a_{13} \\ o_{24} = b_{12} \oplus a_3 \oplus a_{12} \\ o_{25} = b_{03} \oplus a_{02} \oplus a_{13} \\ o_{26} = b_{02} \oplus a_1 \oplus a_{02} \\ o_{27} = b_1 \oplus a_0 \oplus a_{13} \\ o_{28} = b_1 \oplus a_{01} \\ o_{29} = b_{02} \oplus a_0 \\ o_{30} = b_{12} \oplus a_{23} \\ o_{31} = b_3 \oplus a_2 \\ o_{32} = b_{03} \oplus a_3 \oplus a_{01} \\ o_{33} = b_1 \oplus a_2 \oplus a_{01} \\ o_{34} = b_3 \oplus a_{13} \oplus a_2 \\ o_{35} = b_{02} \oplus a_{02} \oplus a_3 \\ o_{36} = b_3 \oplus b_{01} \oplus a_0 \\ o_{37} = b_2 \oplus b_{01} \oplus a_2 \\ o_{38} = b_{13} \oplus b_2 \oplus a_{13} \\ o_{39} = b_{02} \oplus b_3 \oplus a_{23} \\ o_{40} = b_0 \oplus a_{03} \\ o_{41} = b_2 \oplus a_1 \\ o_{42} = b_{13} \oplus a_3 \\ o_{43} = b_{23} \oplus a_{02} \\ o_{44} = b_1 \oplus b_{02} \oplus a_3 \\ o_{45} = b_2 \oplus b_{03} \oplus a_{12} \\ o_{46} = b_{13} \oplus b_{02} \oplus a_1 \\ o_{47} = b_2 \oplus b_{13} \oplus a_{03} \end{bmatrix} \tag{4.29}$$

The focused-search logic minimization algorithm [69] is used, with no maximum delay constraint set, to minimize $\mathbf{T}_{in_{new}}$ in (4.27). The corresponding formulations for $\mathbf{T}_{in_{new}}$ are presented in Table 4.7. For the input transformation block $\mathbf{T}_{in_{new}}$, the presented implementation in Table 4.7 requires 46 XOR2 gates with a maximum propagation delay of 6 XOR2 gates. We can notice from table 4.7 that the maximum delay of all 20 bits of $\mathbf{g}_{20}$ in (4.28) is still $3D_X$, as a result, the critical path delay of Fast 2 will not be increased by applying no maximum delay constraint on $\mathbf{T}_{in_{new}}$, while saving some area from the design. The maximum delay of $\mathbf{g}_{28}$ signal is $6D_X$. The $\mathbf{g}_{28}$ signals will be implemented in parallel to the exponentiation and subfield inverter block, thus not affecting the S-box delay.

If we apply a maximum delay constraint of $3D_X$ to $\mathbf{T}_{in_{new}}$, the logic minimization algorithm would give a hardware complexity of 51 XOR2, while the maximum delay for all $\mathbf{T}_{in_{new}}$ signals, including $\mathbf{g}_{20}$ in (4.28) and $\mathbf{g}_{28}$ in (4.29), will be $3D_X$. This result will not lead to any improvement of the CPD of the S-box, as $\mathbf{g}_{28}$ is not on the critical path of the S-box, and the S-box will have the same delay whether we use the 46 XOR2 implementation or the 51 XOR2 one. As a result, it is better to use the implementation in table 4.7 to save 5 XOR2 gates, while keeping the same delay for the S-box.

As shown in Figure 4.7, the 28 new signals $\boldsymbol{g}_{28}$, in addition to the 4 signals $a_0, a_{01}, a_{13}, a_{23}$ are applied to 32NAND2+8XOR4 block. Four XOR2 gates have been replaced with XNOR2 gates in order to incorporate the effect of adding the constant $\mathbf{h} = [01100011]^{tr}$ in (4.26).

Table 4.7: Fast 2 implementation of transformations.

$a_0 = g_4 \oplus g_3$	$(1D_X)$	$a_1 = g_6 \oplus g_0$	$(1D_X)$	$a_2 = a_0 \oplus a_{02}$	$(3D_X)$
$a_3 = g_2$	$(0D_X)$	$b_0 = a_{02} \oplus g_6$	$(3D_X)$	$b_1 = b_{12} \oplus b_2$	$(3D_X)$
$b_2 = a_1 \oplus o_{41}$	$(2D_X)$	$b_3 = b_{23} \oplus b_2$	$(3D_X)$	$a_{01} = a_0 \oplus a_1$	$(2D_X)$
$a_{02} = o_{23} \oplus g_5$	$(2D_X)$	$a_{03} = a_0 \oplus g_2$	$(2D_X)$	$a_{12} = o_{29} \oplus o_{26}$	$(3D_X)$
$a_{13} = a_1 \oplus g_2$	$(2D_X)$	$a_{23} = a_{02} \oplus a_{03}$	$(3D_X)$	$b_{01} = g_4 \oplus g_1$	$(1D_X)$
$b_{02} = g_7 \oplus g_4$	$(1D_X)$	$b_{03} = b_{01} \oplus b_{13}$	$(2D_X)$	$b_{12} = g_7 \oplus g_1$	$(1D_X)$
$b_{13} = g_7 \oplus g_2$	$(1D_X)$	$b_{23} = g_2 \oplus g_1$	$(1D_X)$	$o_{20} = g_6 \oplus g_4$	$(1D_X)$
$o_{21} = a_{12} \oplus g_4$	$(4D_X)$	$o_{22} = o_{30} \oplus g_0$	$(5D_X)$	$o_{23} = g_7 \oplus g_0$	$(1D_X)$
$o_{24} = b_{03} \oplus o_{21}$	$(5D_X)$	$o_{25} = o_{26} \oplus g_1$	$(3D_X)$	$o_{26} = o_{20} \oplus g_5$	$(2D_X)$
$o_{27} = o_{33} \oplus a_{23}$	$(4D_X)$	$o_{28} = o_{33} \oplus a_2$	$(4D_X)$	$o_{29} = g_7 \oplus g_3$	$(1D_X)$
$o_{30} = b_{12} \oplus a_{23}$	$(4D_X)$	$o_{31} = a_2 \oplus b_3$	$(4D_X)$	$o_{32} = a_2 \oplus o_{25}$	$(4D_X)$
$o_{33} = b_{01} \oplus g_0$	$(2D_X)$	$o_{34} = o_{32} \oplus g_6$	$(5D_X)$	$o_{35} = a_{13} \oplus o_{26}$	$(3D_X)$
$o_{36} = o_{42} \oplus o_{40}$	$(5D_X)$	$o_{37} = o_{39} \oplus g_7$	$(6D_X)$	$o_{38} = b_{02} \oplus g_5$	$(2D_X)$
$o_{39} = o_{43} \oplus o_{40}$	$(5D_X)$	$o_{40} = a_{23} \oplus g_6$	$(4D_X)$	$o_{41} = g_5 \oplus g_4$	$(1D_X)$
$o_{42} = g_7$	$(0D_X)$	$o_{43} = b_{23} \oplus a_{02}$	$(3D_X)$	$o_{44} = b_3 \oplus g_4$	$(4D_X)$
$o_{45} = o_{30} \oplus g_5$	$(5D_X)$	$o_{46} = a_{13} \oplus g_4$	$(3D_X)$	$o_{47} = b_0 \oplus g_3$	$(4D_X)$

4.4.3 Complexity Analysis

The space and time complexities of the modified fast architectures Fast 1 and Fast 2 derived in section 4.4.1 and 4.4.2, respectively along with all the blocks used in Figure 4.6 and Figure 4.7 are summarized in Table 4.8. The corresponding GEs of all the blocks are also presented. GE is the chip areas in terms of an equivalent of 2-input NAND gates. The provided GEs are based on the used 65nm CMOS technology.

4.4.4 Comparisons of the Space and Time Complexities

In this section, we compare the space and time complexities of the modified fast S-box architectures Fast 1 and Fast 2 derived in section 4.4.1 and section 4.4.2, respectively against the fastest S-boxes available in the literature.

Tables 4.9 and 4.10 provide the gate count and time delay of two S-boxes and compare them with the proposed architectures. We chose two of the fastest schemes, namely Reyhani et al. fast

Table 4.8: Space and time complexities of the modified S-box architecture Fast 1 and Fast 2.

Block/ Target	Space Complexity*						Time Complexity
	X	ND	N3	NR	NT	GE	
Modified Input Transformation Block							
$\mathbf{T}_{in_{new1}}$ (4.24)	51					102	$3D_X$
$\mathbf{T}_{in_{new2}}$ (4.27)	46					92	$6D_X$**
$GF((2^4)^2)$ Inversion and Affine Transformation							
Exponentiation	14	7		3		38	$3D_X + 1D_{ND}$
Subfield Inverter	4	12	4		4	28	$3D_{ND} + 1D_{NT}$
32NAND2+8XOR4	24	32				80	$2D_X + 1D_{ND}$
Total	42	51	4	3	4	146	$5D_X + 5D_{ND} + 1D_{NT}$
Total Complexity of Modified Fast S-Box (Figure 4.6)							
Fast 1	93	51	4	3	4	248	$8D_X + 5D_{ND} + 1D_{NT}$
Fast 2	88	51	4	3	4	238	$8D_X + 5D_{ND} + 1D_{NT}$

*All GEs values are estimated using STM 65nm technology where X is XOR2/XNOR2 = 2GEs, ND is NAND2 = 1GE, N3 is NAND3 = 1.25GEs, NR is NOR2 = 1GE, NT is NOT = 0.75GEs.

**The maximum delay for all $\mathbf{g}_{20}$ signals is $3D_X$.

Table 4.9: Space complexity comparison of different fast S-boxes.

S-boxes	Gate count							GE
	X	AD	ND	N3	NR	NT	MX	
Reyhani Fast [69]	79	-	39	4	3	4	-	208
Maximov Fast [54]	78	4	37	-	5	-	6	215
Fast 1 (Proposed)	93	-	51	4	3	4	-	248
Fast 2 (Proposed)	88	-	51	4	3	4	-	238

*All GEs values are estimated using STM 65nm technology where X is XOR2/XNOR2 = 2GEs, AD is AND2 = 1.25GEs, ND is NAND2 = 1GE, N3 is NAND3 = 1.25GEs, NR is NOR2 = 1GE, NT is NOT = 0.75GEs, MX is 2-to-1 noninverting MUX = 2GEs.

design [69] and Maximov et al. fast S-box [54]. The space complexity comparison of fast designs helps in detecting where the speed improvements came from and also in validating the CAD tool results.

We provides the GEs for all S-box architectures using the corresponding GEs of the gates from the CMOS 65nm technology. The critical path delay (CPD) was obtained by the CAD tool after coding the four S-boxes listed in VHDL.

4.4.5 Calculating Delay in CMOS Circuits

Finding the critical path delay of the S-box is crucial in determining its speed. In 2019, Maximov et al. [54] introduced the concept of “Tech. XOR depth” in an attempt to standardize the calculation of the critical path delay of the S-box among different technology libraries. We will show that the “Tech. XOR depth” concept can not be used as a standard method to measure the delay of the S-box.

In Table 6 of [54], they proposed to normalize the delay of any basic gate by the delay of an XOR2 gate. They used the standard cell library Samsung’s STD90/MDL90 0.35 μm [47] to get

Table 4.10: Time complexity comparison for different fast S-boxes.

S-Boxes	CPD*
Reyhani Fast [69]	$11D_X + 5D_{ND} + 1D_{NT}$
Maximov [54]	$8D_X + 2D_{NR} + 1D_{AD} + 1D_{MX}$
Fast 1 (Proposed)	$8D_X + 5D_{ND} + 1D_{NT}$
Fast 2 (Proposed)	$8D_X + 5D_{ND} + 1D_{NT}$

*X = XOR2/XNOR2, AD = AND2, ND = NAND2, NR = NOR2, NT = NOT, MX = 2-to-1 noninverting MUX.

Table 4.11: The input capacitance, C_{in} of some basic gates in Samsung's STD90/MDL90 0.35 μm library normalized to SL [47].

Std. Cell*	XR	XN	AD	ND	OR	NR	INV	MX	MXI
C_{in} at input A	1	1	0.8	1.1	0.7	1	1	0.8	0.6
C_{in} at input B	1.5	2	0.8	1.1	0.7	1		0.8	0.6

*XR = XOR2, XN = XNOR2, AD = AND2, ND = NAND2, OR = OR2, NR = NOR2, INV = NOT, MX = 2-to-1 noninverting MUX, MXI = 2-to-1 inverting MUX.

the maximum delay of each basic gate, at the same value of the load capacitance at the gate's output. They used a normalized value of $SL = 2$ at the output of each gate along the critical path of the S-box, where SL refers to standard load and for the specific technology library used [47], $1\,SL = 0.01352$ pF. The delay model in the Samsung's STD90/MDL90 0.35 μm library uses an effective capacitance C_{EFF} as an index to the delay table of each gate. The effective capacitance C_{EFF} approximates the distributed interconnection wire resistance and capacitance at the output of the gate. In [54], they used the same value for this index capacitance ($SL = 2$) for all the gates on the critical path to calculate the CPD. Afterwards, they used the maximum delay of the XOR2 gate calculated using the same load value of $SL = 2$ to normalize the path delay with respect to the delay of the XOR2 gate. They refer to the dealy of each gate normalized to the delay of the XOR gate as a number of "Tech. XOR depth". This is not accurate as the output capacitance of each gate on the critical path is different from $SL = 2$. Table 4.11 shows the input capacitance of some basic gates in Samsung's STD90/MDL90 0.35 μm library. If we neglect the paths that are branching from the critical path as [54] and only model the critical path, then the load capacitance of one gate on the critical path will depend on the input capacitance of the following gate on the critical path. For example, if the critical path of the S-box has 1 XOR2 gate followed by a NAND2 gate, then the capacitive load of the XOR2 gate will be the same as the input capacitance of the NAND2 gate (i.e. $SL = 1.1$). In case that the gate has different input capacitances for its inputs, e.g. XOR2 and XNOR2, we should determine which gate's input is on the critical path and use the corresponding input capacitance value from Table 4.11 as a load for the previous gate. After determining the load capacitance of each gate along the critical path, each load capacitance should be used as an index to the delay table of the corresponding gate to calculate the gate's delay. The critical path delay will be the sum of the individual gate delays on the path.

We conclude that the "Tech XOR depth" concept suggested in [54] is not accurate for calculating the path delay. To get a better insight into the path delay, we could use the method of logical effort [74]. We will only model the critical path and neglect branching as [54]. The logical effort provides a simple method to estimate the minimum possible delay of a circuit based on the linear delay model. In this method, the normalized delay d of a gate is expressed as the sum of the stage effort f and the parasitic delay p . The gate delay d is normalized in units of τ, where τ is the delay of an ideal fanout-of-1 inverter with no parasitic capacitance [74]. By normalizing the gate delay to τ, the circuits can be compared based on the topology rather than the speed of standard cell library used in the synthesis process. The inverter delay can be written as $\tau = 3RC$, where R is the effective resistance of an nMOS transistor and C is the transistor output capacitance. The normalized gate delay d can be expressed relative to this inverter delay as $d = f + p$, where p is the parasitic delay inherent to the gate with no load attached and f is the effort delay. The inverter has three units of diffusion capacitance, $3C$ on its output, so the parasitic delay is $\tau = 3RC$. If we normalize the inverter parasitic delay p_{inv} to τ, we will have $p_{inv} = 1$. Table 4.12 shows the parasitic delays of some common gates relative to the inverter parasitic delay p_{inv}.

The effort delay f depends on the complexity and fanout of the gate, $f = gh$, where g is the logical effort of the gate and h is its electrical effort. The logical effort g represents the complexity of the gate relative to an inverter. The inverter has a logical effort of 1. Table 4.13 shows the

Table 4.12: The parasitic delay of some basic gates normalized to p_{inv} [74].

Gate	XOR2	XNOR2	NAND2	NOR2	NOT	MUX21
Parasitic Delay p	4	4	2	2	1	4

Table 4.13: The logical effort of some basic gates [74].

Gate	XOR2	XNOR2	NAND2	NOR2	NOT	MUX21
Logical effort g	4	4	$\frac{4}{3}$	$\frac{5}{3}$	1	2

logical effort of some basic gates used in standard cell libraries. For example, the 2-input NAND gate has a logical effort of $\frac{4}{3}$,which means that the NAND2 gate will take $\frac{4}{3}$ more time to drive a given load than an inverter. The electrical effort $h = \frac{C_{out}}{C_{in}}$, where C_{out} is the capacitance at the output of the gate and C_{in} is the input capacitance of the gate. In case that the gate is driving identical copies of itself, the term fanout is used to refer to the electrical effort of the gate. In Table 4.13, there is no 2-input AND or 2-input OR gates. The non-inverting gates, like AND and OR are built from multiple stages of inverting gates. For example, to build a 2-input AND gate, we can use a NAND2 followed by an inverter. A 2-input NAND gate that delivers the same current as the inverter is called NAND2-1x.

We can use the logical effort method to estimate the minimum possible delay D of an N-stage path. The path delay D can be written as [83]

$$D = NF^{\frac{1}{N}} + P \tag{4.30}$$

where, F is the path effort, P is the path parasitic delay and N is the number of stages (i.e. gates) on the path under consideration. The path effort F is defined as

$$F = GBH \tag{4.31}$$

where, G is the path logical effort, B is the path branching effort and H is the path electrical effort.

The path logical effort is defined as $G = \prod g_i$, where g_i is the logical efforts of stage (i.e. gate) i along the path.

The path electrical effort $H = \frac{C_L}{C_{in}}$, where C_L is the load capacitance of the path and C_{in} is the input capacitance presented by the path.

To account for branching between stages of a path, the branching effort of stage i, b_i, is defined as the ratio of the total capacitance seen by the stage i to the capacitance on the path. Then the path branching effort $B = \prod b_i$ is the product of the branching efforts between stages.

In a path that does not branch, $B = 1$, then the path effort F is defined as

$$F = GH \tag{4.32}$$

To find the normalized critical path delay of the S-box using the logical effort method, we will only model the critical path rather than the entire circuit (i.e. $B = 1$, no branching from the critical path). This allows us to simplify the comparison between the different S-boxes listed in Table 4.10. We will illustrate the steps for the Fast 2 design.

The path logical effort $G = 4^8 \times (\frac{4}{3})^5 \times 1$

The path electrical effort $H = \frac{C_L}{C_{in}}$, where C_L is the capacitance at the output of the critical path and C_{in} is the capacitance at the input of the critical path.

The path parasitic delay $P = (8 \times 4) + (5 \times 2) + 1$

The number of stages $N = 14$

Then from (4.30), the normalized critical path delay of Fast 2 S-box at a unity path electrical effor is $D_{Fast2} = 77.26$. We summarize the normalized delay of the other S-boxes at a unity path

Table 4.14: Normalizd delay values for different fast S-boxes using the method of logical effort.

S-Boxes	D at $\frac{C_L}{C_{in}} = 1$
Reyhani Fast [69]	100.37
Maximov Fast [54]	78.73
Fast 1 (Proposed)	77.26
Fast 2 (Proposed)	77.26

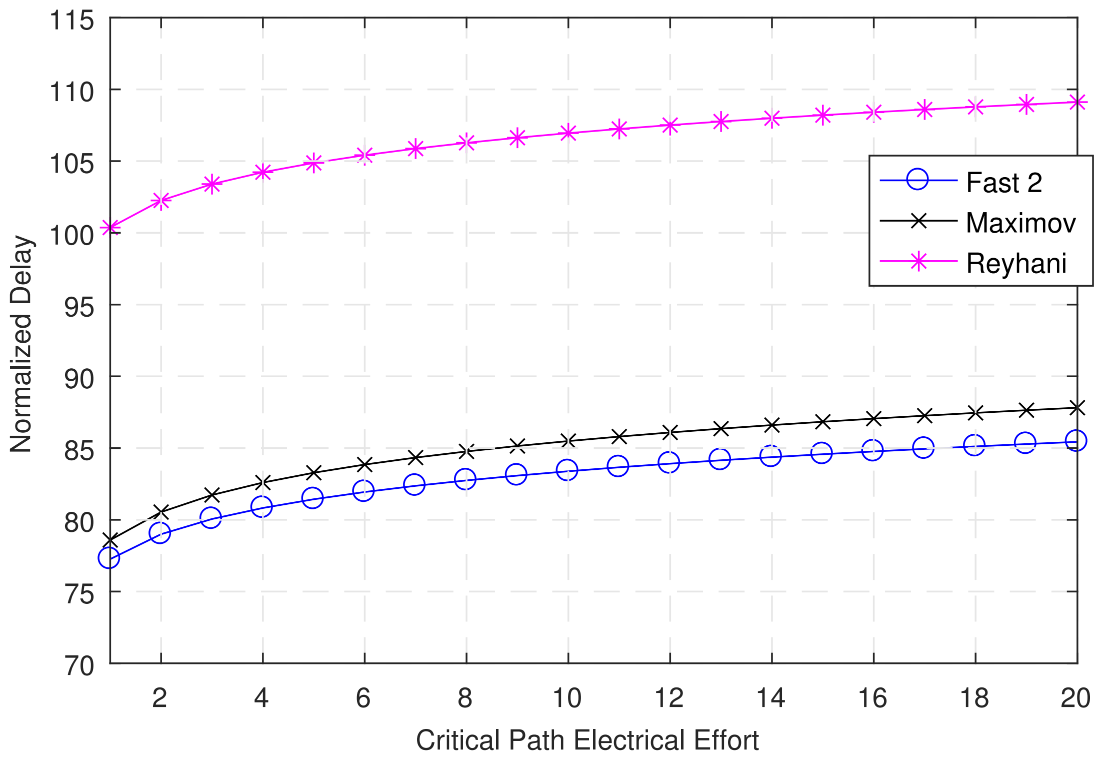


Figure 4.8: Influence of the load on the normalized delay values of the different fast S-boxes.

electrical effort in Table 4.14.

We can plot the the minimum delay values in Table 4.14 as a function of the electrical effort of the critical path, $\frac{C_L}{C_{in}}$ as shown in Figure 4.8. The Fast 2 design has a lower normalized delay for all the values of electrical effort considered.

4.4.6 ASIC Synthesis Results and Comparison

CAD tools are very fast and accurate at evaluating complex delay models. We use Synopsys Design Compiler® in order to give an estimate of the critical path delay, as well as the area and power estimates of the different S-boxes considered. The overall implementation results are presented in Table 4.15. We propose two fast structures: Fast 1 and Fast 2. Logic synthesis was done using VHDL as a design entry to the Synopsys Design Vision®. The technology library used was STM 65nm CMOS standard library and the CORE65LPSVT standard cell library which is optimized for low power applications. The area, delay and power for all the considered S-boxes are generated by the CAD tool with relaxed constraints at a clock frequency of 100 MHz. We coded the above-mentioned two fast S-boxes in VHDL and present their ASIC results in Table 4.15. For each and every code, we verified the codes using the S-box testbenches and Modelsim®. The delay value results from the CAD tool confirm that our normalized delay analysis in Figure 4.8 are correct and that Fast 2 design is the fastest S-box among the four S-boxes considered.

From Table 4.15, the new fast S-box, Fast 2, is the currently fastest S-box in the literature, to our knowledge. The new S-box is about 24% faster than the fast S-box design in [69] with 14% increase in area. It is 9% faster than the fast S-box design in [54] with 11% more area. While the depth of the Fast 2 design is 14 gates and the depth of [54] is 12 gates, Fast 2 design is faster than

Table 4.15: ASIC comparisons of the fast S-Box architectures.

S-Box	Area		Delay	Power	Area-Time
	μm^2	GE	ns	μW	product
Reyhani Fast [69]	432.64	208	0.779697	42.750	162.177
Maximov [54]	447.2	215	0.686869	41.398	**147.677**
Fast 1 (Proposed)	515.84	248	0.636572	49.310	157.870
Fast 2 (Proposed)	**495.04**	**238**	**0.627886**	47.807	149.437

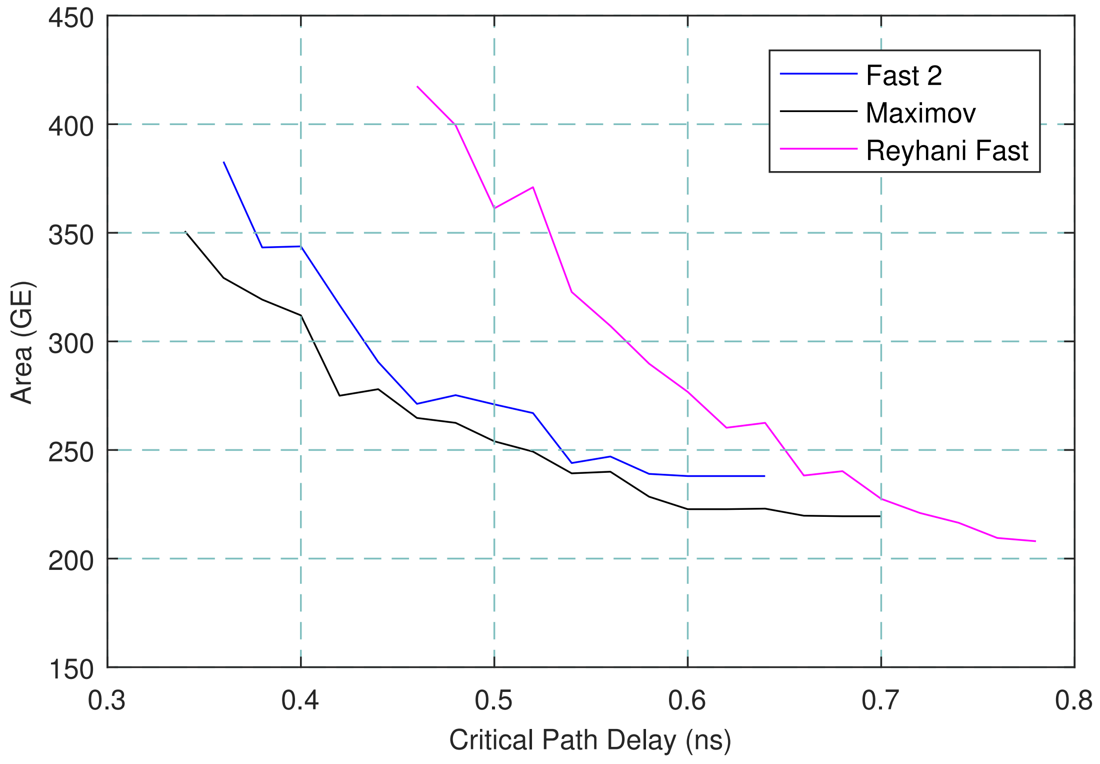


Figure 4.9: Area in GEs of the proposed Fast 2 S-box as compared to previous work [69, 54] at different input delay constraints for the STM 65nm library.

[54] as confirmed by the normalized delay values in Table 4.14 and the synthesis results in Table 4.15. This is due to the better circuit topology used in Fast 2 design and the smaller path effort. The critical path in Fast 2 has less path logical effort than [54], which reduces the path effort in (4.32).

In Figure 4.9, we evaluate the proposed Fast 2 S-box design against previous works under different delay constraints as an input design requirement for the CAD tool. The CAD tool is free to select different output strengths for each gate to reduce the delay of the critical path. Gates with higher output strengths have a smaller delay than their regular counterparts at a slight increase of the synthesized area. We have started at the non constraint delay and area values. Timing constraint are then reduced by 20 ps until the timing constraint could not be met and the slack is violated.

The Fast 2 design has the highest speed among the other two S-boxes considered in Figure 4.9. Fast 2 design has a lower area than Reyhani fast S-box under all timing constraints considered. Maximov fast S–box achives a lower area than Fast 2 and Reyhani fast under tighter timing constraints.

4.5 Additional Extended Transformation Matrices

In this section, we will derive the extended 52×8 input transformation matrix $\mathbf{T}'_{in}$ for the fast S-box architecture shown in Figure 4.10. We will also conduct an exhaustive search over all possible extended 52×8 input transformation matrices in order to find a fast S-box with a minimal area, without sacrificing its speed.

The S-box architecture shown in Figure 4.5 is based on composite field $GF((2^4)^2)$. As pointed out in section 4.3.2, the input to the S-box can be modified by multiplying it by a binary field element $\eta = (1 - 255)$. The matrix form, **C**, of multiplication by η is defined by (4.15). As a result, the 20×8 input transformation matrix $\mathbf{T}_{in}$ can be defined as

$$\mathbf{T}_{in} = \begin{bmatrix} \mathbf{X}^{-1}\mathbf{C} \\ \mathbf{a}_{ij}^{tr} \\ \mathbf{b}_{ij}^{tr} \end{bmatrix} \quad \text{for } 0 \leq i, j \leq 3, i \neq j \tag{4.33}$$

The first and the last 4 rows of $(\mathbf{X}^{-1}\mathbf{C})$ are denoted as $\mathbf{a}_i^{tr}$ and $\mathbf{b}_i^{tr}$, $0 \leq i \leq 3$, respectively. The rows of $(\mathbf{X}^{-1}\mathbf{C})$ generate the coordinates of A and B as $a_i = \mathbf{a}_i^{tr}\mathbf{g}$ and $b_i = \mathbf{b}_i^{tr}\mathbf{g}$, respectively. The next 12 rows of $\mathbf{T}_{in}$ generate $a_{ij} = a_i \oplus a_j = (\mathbf{a}_i^{tr} \oplus \mathbf{a}_j^{tr})\mathbf{g}$ and $b_{ij} = b_i \oplus b_j = (\mathbf{b}_i^{tr} \oplus \mathbf{b}_j^{tr})\mathbf{g}$.

The 20×8 input transformation matrix $\mathbf{T}_{in}$ defined in (4.33) computes the 20 bits of:

$$\mathbf{g}_{20} = [a_0a_1a_2a_3b_0b_1b_2b_3a_{01}a_{02}a_{03}a_{12}a_{13}a_{23}b_{01}b_{02}b_{03}b_{12}b_{13}b_{23}]^{tr}$$

from the S-box input vector $\mathbf{g} = [g_7, \cdots, g_1, g_0]^{tr}$ as follows

$$\mathbf{g}_{20} = \mathbf{T}_{in} \times \mathbf{g} \tag{4.34}$$

The 10-bit signal at the output of the multipliers block are denoted as $W = (w_0w_1w_2w_3w_4) = \sum_{i=0}^{3} w_i\beta^{2^i} + w_4$ and $Z = (z_0z_1z_2z_3z_4) = \sum_{i=0}^{3} z_i\beta^{2^i} + z_4$. The formulations to generate w_i and $z_i, 0 \leq i \leq 4$ are derived in [69] as

$$\begin{aligned} w_0 &= (e_0b_0)' \oplus (e_{12}b_{12})' \\ w_1 &= (e_1b_1)' \oplus (e_{23}b_{23})' \\ w_2 &= (e_2b_2)' \oplus (e_{30}b_{30})' \\ w_3 &= (e_3b_3)' \oplus (e_{01}b_{01})' \\ w_4 &= (e_{02}b_{02})' \oplus (e_{13}b_{13})' \end{aligned} \tag{4.35}$$

$$\begin{aligned} z_0 &= (e_0a_0)' \oplus (e_{12}a_{12})' \\ z_1 &= (e_1a_1)' \oplus (e_{23}a_{23})' \\ z_2 &= (e_2a_2)' \oplus (e_{30}a_{30})' \\ z_3 &= (e_3a_3)' \oplus (e_{01}a_{01})' \\ z_4 &= (e_{02}a_{02})' \oplus (e_{13}a_{13})' \end{aligned} \tag{4.36}$$

At the output, the equations that are required to convert from the RNB back to the NB are included into **MX**, resulting in $\mathbf{T}_{out}$ matrix. The output of the S-box **s** is defined as

$$\mathbf{s} = \mathbf{T}_{out} \times \mathbf{f}_{10} \oplus \mathbf{h} \tag{4.37}$$

where $\mathbf{f}_{10} = [w_0w_1w_2w_3w_4z_0z_1z_2z_3z_4]^{tr}$ is the output of composite field inversion, represented in the RNB and $\mathbf{h} = [01100011]^{tr}$ as presented in (1.2). The output transformation matrix $\mathbf{T}_{out}$ is written as

$$\mathbf{T}_{out} = \mathbf{MCX} \times \mathbf{T1} \tag{4.38}$$

where **T1** is the 8×10 matrix that converts from the RNB representations of $W = \sum_{i=0}^{3} w_i\beta^{2^i} + w_4$ and $Z = \sum_{i=0}^{3} z_i\beta^{2^i} + z_4$ at the outputs of the $GF((2^4)^2)$ inverter back to the non-redundant NB representations. $\mathbf{MCX} = \mathbf{M} \times \mathbf{C} \times \mathbf{X}$ is the 8×10 output transformation matrix, in which **M** is the S-box affine transformation as presented in (1.2), **C** is the matrix form of the binary field element η as given by (4.15), and **X** is the inverse of the input binary matrix, $\mathbf{X}^{-1}$ used for isomorphic mapping from $GF(2^8)$ to $GF((2^4)^2)$.

The S-box architecture in Figure 4.5 can be optimized for speed by removing the output transformation matrix $\mathbf{T}_{out}$ from the critical path. The resulting improved fast architecture is shown in Figure 4.10. In this improved fast S-box architecture, the output **s** of the S-box can be written

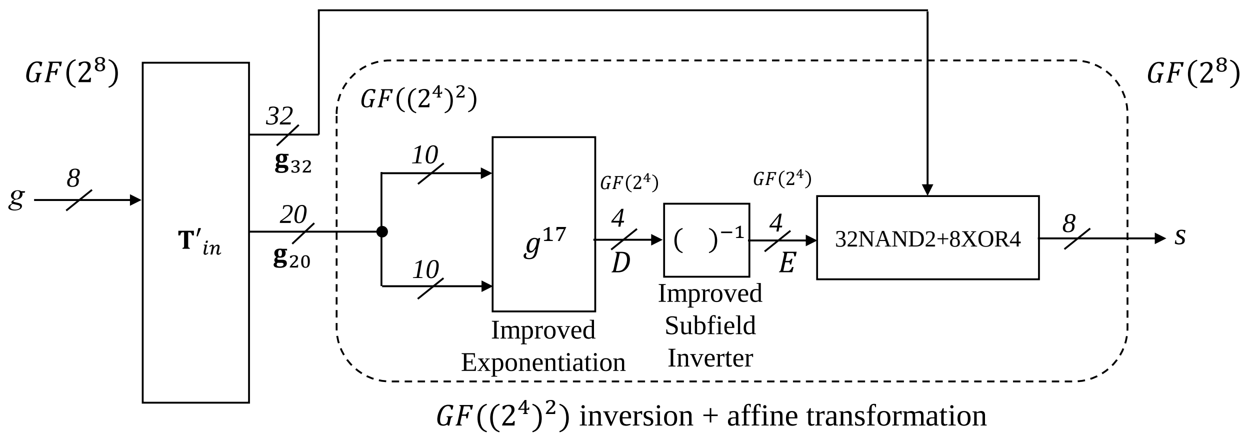


Figure 4.10: Improved fast S-box architecture.

according to (4.37) as

$$\mathbf{s} = \begin{bmatrix} s_7 \\ s_6 \\ s_5 \\ s_4 \\ s_3 \\ s_2 \\ s_1 \\ s_0 \end{bmatrix} = \mathbf{T}_{out} \times \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ w_3 \\ w_4 \\ z_0 \\ z_1 \\ z_2 \\ z_3 \\ z4 \end{bmatrix} \oplus \mathbf{h} \tag{4.39}$$

By substituting $\mathbf{f}_{10} = [w_0w_1w_2w_3w_4z_0z_1z_2z_3z_4]^{tr}$ from (4.35) and (4.36)

$$\mathbf{s} = \begin{bmatrix} s_7 \\ s_6 \\ s_5 \\ s_4 \\ s_3 \\ s_2 \\ s_1 \\ s_0 \end{bmatrix} = \mathbf{T}_{out} \times \begin{bmatrix} (e_0b_0)' \oplus (e_{12}b_{12})' \\ (e_1b_1)' \oplus (e_{23}b_{23})' \\ (e_2b_2)' \oplus (e_{30}b_{30})' \\ (e_3b_3)' \oplus (e_{01}b_{01})' \\ (e_{02}b_{02})' \oplus (e_{13}b_{13})' \\ (e_0a_0)' \oplus (e_{12}a_{12})' \\ (e_1a_1)' \oplus (e_{23}a_{23})' \\ (e_2a_2)' \oplus (e_{30}a_{30})' \\ (e_3a_3)' \oplus (e_{01}a_{01})' \\ (e_{02}a_{02})' \oplus (e_{13}a_{13})' \end{bmatrix} \oplus \mathbf{h} \tag{4.40}$$

The entries of the 8 × 10 output transformation matrix $\mathbf{T}_{out}$ depends on the matrix **C** corresponding to the binary field element $\eta = (1-255)$. It also depends on the chosen 8×8 input matrix **X**. The binary matrix **X** in turn depends on the chosen v and the field generator. As previously mentioned in section 4.3.3, there are only 8 different input binary matrices **X**, corresponding to $v = 1000$, $v = 0111$ and Generator # 1, 2, 3 and 4. For each matrix of the 8 different input matrices **X**, and using Matlab®, we use (4.33) to calculate the input transformation matrix $\mathbf{T}_{in}$ at the different values of $\eta = (1-255)$, a total of 2040 different matrices. We also use (4.38) to compute the corresponding output transformation matrices $\mathbf{T}_{out}$. Each output transformation matrix $\mathbf{T}_{out}$ is then substituted in (4.40) and multiplied by $\mathbf{f}_{10}$. Each row of $\mathbf{T}_{out}$, when multiplied by $\mathbf{f}_{10}$ will result in 4 AND operations between e_0, e_1, e_2 and e_3, respectively and 4 other terms that depend on the $0's$ and $1's$ in that row as shown in Table 4.16. e.g., if a row of $\mathbf{T}_{out}$ has all $1's$, then the 4 AND operations will be the following

$e_0(b_0 \oplus b_{03} \oplus b_{01} \oplus b_{02} \oplus a_0 \oplus a_{03} \oplus a_{01} \oplus a_{02})$
$e_1(b_{12} \oplus b_1 \oplus b_{01} \oplus b_{13} \oplus a_{12} \oplus a_1 \oplus a_{01} \oplus a_{13})$
$e_2(b_{12} \oplus b_{23} \oplus b_2 \oplus b_{02} \oplus a_{12} \oplus a_{23} \oplus a_2 \oplus a_{02})$
$e_3(b_{23} \oplus b_{03} \oplus b_3 \oplus b_{13} \oplus a_{23} \oplus a_{03} \oplus a_3 \oplus a_{13})$

Table 4.16: Testing $\mathbf{T}_{out}$ entries for 1's in order to determine the 4 signals that will be used to extend $\mathbf{T}_{in}$. i is the row number, $1 \leq i \leq 8$.

$\mathbf{T}_{out}$	$(i,1)$	$(i,2)$	$(i,3)$	$(i,4)$	$(i,5)$	$(i,6)$	$(i,7)$	$(i,8)$	$(i,9)$	$(i,10)$
term1	b_0	-	b_{03}	b_{01}	b_{02}	a_0	-	a_{03}	a_{01}	a_{02}
term2	b_{12}	b_1	-	b_{01}	b_{13}	a_{12}	a_1	-	a_{01}	a_{13}
term3	b_{12}	b_{23}	b_2	-	b_{02}	a_{12}	a_{23}	a_2	-	a_{02}
term4	-	b_{23}	b_{03}	b_3	b_{13}	-	a_{23}	a_{03}	a_3	a_{13}

By using conditional statements that test whether the $\mathbf{T}_{out}$ entry is zero or one, we can determine which columns of Table 4.16 will be XORed together to get the 4 terms that will be ANDed with e_0, e_1, e_2 and e_3. According to (4.40), row 1 of $\mathbf{T}_{out}$ will be used to get the 4 terms used to derive s_7 and row 2 will be used for s_6, etc. As a result, we will have an additional 32 signals, termed $\mathbf{g}_{32}$.

The 32 signals, $\mathbf{g}_{32}$, will be used to extend the 20×8 input transformation matrix $\mathbf{T}_{in}$. $\mathbf{g}_{32}$ will be later used by the 32NAND+8XOR4 block shown in Figure 4.10 to get the output of the S-box, by performing and AND operation between e_0, e_1, e_2 and e_3 and the corresponding 4 signals from $\mathbf{g}_{32}$. We use Matlab® to extend each 20×8 input transformation matrix $\mathbf{T}_{in}$ in (4.33) into the 52×8 input transformation matrix $\mathbf{T}'_{in}$, where

$$\mathbf{T}'_{in} = \begin{bmatrix} \mathbf{T}_{in} \\ \mathbf{T}_{in_{32}} \end{bmatrix} \tag{4.41}$$

where $\mathbf{g}_{32} = \mathbf{T}_{in_{32}}\mathbf{g}$

The focused-search logic minimization algorithm is then used to perform an exhaustive search over all possible $\mathbf{T}'_{in}$ matrices (a total of 2040 matrices) in order to find the smallest number of XOR2 gates required to implement $\mathbf{T}'_{in}$. We noticed that for certain $\mathbf{T}'_{in}$ matrices, some rows are repeated versions of each other. As a result those rows will be implemented by the logic minimization algorithm using the same formulations, which lower the area of implementation. We will have 52 formulations resulting from the logic minimization algorithm for each $\mathbf{T}'_{in}$ matrix. As shown in Figure 4.10, the first 20 signals, $\mathbf{g}_{20}$ will be used by the exponentiation block. The following 32 signals, $\mathbf{g}_{32}$, will be used by the 32NAND+8XOR4 block to get the output of the S-box $\mathbf{s}$. e.g. s_7 can be computed by performing 4 AND operations between e_0, e_1, e_2, e_3 and the logic minimization results for rows $21-24$ of $\mathbf{T}'_{in}$, $o_{20} - o_{23}$, then XOR the four resulting terms from the AND operations using 1 XOR4, which is a two level XOR tree consisting of 3 XOR2. Replacing AND by NAND in the 32NAND+8XOR4 block will not affect the operation of XOR2, as inverting both inputs of an XOR2 does not affect its operation.

We applied the focused-search logic minimization algorithm [69] on each $\mathbf{T}'_{in}$, with no timing constraints, for all values of the arbitrary binary field element η, a total of $8 \times 255 = 2040$, in order to select the smallest area one. The results of the Matlab® simulations are shown in Figures A.9, A.10, A.11, A.12, A.13, A.14, A.15 and A.16. Those Figures show the number of XOR2 gates used to implement each of the 2040 $\mathbf{T}'_{in}$ as a function of η as well as the input binary matrix $\mathbf{X}^{-1}$.

We can see from the implementation area graph shown in Figure A.9 (where $\nu = 1000$ and Generator #1), at $\eta = 4, 48, 91$ and 153, the implementation area of $\mathbf{T}'_{in}$ is only 37 XOR2 gates. We choose the case of $\eta = 48$ as the maximum delay associated with this circuit is only $7D_X$, instead of $8D_X$ for $\eta = 4, 91, 153$. For this specific $\mathbf{T}'_{in}$, we have to constraint the maximum delay of the first 20 rows, which corresponds to $\mathbf{g}_{20}$, to $3D_X$ so that the delay of the improved fast S-box won't be affected. In order to achieve that, we use the first 20 rows of $\mathbf{T}'_{in}$ as input the focused-search logic minimization algorithm [69], with a timing constraint of $3D_X$. The result of the logic minimization algorithm shows that we will need 4 auxiliary signals that will be needed by the logic minimization algorithm to minimize $\mathbf{T}'_{in}$ such that the delay of $\mathbf{g}_{20}$ is $3D_X$. We find that using those 4 auxiliary signals also reduced the maximum delay of $\mathbf{g}_{32}$ to $6D_X$. The final formulations are shown in Table

Table 4.17: Fast implementation of transformations in (4.41) for $\nu = 1000$, Generator #1 and $\eta = 48$ (Fast 3 design).

$a_0 = a_1 \oplus a_{01}$	$(3D_X)$	$a_1 = g_5 \oplus g_4$	$(1D_X)$	$a_2 = b_3 \oplus o_{44}$	$(3D_X)$
$a_3 = o_{43} \oplus o_{44}$	$(3D_X)$	$b_0 = b_{01} \oplus b_1$	$(3D_X)$	$b_1 = b_2 \oplus b_{12}$	$(2D_X)$
$b_2 = g_7 \oplus g_5$	$(1D_X)$	$b_3 = b_2 \oplus g_1$	$(2D_X)$	$a_{01} = o_{29} \oplus o_{41}$	$(2D_X)$
$a_{02} = g_7$	$(0D_X)$	$a_{03} = o_{29} \oplus o_{43}$	$(3D_X)$	$a_{12} = o_{41} \oplus b_3$	$(3D_X)$
$a_{13} = o_{41} \oplus o_{43}$	$(3D_X)$	$a_{23} = b_3 \oplus o_{43}$	$(3D_X)$	$b_{01} = o_{40} \oplus g_4$	$(2D_X)$
$b_{02} = b_{12} \oplus b_{01}$	$(3D_X)$	$b_{03} = b_{01} \oplus b_{13}$	$(3D_X)$	$b_{12} = g_3 \oplus g_2$	$(1D_X)$
$b_{13} = b_{12} \oplus g_1$	$(2D_X)$	$b_{23} = g_1$	$(0D_X)$	$o_{20} = o_{34} \oplus g_6$	$(5D_X)$
$o_{21} = o_{29} \oplus g_6$	$(4D_X)$	$o_{22} = a_0 \oplus g_6$	$(4D_X)$	$o_{23} = o_{39} \oplus g_0$	$(5D_X)$
$o_{24} = b_3$	$(2D_X)$	$o_{25} = b_{12}$	$(1D_X)$	$o_{26} = b_1$	$(2D_X)$
$o_{27} = b_{03}$	$(3D_X)$	$o_{28} = b_2 \oplus g_0$	$(2D_X)$	$o_{29} = a_1 \oplus t_0$	$(3D_X)$
$o_{30} = g_2$	$(0D_X)$	$o_{31} = g_6 \oplus g_0$	$(1D_X)$	$o_{32} = b_1 \oplus g_6$	$(3D_X)$
$o_{33} = g_5 \oplus g_1$	$(1D_X)$	$o_{34} = o_{28} \oplus o_{37}$	$(4D_X)$	$o_{35} = t_0 \oplus o_{20}$	$(6D_X)$
$o_{36} = a_{13} \oplus g_5$	$(4D_X)$	$o_{37} = b_{01} \oplus g_3$	$(3D_X)$	$o_{38} = b_{12} \oplus o_{28}$	$(3D_X)$
$o_{39} = b_{23} \oplus o_{51}$	$(4D_X)$	$o_{40} = a_3$	$(3D_X)$	$o_{41} = a_{12}$	$(3D_X)$
$o_{42} = a_1$	$(1D_X)$	$o_{43} = a_{03}$	$(3D_X)$	$o_{44} = a_{23}$	$(3D_X)$
$o_{45} = a_{13}$	$(3D_X)$	$o_{46} = a_0$	$(3D_X)$	$o_{47} = a_{01}$	$(2D_X)$
$o_{48} = a_1 \oplus o_{28}$	$(3D_X)$	$o_{49} = b_2 \oplus o_{37}$	$(4D_X)$	$o_{50} = b_3 \oplus g_0$	$(3D_X)$
$o_{51} = b_1 \oplus g_4$	$(3D_X)$	$t_0 = t_3 \oplus g_3$	$(2D_X)$	$t_1 = a_1 \oplus t_2$	$(2D_X)$
$t_2 = g_2 \oplus g_0$	$(1D_X)$	$t_3 = g_7 \oplus g_6$	$(1D_X)$		

4.17.

The implementation in Table 4.17 requires 41 XOR2 gates with a maximum propagation delay of 6 XOR2 gates. We can notice from table 4.17 that the maximum delay of all 20 bits of $\mathbf{g}_{20}$ in (4.41) is still $3D_X$, as a result, the critical path delay of the new S-box will not be increased. The maximum delay of $\mathbf{g}_{32}$ signal is $6D_X$. The $\mathbf{g}_{32}$ signals will be implemented in parallel to the exponentiation and subfield inverter block, thus not affecting the S-box delay.

We will use the formulations in Table 4.17 to build the fast S-box in Figure 4.10. We will use the improved exponentiation block and the improved subfield inverter derived in section 4.2.1 and section 4.2.2, respectively. We will refer to this design as Fast 3.

4.5.1 Complexity Analysis

The space and time complexities of the modified fast architecture Fast 3 derived in section 4.5, along with all the blocks used in Figure 4.10 are summarized in Table 4.18. The corresponding GEs of all the blocks are also presented. GE is the chip areas in terms of an equivalent of 2-input NAND gates. The provided GEs are based on the used 65nm CMOS technology.

4.5.2 Comparisons of the Space and Time Complexities

In this section, we compare the space and time complexities of the modified fast S-box architectures, Fast 2 and Fast 3, against the fastest S-boxes available in the literature. Table 4.19 and Table 4.20 provide the gate count and time delay, respectively of two S-boxes and compare them with the proposed architectures. We chose two of the fastest schemes, namely Reyhani et al. fast design [69] and Maximov et al. fast S-box [54]. We provides the GE for all S-box architectures using the corresponding GE of the gates from the CMOS 65nm technology. The CPD was obtained by the CAD tool after coding the four S-boxes listed in VHDL.

4.5.3 ASIC Synthesis Results and Comparison

CAD tools are very fast and accurate at evaluating complex delay models. We use Synopsys Design Compiler® in order to give an estimate the critical path delay, as well as the area and power

Table 4.18: Space and time complexities of the modified fast S-box architecture, Fast 3.

Block/ Target	Space Complexity[*]							Time Complexity
	X	ND	NR	AOI12	OAI212	NT	GE	
Extended Input Transformation Block								
$\mathbf{T}'_{in}$ Table 4.17	41						82	$6D_X$[**]
$GF((2^4)^2)$ Inversion and Affine Transformation								
Imp. Exp.	12	6	6				36	$2D_X + 1D_{NR}$
Imp. Inv.		4		4	4	4	20	$1D_{AOI12} + 1D_{OAI212} + 1D_{NT}$
32ND2+8XR4	24	32					80	$2D_X + 1D_{ND}$
Total	36	42	6	4	4	4	136	$4D_X + 1D_{ND} + 1D_{NR} +$ $1D_{AOI12} + 1D_{OAI212} + 1D_{NT}$
Total Complexity of Modified Fast S-Box (Figure 4.10)								
Fast 3	77	42	6	4	4	4	218	$7D_X + 1D_{ND} + 1D_{NR} +$ $1D_{AOI12} + 1D_{OAI212} + 1D_{NT}$

[*]All GEs values are estimated using STM 65nm technology where X is XOR2/XNOR2 = 2GEs, ND is NAND2 = 1GE, NR is NOR2 = 1GE, AOI12 is AND2 to NOR2 = 1.25GEs, OAI212 is 2 OR2 to NAND3 = 2GEs, NT is NOT = 0.75GEs.

[**]The maximum delay for $\mathbf{g}_{20}$ signals is $3D_X$.

Table 4.19: Space complexity comparison of different fast S-boxes.

S-boxes	Gate count[*]									GE
	X	AD	ND	N3	NR	AOI12	OAI212	NT	MUX21	
Reyhani Fast [69]	79	-	39	4	3	-	-	4	-	208
Maximov Fast [54]	78	4	37	-	5	-	-	-	6	215
Fast 2 (Proposed)	88	-	51	4	3	-	-	4	-	238
Fast 3 (Proposed)	77	-	42	-	6	4	4	4	-	218

[*]All GEs values are estimated using STM 65nm technology where X is XOR2/XNOR2 = 2GEs, AD is AND2 =1.25GEs, ND is NAND2 = 1GE, N3 is NAND3 =1.25GEs, NR is NOR2 = 1GE, AOI12 is AND2 into NOR2 = 1.25GEs, OAI212 is 2 OR2 into NAND3 = 2GEs, NT is NOT = 0.75GEs, MUX21 is 2-to-1 noninverting MUX =2GEs.

Table 4.20: Time complexity comparison for different fast S-boxes.

S-Boxes	CPD[*]
Reyhani Fast [69]	$11D_X + 5D_{ND} + 1D_{NT}$
Maximov Fast [54]	$8D_X + 2D_{NR} + 1D_{AD} + 1D_{MUX21}$
Fast 2 (Proposed)	$8D_X + 5D_{ND} + 1D_{NT}$
Fast 3 (Proposed)	$7D_X + 1D_{ND} + 1D_{NR} + 1D_{AOI12} + 1D_{OAI212} + 1D_{NT}$

[*]X = XOR2/XNOR2, AD = AND2, ND = NAND2, NR = NOR2, AOI12 = AND2 into NOR2, OAI212 = 2 OR2 into NAND3, NT = NOT = 0.75GEs, MUX21 = 2-to-1 noninverting MUX .

Table 4.21: ASIC comparisons of the S-Box architectures

S-Box	Area		Delay	Power	Area-Time
	μm^2	GE	ns	μW	product
Reyhani Fast [69]	432.64	208	0.779697	42.750	162.177
Maximov Fast [54]	447.2	**215**	0.686869	41.398	147.677
Fast 2 (Proposed)	495.04	238	0.627886	47.807	149.437
Fast 3 (Proposed)	453.44	218	**0.626698**	46.580	**136.62**

estimates of the different S-boxes considered. The overall implementation results are proposed in Table 4.21. We propose the Fast 3 structure. Logic synthesis was done using VHDL as a design entry to the Synopsys Design Vision®. The technology library used was STM 65-nm CMOS standard library and the CORE65LPSVT standard cell library which is optimized for low power applications. The area, delay and power for all the considered S-boxes are generated by the CAD tool with relaxed constraints at a clock frequency of 100 MHz. We coded the above-mentioned fast S-boxes in VHDL and present their ASIC results in Table 4.21. For each and every code, we verified the codes using the S-box testbenches and Modelsim®. The delay value results from the CAD tool confirm that Fast 3 design is the fastest and the most efficient (as measured by Area-Time product) S-box among the four S-boxes considered.

From Table 4.21, the modified fast S-box, Fast 3, is the currently fastest S-box in the literature, to our knowledge. The new S-box is about 24% faster than the fast S-box design in [69] with 5% increase in area. It is 10% faster than the fast S-box design in [54] with 1% more area.

In Figure 4.11, we evaluate the proposed Fast 3 and Fast 2 S-box designs against previous works under different delay constraints as an input design requirement for the CAD tool. The CAD tool is free to select different output strengths for each gate to reduce the delay of the critical path. Gates with higher output strengths have a smaller delay than their regular counterparts at a slight increase of the synthesized area. We have started at the non constraint delay and area values. Timing constraint are then reduced by 20 ps until the timing constraint could not be met and the slack is violated. Fast 3 design has the same speed as Fast 2, however it has 8% less area. Fast 3 unconstrained design is the fastest, however Maximov fast S-box achieves a lower area at tighter timing constraints.

4.6 Conclusion

In this Chapter, we improved the area and the delay of the exponentiation stage. We designed a new subfield inverter block and the synthesis results confirm an improvement in the area of the subfield inverter. We also modified the input and output transformation matrices of the lightweight S-box, by multiplying the input of the S-box by a binary field element, and then study the effect of the new transformations on the implementation area and the speed of the S-box. We also used the method of logical effort [74] to linearly model the critical path delay of the S-box in order to standardize the calculation of the delay among different technology libraries used in the synthesis process. We also derived two modified fast S-box architectures, Fast 1 and Fast 2, and the synthesis results show a great improvement in the speed. An extended input transformation matrix is derived and a search is conducted in order to find a lower area improved fast architecture. The synthesis results show that the new fast design, Fast 3, is 24% faster than Reyhani fast S-box [69], with only 5% increase in area. It is also 10 % faster, with just 1% more area, than Maximov fast S-box [54], which is currently the fastest S-box in the literature, up to our knowledge.

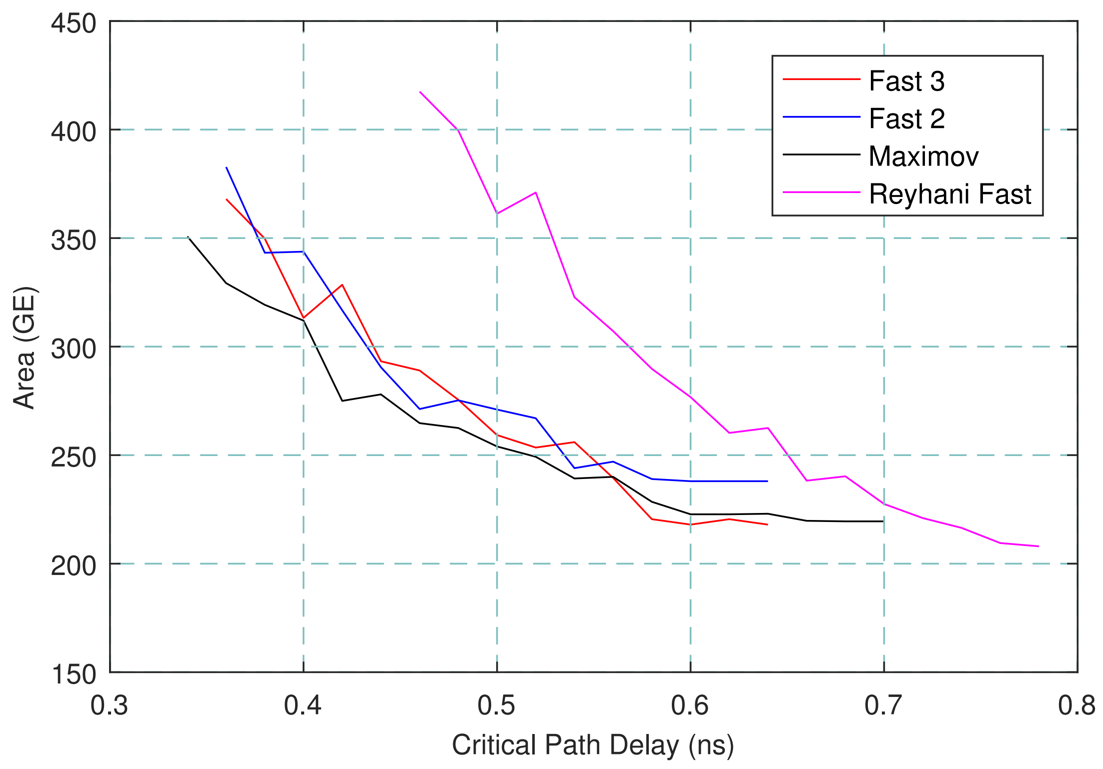


Figure 4.11: Area in GEs of the proposed Fast 3 and Fast 2 S-boxes as compared to previous work [69, 54] at different input delay constraints for the STM 65nm library.

Chapter 5

Improved $GF(((2^2)^2)^2)$ Forward AES S-box

5.1 Introduction

In this chapter, we will improve the AES forward S-box using tower field representation $GF(((2^2)^2)^2)$ over normal basis that was introduced in [70]. The input and output transformation matrices of the tower field S-box are modified, by multiplying the input to the S-box by an 8-bit binary field element that ranges from (1-255). A search is conducted in order to find a lower area input and output mappings of the S-box. The improved S-box is smaller than the original design [70] by 4.25 GEs and smaller than the lightweight S-box [69] by 8.75 GEs in STM 65nm technology, where GE is the equivalent of NAND2 gates. Due to the different field representation selected for the input and output mappings of the improved S-box than [70], a slightly different inversion circuit is derived in this chapter. The modified inversion circuit has the same area and delay as the original one in [70].

5.2 Architecture of the Improved AES S-box

In this section, we introduce the overall architecture for the improved forward S-box. The design introduced by Reyhani et al. [70] in 2019 will be the starting point of this Chapter. Using the idea proposed in [54], we will multiply the input of the S-box by a binary field element η that ranges from $(1 - 255)$ and track the complexity of the new input and output transformation matrices in order to find a lower area mappings. The inversion circuit of the S-box would need to be changed a little if the field chosen based on the new transformations change, which was the case as will be illustrated in section 5.4.

The improved architecture is highlighted in Figure 5.1. As shown in this figure, the inversion is done over the tower field $GF(((2^2)^2)^2)$. In the beginning, the 8-bit input g is multiplied by a binary field element η, that ranges from $(1 - 255)$. The multiplication by an 8-bit binary field element can be represented in matrix form as $\mathbf{C}$. The matrix $\mathbf{C}$ was derived before in (4.15), and will be used

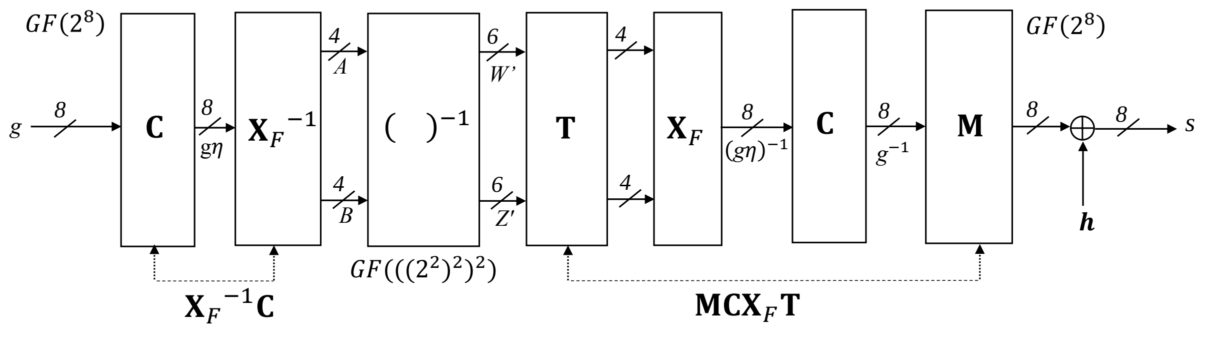


Figure 5.1: The overall architecture of the improved forward S-box using $GF(((2^2)^2)^2)$ in the NB representation.

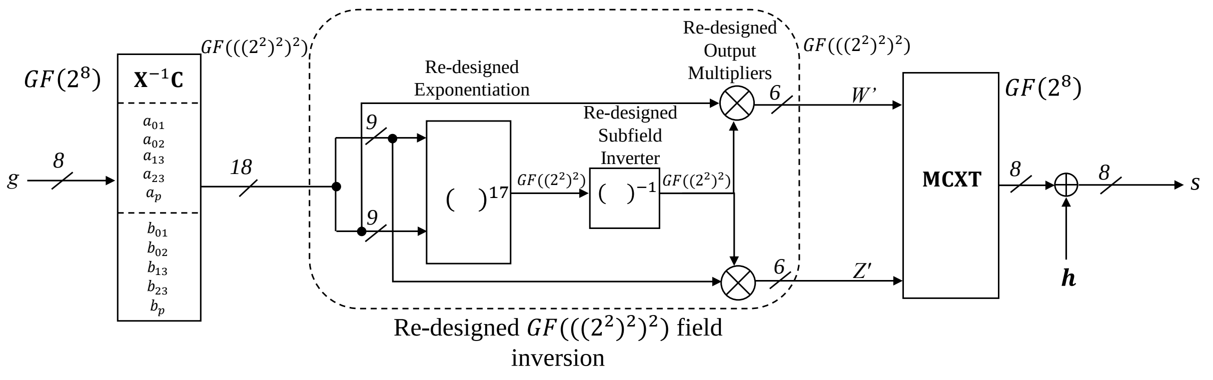

$GF(2^8)$
g
8
$X^{-1}C$
a_{01} a_{02} a_{13} a_{23} a_p
b_{01} b_{02} b_{13} b_{23} b_p
$GF(((2^2)^2)^2)$
18
Re-designed Exponentiation
9
9
$(\ )^{17}$
$GF((2^2)^2)$
Re-designed Subfield Inverter
$(\ )^{-1}$
$GF((2^2)^2)$
Re-designed Output Multipliers
6
6
$GF(((2^2)^2)^2)$
W'
Z'
MCXT
$GF(2^8)$
8
8
h
s
Re-designed $GF(((2^2)^2)^2)$ field inversion

Figure 5.2: The overall architecture of the improved forward S-box using $GF(((2^2)^2)^2)$ in the NB representation.

here to get the new input mapping of the forward S-box as $\mathbf{X}_{new}^{-1} = \mathbf{X_F}^{-1}\mathbf{C}$. The 8-bit input g is processed through the input isomorphic mapping ($\mathbf{X}_{new}^{-1} = \mathbf{X}_F^{-1}\mathbf{C}$) to find the equivalent elements A and B, of 4-bit each, in the tower field representation. The $GF((2^2)^2)$ elements A and B represent the inputs of the inversion circuit. The 18×8 input mapping$\mathbf{T}_{in_{new}}$ generates A and B, of 4-bit each, and it also generates $\{a_{01}, a_{02}, a_{13}, a_{23}, a_p\}$ and $\{b_{01}, b_{02}, b_{13}, b_{23}, b_p\}$, where $a_{jk} = a_j \oplus a_k$ and $b_{jk} = b_j \oplus b_k$ for $0 \leq j, k \leq 3$, and $j \neq k$. Also, $a_p = a_{02} \oplus a_{13}$ and $b_p = b_{02} \oplus b_{13}$ which are the parities of A and B, respectively. The modified input mapping matrix can be written as

$$\mathbf{T}_{in_{new}} = \begin{bmatrix} \mathbf{X}_F^{-1}\mathbf{C} \\ \mathbf{a}_{ij}^{tr} \\ \mathbf{b}_{ij}^{tr} \end{bmatrix} \text{for}\, 0 \leq i, j \leq 3, i \neq j \tag{5.1}$$

Similarly, the output mapping implements the last stage of the output multipliers, $\mathbf{T}$ (Sec. 5.4.5). The $GF((2^2)^2)$ multiplier output is represented as six terms (6-bit output) denoted by W' and Z' as RNB6 representation, before reduced to four terms (4-bit output). Therefore, the output mapping of the forward S-box accepts two redundant RNB6 elements with a total of 12 bits, and generates an 8-bit output in the binary field of $GF(2^8)$. To cancel the effect of multiplying the input of the S-box by η, the matrix $\mathbf{C}$ is also included in the output mapping of the forward S-box as shown in Figure 5.1. As a result, the new output mapping is the 8×12 output matrix defined as

$$\mathbf{T}_{out_{new}} = \mathbf{MCX}_F\mathbf{T} \tag{5.2}$$

The improved architecture is highlighted in Figure 5.2.

5.3 Input and Output Mappings of the Improved AES S-box

In this section, we will study the effect of multiplying the input of the S-box by $\eta = (1 - 255)$, on the input and output mappings between the AES binary field in $GF(2^8)$ and the 16 possible $GF(((2^2)^2)^2)$ tower field representations over normal basis considerd in [70]. We noticed that among the 16 possible tower field representations, only 4 field representations are unique and the other 12 representations are replicas of those 4, where the corresponding isomorphic mapping matrices, $\mathbf{X_F}^{-1}$, are row-wise related. The 4 unique field representations correspond to $\{\nu = [0010], N = [01]\}, \{\nu = [0010, N = [10]\}\{\nu = [0111], N = [01]\}, \{\nu = [0111], N = [10]\}$. Since there are 4 possible unique mappings per field [70], corresponding to Generators #1, #2, #3 and #4, leading to a total of $4 \times 4 = 16$ mappings. Also, we have 255 different values of η, then there are a total of $16 \times 255 = 4080$ different input transformation matrices and 4080 different output transformation matrices that could be used for input and output mappings, respectively.

For each $\mathbf{X}_F^{-1}$, we derived, using Matlab®, 16 different input transformation matrices $\mathbf{T}_{in_{new}}$ as defined by (5.1), and 16 different output transformation matrices $\mathbf{T}_{out_{new}}$ as defined by (5.2). Then

Table 5.1: The number of XOR2 gates and the maximum delay results for some $\mathbf{T}_{out_{new}}$ matrices.

v	N	Generator #	η	# XOR2	Max. Delay
[0010]	[01]	4	1	17[1]	$4D_X$[1]
[0010]	[10]	3	49	19	$5D_X$
[0010]	[10]	3	210	18	$4D_X$
[0010]	[10]	3	227	19	$6D_X$

[1]The output mapping of the original forward S-box [70] reuires 17 XOR2 gates with a maximum delay of $4D_X$.

for each $\mathbf{X}_F^{-1}$, we multiply by $\eta = (1 - 255)$ using the matrix form, $\mathbf{C}$, of η derived in (4.15). We applied the Focused-Search logic-minimization algorithm proposed in [69] on each $\mathbf{T}_{in_{new}}$ for all the values of the arbitrary binary field element η, i.e. a total of $16 \times 255 = 4080$, in order to select the smallest area one. The results of the Matlab® simulations are shown in Figures B.1, B.2, B.3, B.4, B.5, B.6, B.7, B.8, B.9, B.10, B.11, B.12, B.13, B.14, B.15 and B.16. Those figures show the number of XOR2 gates used to implement each of the 4080 $\mathbf{T}_{in_{new}}$ as a function of η, as well as the maximum delay associated with each matrix.

In the original forward S-box [70], the used field is $\{v = [0010] = \omega\alpha^4, N = [01] = \omega^2\}$ and Generator #4. The input mapping requires 19 XOR2 gates with a maximum delay of $7D_X$, while the output mapping requires 17 XOR2 gates with a maximum delay of $4D_X$. As can be seen from Figure B.7, the field represented by $\{v = [0010] = \omega\alpha^4, N = [10] = \omega\}$ and Generator #3 gives a more compact implementation of 16 XOR2 gates for $\mathbf{T}_{in_{new}}$. This compact implementation of $\mathbf{T}_{in_{new}}$ occurs at $\eta = 49, 210, 227$, where the associated maximum delay for all three transformations is $7D_X$. To decide which η could be better in saving the area of the improved S-box, we run the focused-search logic minimization algorithm on the corresponding $\mathbf{T}_{out_{new}}$ matrices and the results are summarized in Table 5.1. For all the 3 cases in Table 5.1, we choose the second one, where the improved output mapping requires only 18 XOR2 gates with a maximum delay of $4D_X$.

We can conclude from the Matlab® simulations for $\mathbf{T}_{in_{new}}$ and $\mathbf{T}_{out_{new}}$ that multiplication of the S-box input by a binary field element $\eta = (1 - 255)$ could improve the area of the input and output transformations of the S-box. Using the field $\{v = [0010] = \omega\alpha^4, N = [10] = \omega\}$, Generator #3 and $\eta = 210$ reduce the area of the input and output mapping of the forward S-box in [70] by 2 XOR2 gates, while keeping the maximum delay at the same level. However, the inversion circuit of the S-box needs to be slightly changed due to the different field representation we are using than [70]. In the following section, we will introduce the equations used for the input and output mapping of the improved S-box

5.3.1 Input Mapping of the Improved AES S-box

We used Matlab® to get the input transformation matrix defined in (5.1). The binary input matrix $\mathbf{X}_F^{-1}$ corresponds to the selected field $\{v = [0010] = \omega\alpha^4, N = [10] = \omega\}$ and Generator #3 is used. The multiplication matrix $\mathbf{C}$ is computed from (4.15) for the case of $\eta = 210$. As a result, the input mapping of the improved forward S-box is defined as follows:

$$\mathbf{i} = \left[\begin{array}{c} a_0 \\ a_1 \\ a_2 \\ a_3 \\ \hline b_0 \\ b_1 \\ b_2 \\ b_3 \\ \hline a_{01} \\ a_{02} \\ a_{13} \\ a_{23} \\ a_p \\ \hline b_{01} \\ b_{02} \\ b_{13} \\ b_{23} \\ b_p \end{array}\right] = \mathbf{T}_{in_{new}} \times \mathbf{g} = \left[\begin{array}{c} 0\,0\,0\,0\,0\,0\,1\,0 \\ 1\,0\,1\,0\,1\,1\,1\,0 \\ 0\,1\,1\,1\,1\,1\,0\,0 \\ 0\,1\,1\,1\,0\,0\,0\,0 \\ \hline 0\,1\,0\,1\,1\,0\,0\,1 \\ 0\,0\,0\,0\,0\,1\,0\,0 \\ 0\,0\,0\,1\,1\,0\,0\,0 \\ 1\,1\,1\,1\,1\,0\,0\,0 \\ \hline 1\,0\,1\,0\,1\,1\,0\,0 \\ 0\,1\,1\,1\,1\,1\,1\,0 \\ 1\,1\,0\,1\,1\,1\,1\,0 \\ 0\,0\,0\,0\,1\,1\,0\,0 \\ 1\,0\,1\,0\,0\,0\,0\,0 \\ \hline 0\,1\,0\,1\,1\,1\,0\,1 \\ 0\,1\,0\,0\,0\,0\,0\,1 \\ 1\,1\,1\,1\,1\,1\,0\,0 \\ 1\,1\,1\,0\,0\,0\,0\,0 \\ 1\,0\,1\,1\,1\,1\,0\,1 \end{array}\right] \times \begin{bmatrix} g_7 \\ g_6 \\ g_5 \\ g_4 \\ g_3 \\ g_2 \\ g_1 \\ g_0 \end{bmatrix}. \tag{5.3}$$

The implementation of (5.3) is highlighted in Table 5.2 and its complexity is presented below.

Proposition 5.3.1 *The input mapping of the improved forward S-box can be implemented using 16 XOR2 gates, with a maximum delay of $7D_X$, where D_X is the delay of a 2-input XOR2 gate.*

Table 5.2: Equations used to implement the input mappings of the improved forward S-box.

$a_0 = g_1$	$0D_X$	$b_0 = b_2 \oplus b_{02}$	$2D_X$
$a_1 = a_{01} \oplus g_1$	$3D_X$	$b_1 = g_2$	$0D_X$
$a_2 = b_{13} \oplus g_7$	$5D_X$	$b_2 = g_4 \oplus g_3$	$1D_X$
$a_3 = a_{23} \oplus a_2$	$6D_X$	$b_3 = b_2 \oplus b_{23}$	$3D_X$
$a_{01} = a_{23} \oplus a_p$	$2D_X$	$b_{01} = b_o \oplus g_2$	$3D_X$
$a_{02} = a_2 \oplus g_1$	$6D_X$	$b_{02} = g_6 \oplus g_0$	$1D_X$
$a_{13} = a_p \oplus a_{02}$	$7D_X$	$b_{13} = b_3 \oplus g_2$	$4D_X$
$a_{23} = g_3 \oplus g_2$	$1D_X$	$b_{23} = a_p \oplus g_6$	$2D_X$
$a_p = g_7 \oplus g_5$	$1D_X$	$b_p = b_{23} \oplus b_{01}$	$4D_X$

5.3.2 Output Mapping of the Improved AES S-box

As mentioned before, the output mapping of the improved forward S-box calculates $\mathbf{s} = (\mathbf{T}_{out_{new}} \times \mathbf{o}) \oplus \mathbf{h}$, where $\mathbf{T}_{out_{new}} = \mathbf{MCX}_F\mathbf{T}$ which is provided below for the selected field $\{\nu = [0010] = \omega\alpha^4, N = [10] = \omega\}$ and Generator #3 at $\eta = 210$.

$$\begin{bmatrix} s_7 \\ s_6 \\ s_5 \\ s_4 \\ s_3 \\ s_2 \\ s_1 \\ s_0 \end{bmatrix} = \begin{bmatrix} 1\,0\,1\,0\,0\,0\,0\,1\,0\,0\,0\,1 \\ 0\,0\,0\,0\,0\,0\,1\,1\,1\,1\,0\,0 \\ 0\,1\,0\,1\,0\,0\,0\,0\,0\,0\,0\,0 \\ 1\,0\,0\,0\,1\,0\,1\,0\,0\,0\,1\,0 \\ 1\,1\,0\,1\,1\,0\,1\,1\,1\,0\,0\,1 \\ 1\,1\,1\,0\,0\,1\,0\,1\,1\,0\,1\,1 \\ 0\,1\,0\,0\,0\,1\,0\,1\,0\,1\,0\,0 \\ 1\,1\,0\,1\,1\,0\,1\,0\,1\,0\,0\,0 \end{bmatrix} \times \begin{bmatrix} w'_0 \\ w'_1 \\ w'_2 \\ w'_3 \\ w'_4 \\ w'_5 \\ z'_0 \\ z'_1 \\ z'_2 \\ z'_3 \\ z'_4 \\ z'_5 \end{bmatrix} \oplus \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}, \tag{5.4}$$

where w'_i and z'_i, $0 \leq i \leq 5$, as required by (5.25) in Sec. 5.4.5. The implementation of (5.4) is highlighted in Table 5.3. In this table, a 3-input XOR (XOR3) and a 3-input XNOR (XNOR3) are represented by $\oplus_3$ and $\odot_3$, respectively. The delay of these gates is represented by D_{X3} in this table. It is noted that each XOR3 gate can be implemented using 2 XOR2 gates and one XNOR3 gate can be implemented by one XOR2 and one XNOR2 gates with a delay of $2D_X$. As a result, one can obtain the complexities of this block.

Table 5.3: Equations used to implement the proposed output mappings of the improved forward-only S-box.

$s_7 = \oplus_3(t_1, w_0, w_2)$	$D_{X3} + D_X$	$s_6 = \odot_3(t_3, z_0, z_2)$	$D_{X3} + D_X$
$s_5 = w_1 \odot w_3$	D_X	$s_4 = t_0 \oplus z_4$	$D_{X3} + D_X$
$s_3 = t_1 \odot s_0$	$2D_{X3} + D_X$	$s_2 = \oplus_3(z_2, z_4, t_4)$	$2D_{X3} + 2D_X$
$s_1 = t_2 \odot t_3$	$2D_X$	$s_0 = \oplus_3(t_0, s_5, z_2)$	$2D_{X3}$
$t_0 = \oplus_3(w_0, w_4, z_0)$	D_{X3}	$t_1 = z_1 \oplus z_5$	$1D_X$
$t_2 = w_1 \oplus w_5$	$1D_X$	$t_3 = z_1 \oplus z_3$	$1D_X$
$t_4 = t_2 \oplus s_7$	$D_{X3} + 2D_X$		

Proposition 5.3.2 *The output mapping of the improved forward S-box can be implemented using 8 XOR2/XNOR2 and 5 XOR3/XNOR3 gates with the maximum delay of* $2D_{X3} + 2D_X$. *If only XOR2/XNOR2 is used, it requires 18 XOR2/XNOR2gates, with a maximum delay of* $4D_X$, *where* D_X *and* D_{X3} *are the delay of a XOR2/XNOR2 and a XOR3/XNOR3, respectively.*

5.4 Modified Inversion over $GF(((2^2)^2)^2)$

Inversion over tower field $GF(((2^2)^2)^2)$ is the core block in the forward S-box. In this section, we slightly modify the inversion design over $GF(((2^2)^2)^2)$ provided in [70] in order to use the selected field $\{\nu = [0010] = \omega\alpha^4, N = [10] = \omega\}$. As shown in Figure 5.1, the inversion over tower field consists of exponentiation, subfield inversion and output multipliers. In this section, we illustrate the designs for these three blocks according to the new field used.

The first block in Figure 5.1 is the exponentiation block, which generates $D = g^{17}$ as required in (2.12). Since we selected $\nu = \omega\alpha^4$, (2.12) can be written as $D = A \times B + (A+B)^2\omega\alpha^4$ which requires multiplication, squaring and scaling over $GF((2^2)^2)$. In the following subsections, we first analyze these operations to find the coordinates of $D = (d_0d_1d_2d_3) = (d_0\omega + d_1\omega^2)\alpha + (d_2\omega + d_3\omega^2)\alpha^4$ for the new tower field.

5.4.1 Multiplication over $GF((2^2)^2)$

Let $A = A_0\alpha + A_1\alpha^4$ and $B = B_0\alpha + B_1\alpha^4$ be subfield elements over $GF((2^2)^2)$ where $A_i, B_i \in GF(2^2)$, $i = 0, 1$. Also, we will use the RNB representation to define the output of their multiplication as $C = A \times B = \hat{C}_0\alpha + \hat{C}_1\alpha^4 + \hat{C}_2$, where $\hat{C}_0, \hat{C}_1, \hat{C}_2 \in GF(2^2)$ can be computed as

$$\begin{aligned} \hat{C}_0 &= A_0B_0, \\ \hat{C}_1 &= A_1B_1, \\ \hat{C}_2 &= (A_0 + A_1)(B_0 + B_1)\omega. \end{aligned} \tag{5.5}$$

Here, we used $\alpha\alpha^4 = \eta$ as implied from (2.9), and $\eta = \omega$ following the mapping selection. As seen from (5.5), multiplication in $GF((2^2)^2)$ requires three multiplications over $GF(2^2)$. Since the polynomial used to define the $GF(2^2)$ field in (2.10) is an AOP with degree 2 ($m = 2$) and is irreducible, we use the type-I optimal normal basis (ONB-I) multiplication scheme proposed in [66] for $m = 2$.

Lemma 5.4.1 *From [66], let* $A_0 = a_0\omega + a_1\omega^2 \in GF(2^2)$ *and* $B_0 = b_0\omega + b_1\omega^2 \in GF(2^2)$ *be represented in the ONB-I* $\{\omega, \omega^2\}$. *Then, the coordinates of their multiplication, represented in the redundant normal basis (RNB)* $\{\omega, \omega^2, 1\}$, *can be calculated as follows:* $\hat{C}_0 = A_0B_0 = a_0b_0\omega + a_1b_1\omega^2 + a_{01}b_{01}$ *where* $a_{01} = a_0 \oplus a_1$ *and* $b_{01} = b_0 \oplus b_1$.

Since $A_1 = a_2\omega + a_3\omega^2 \in GF(2^2)$ and $B_1 = b_2\omega + b_3\omega^2 \in GF(2^2)$, one can use Lemma 5.4.1 to find $(A_0 + A_1)(B_0 + B_1) = a_{02}b_{02}\omega + a_{13}b_{13}\omega^2 + a_pb_p$. As a result, (5.5) can be computed as

$$\begin{aligned}\hat{C}_0 &= a_0b_0\omega + a_1b_1\omega^2 + a_{01}b_{01}\\ \hat{C}_1 &= a_2b_2\omega + a_3b_3\omega^2 + a_{23}b_{23}\\ \hat{C}_2 &= a_pb_p\omega + a_{02}b_{02}\omega^2 + a_{13}b_{13}\end{aligned} \tag{5.6}$$

where $a_{jk} = a_j \oplus a_k$ and $b_{jk} = b_j \oplus b_k$ for $0 \le j,k \le 3$, and $j \ne k$. Also, $a_p = a_{02} \oplus a_{13}$ and $b_p = b_{02} \oplus b_{13}$ which are the parities of A and B, respectively.

5.4.2 Squaring with Scaling

If we denote $V = (A + B)^2\omega\alpha^4 = V_0\alpha + V_1\alpha^4$, with $V_0, V_1 \in GF((2^2)^2)$. Using (5.5), one can find $A^2 = A_0^2\alpha + A_1^2\alpha^4 + (A_0 + A_1)^2\omega$ and similar expression for B^2. Since $(A + B)^2 = A^2 + B^2$, we can simplify the computation of the coefficients of V to:

$$\begin{aligned}V_0 &= (A_0 + A_1 + B_0 + B_1)^2\omega^2\\ V_1 &= (A_1 + B_1)^2\omega\end{aligned} \tag{5.7}$$

Representing the equations in $GF(2^2)$, similar to (5.6), one can find that

$$\begin{aligned}V_0 &= (a_{13} \oplus b_{13}) + (a_{02} \oplus b_{02})\omega\\ V_1 &= (a_2 \oplus b_2) + (a_3 \oplus b_3)\omega^2\end{aligned} \tag{5.8}$$

where $a_{13} = a_1 \oplus a_3$, $a_{02} = a_0 \oplus a_2$ and $b_{13} = b_1 \oplus b_3$, $b_{02} = b_0 \oplus b_2$.

5.4.3 Modified Exponentiation Computation

The output of exponentiation block, i.e., D, can be computed by adding $C = \hat{C}_0\alpha + \hat{C}_1\alpha^4 + \hat{C}_2$ with $V = V_0\alpha + V_1\alpha^4$. Then, one can obtain the coefficients of D with respect to the RNB as $D = \hat{D}_0\alpha + \hat{D}_1\alpha^4 + \hat{D}_2$, where

$$\begin{aligned}\hat{D}_0 &= \hat{C}_0 + V_0\\ \hat{D}_1 &= \hat{C}_1 + V_1\\ \hat{D}_2 &= \hat{C}_2\end{aligned} \tag{5.9}$$

Using (5.6) and (5.8), we can simplify the RNB coefficients of D to:

$$\begin{aligned}\hat{D}_0 &= (a_0b_0 \oplus a_{02} \oplus b_{02})\omega + a_1b_1\omega^2 + (a_{13} \oplus b_{13} \oplus a_{01}b_{01})\\ \hat{D}_1 &= a_2b_2\omega + (a_3b_3 \oplus a_3 \oplus b_3)\omega^2 + (a_{23}b_{23} \oplus a_2 \oplus b_2)\\ \hat{D}_2 &= a_pb_p\omega + a_{02}b_{02}\omega^2 + a_{13}b_{13}\end{aligned} \tag{5.10}$$

We can represent D in the NB as $D = D_0\alpha + D_1\alpha^4$, with $D_i = \hat{D}_i + \hat{D}_2$ for $i = 0, 1$. Then, the $GF(2^2)$ coefficients of $D = (d_0\omega + d_1\omega^2)\alpha + (d_2\omega + d_3\omega^2)\alpha^4$ can be found as:

$$\begin{aligned}d_0 &= (a_0b_0 \oplus (a_p \vee b_p) \oplus a_{01}b_{01} \oplus a_{13}b_{13})\\ d_1 &= (a_1b_1 \oplus a_{02}b_{02} \oplus (a_{13} \vee b_{13}) \oplus a_{01}b_{01})\\ d_2 &= ((a_2 \vee b_2) \oplus a_pb_p \oplus a_{23}b_{23} \oplus a_{13}b_{13})\\ d_3 &= (a_3b_3 \oplus (a_{23} \vee b_{23}) \oplus a_{02}b_{02} \oplus a_{13}b_{13})\end{aligned} \tag{5.11}$$

Here, we use $a_{02} \oplus b_{02} \oplus a_{13} \oplus b_{13} = a_p \oplus b_p$ then $a_p \oplus b_p \oplus a_pb_p = a_p \vee b_p$ to obtain d_0. For the computation in d_1, we replace $a_{13} \oplus b_{13} \oplus a_{13}b_{13}$ by $a_{13} \vee b_{13}$. For d_2 we replace $a_2 \oplus b_2 \oplus a_2b_2$ by $a_2 \vee b_2$. Similarly, $a_3 \oplus b_3 \oplus a_2 \oplus b_2 = a_{23} \oplus b_{23}$ then $a_{23} \oplus b_{23} \oplus a_{23}b_{23} = a_{23} \vee b_{23}$ is used to simplify d_3. Also, for low cost implementation, we replace all AND and OR operations in (5.11) to NAND and NOR operations, respectively, without changing their functions. As a result, one can obtain the following lemma for the exponentiation block.

Lemma 5.4.2 *The coordinates of the new exponentiation block that computes* $D = g^{17} = (d_0d_1d_2d_3) = (d_0\omega + d_1\omega^2)\alpha + (d_2\omega + d_3\omega^2)\alpha^4$ *are obtained as follows.*

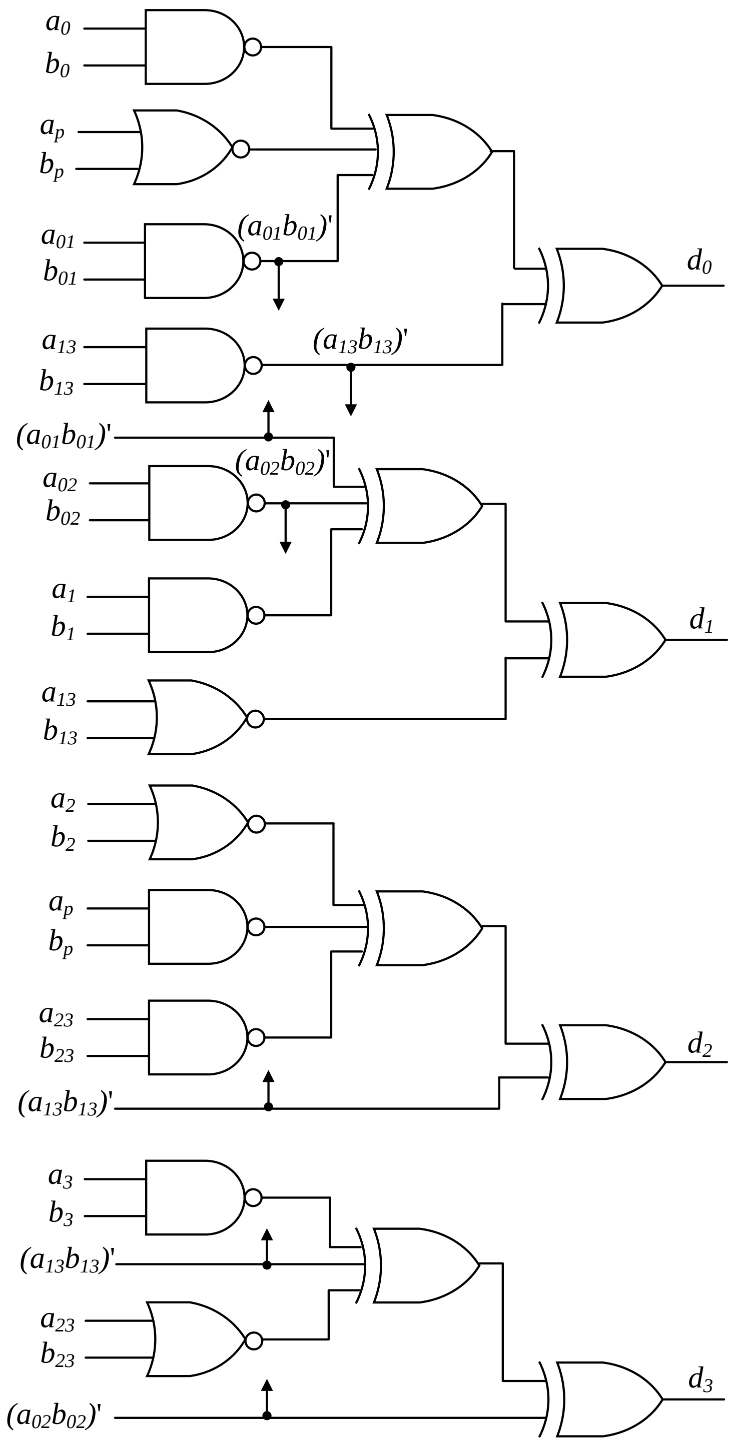


Figure 5.3: The architecture of the modified exponentiation block.

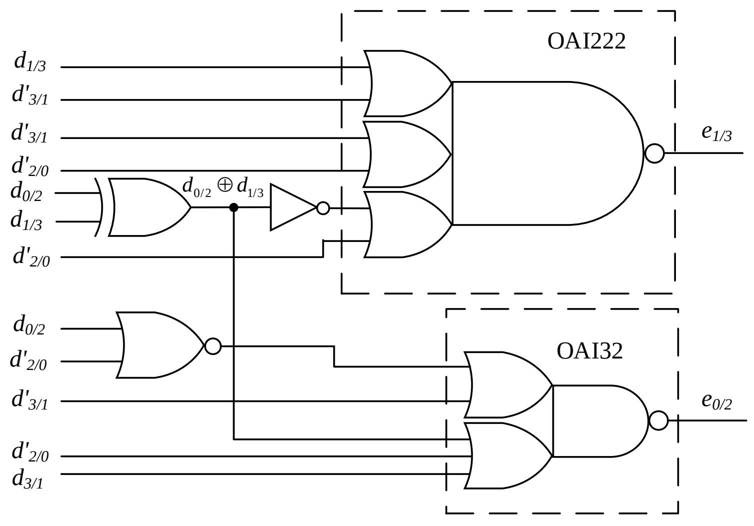


Figure 5.4: The modified subfield inverter block.

Table 5.4: The truth table of the inverter over $GF((2^2)^2)$ using the field represented by $\{\nu = [0010] = \omega\alpha^4, N = [10] = \omega\}$.

$d_0d_1d_2d_3$	$e_0e_1e_2e_3$	$d_0d_1d_2d_3$	$e_0e_1e_2e_3$
0000	0000	1000	0010
0001	1100	1001	1101
0010	1000	1010	0101
0011	0100	1011	1110
0100	0011	1100	0001
0101	1010	1101	1001
0110	0111	1110	1011
0111	0110	1111	1111

$$\begin{aligned}
d_0 &= \overline{a_0b_0} \oplus \overline{(a_p \vee b_p)} \oplus \overline{a_{01}b_{021}} \oplus \overline{a_{13}b_{13}} \\
d_1 &= \overline{a_1b_1} \oplus \overline{a_{02}b_{02}} \oplus \overline{(a_{13} \vee b_{13})} \oplus \overline{a_{01}b_{01}} \\
d_2 &= \overline{(a_2 \vee b_2)} \oplus \overline{a_pb_p} \oplus \overline{a_{23}b_{23}} \oplus \overline{a_{13}b_{13}} \\
d_3 &= \overline{a_3b_3} \oplus \overline{(a_{23} \vee b_{23})} \oplus \overline{a_{02}b_{02}} \oplus \overline{a_{13}b_{13}}
\end{aligned} \tag{5.12}$$

The architecture of new exponentiation block used in the new inversion over $GF(((2^2)^2)^2)$ is shown in Figure 5.3 and its complexity is presented below.

Proposition 5.4.3 *In the improved S-box, the exponentiation computation block (Figure 5.3), with original formulations presented in (5.12), consists of 4 XOR2 (2-input XOR), 4 XOR3 (3-input XOR), 8 NAND2 (2-input NAND), and 4 NOR2 (2-input NOR) gates with the critical path delay of $D_X + D_{X3} + D_{NAND}$ where D_X, D_{X3}, and D_{NAND} are the delays of one XOR2 gate, one XOR3 and one NAND2 gate, respectively.*

5.4.4 Modified Subfield Inversion over $GF((2^2)^2)$

Let $D = (d_0d_1d_2d_3)$ be the input of the subfield inverter. The output of the new subfield inverter is $E = D^{-1} = (e_0e_1e_2e_3) = E_0\alpha + E_1\alpha^4 \in GF((2^2)^2)$, where $E_0 = e_0\omega + e_1\omega^2$, $E_1 = e_2\omega + e_3\omega^2$ and $e_i \in GF(2)$, $0 \leq i \leq 3$, are its coordinates.

The subfield inverter is a 4-bit input 4-bit output combinational circuit which can be designed using truth table design method. We used Matlab® to find the inversion truth table as shown in Table 5.4. Using this table one can obtain the following.

Lemma 5.4.4 *Let $E = D^{-1}$ represented by $E = (e_0e_1e_2e_3) = E_0\alpha + E_1\alpha^4 \in GF((2^2)^2)$, where $E_0 = e_0\omega + e_1\omega^2$ and $E_1 = e_2\omega + e_3\omega^2$. Then, the coordinates of E_0 and E_1 are*

$$\begin{aligned}
e_0 &= d_2d'_3(d_0 \odot d_1) \vee d_3(d_0 \vee d'_2) \\
e_1 &= d_2(d_0 \oplus d_1) \vee d_3(d'_1 \vee d_2)
\end{aligned} \tag{5.13}$$

and

$$\begin{aligned} e_2 &= d_0 d_1'(d_2 \odot d_3) \vee d_1(d_2 \vee d_0') \\ e_3 &= d_0(d_2 \oplus d_3) \vee d_1(d_3' \vee d_0) \end{aligned} \tag{5.14}$$

respectively.

We implemented the above formulations and several other equivalent functions to obtain the optimum design with the least area and delay in the ASIC implementation. In order to reduce the area and improve the speed of the ASIC implementations of (5.13) and (5.14), we use compound OR-AND-Invert (OAI) gates, such as OAI22 and OAI32 gates, instead of using AND/OR as well as NAND/NOR gates [70]. One can convert the formulations in Lemma 5.4.4 to the following using Boolean algebra and De Morgan's laws. As a result, we conclude the following.

Corollary 5.4.5 *[70] The coordinates of E_0 can be found from*

$$\begin{aligned} e_0 &= ((d_2' \vee d_3 \vee (d_0 \oplus d_1))(d_3' \vee (d_0 \vee d_2')'))' \\ e_1 &= ((d_2' \vee (d_0 \oplus d_1)')(d_3' \vee (d_1' \vee d_2)'))' \end{aligned} \tag{5.15}$$

Similarly, the coordinates of E_1 can be found by switching all indices in (5.15) between 0 and 2, i.e., $0 \leftrightarrow 2$ and between 1 and 3, i.e., $1 \leftrightarrow 3$.

Using the compound gate OAI222 (instead of OAI22 and a NOR2), as shown in Figure 5.4, we can slightly improve the area of subfield inversion. One can find the following formulations by applying Boolean algebra and De Morgan's laws to (5.15) for e_0 and e_1:

Corollary 5.4.6 *The coordinates of E_0 can be found from*

$$\begin{aligned} e_0 &= ((d_2' \vee d_3 \vee (d_0 \oplus d_1))(d_3' \vee (d_0 \vee d_2')'))' \\ e_1 &= ((d_2' \vee (d_0 \oplus d_1)')(d_1 \vee d_3')(d_2 \vee d_3'))' \end{aligned} \tag{5.16}$$

Similarly, the coordinates of E_1 can be found by switching all indices in (5.16) between 0 and 2, i.e., $0 \leftrightarrow 2$ and between 1 and 3, i.e., $1 \leftrightarrow 3$.

Using 4 NOT gates at the inputs of Figure 5.4, one can obtain the following regarding the time and space complexities of the modified subfield inverter.

Proposition 5.4.7 *In the improved forward S-box, the space complexity of the proposed subfield inverter block over $GF((2^2)^2)$ includes 2 XOR2, 2 NOR2, 2 OAI222, 2 OAI32 and 6 NOT gates. The time complexity of the proposed subfield inverter is $1D_{OAI222} + 1D_{XOR2} + 1D_{NOT}$.*

5.4.5 Modified Output Multipliers

One can use the formulations presented in section 5.4.1 to obtain the formulations for two output multipliers that generate $Z = A \times E$ and $W = B \times E$. Starting with the formulations for $Z = A \times E$, let A be represented as in (2.11) and similarly represent E as $E = (e_0\omega + e_1\omega^2)\alpha + (e_2\omega + e_3\omega^2)\alpha^4 \in GF((2^2)^2)$, where E is the field element generated by the subfield inverter, with coordinates of $e_i \in GF(2)$, $i \in [0, 3]$. Let us represent $Z = A \times E$ with respect to the RNB as $Z = A \times E = \hat{Z}_0\alpha + \hat{Z}_1\alpha^4 + \hat{Z}_2$. Then, using (5.6), one can obtain

$$\begin{aligned} \hat{Z}_0 &= a_0e_0\omega + a_1e_1\omega^2 + a_{01}e_{01} \\ \hat{Z}_1 &= a_2e_2\omega + a_3e_3\omega^2 + a_{23}e_{23}, \\ \hat{Z}_2 &= a_pe_p\omega + a_{02}e_{02}\omega^2 + a_{13}e_{13}. \end{aligned} \tag{5.17}$$

To implement (5.17), the a_{ij} and a_p signals are available from the implementation of exponentiation block. However, 5 additional XOR gates are required for the implementation of the e_{ij} and

e_p signals used in (5.17). To reduce the CPD, we convert the representations presented in (5.17) from the RNB to the NB. Using $a_{01}e_{01} = a_{01}e_0 \oplus a_{01}e_1$, we can simplify $\hat{Z}_0$ to

$$\hat{Z}_0 = (a_1e_0 \oplus a_{01}e_1)\omega + (a_0e_1 \oplus a_{01}e_0)\omega^2. \tag{5.18}$$

Similarly, using $a_{23}e_{23} = a_{23}e_2 \oplus a_{23}e_3$, $\hat{Z}_1$ can be simplified to

$$\hat{Z}_1 = (a_3e_2 \oplus a_{23}e_3)\omega + (a_2e_3 \oplus a_{23}e_2)\omega^2. \tag{5.19}$$

The computation of e_p is in the longest path which reduces the speed of the output multiplier. To design a fast multiplier, we use $e_p = e_{02} \oplus e_{13}$ in the expression of $\hat{Z}_2$ in (5.17), so that:

$$\hat{Z}_2 = (a_pe_{02} \oplus a_{02}e_{13})\omega + (a_{02}e_{02} \oplus a_{13}e_{13})\omega^2. \tag{5.20}$$

One can write $Z = (z_0, z_1, z_2, z_3) = Z_0\alpha + Z_1\alpha^4 = (z_0\omega + z_1\omega^2)\alpha + (z_2\omega + z_3\omega^2)\alpha^4$, with $Z_i = \hat{Z}_i + \hat{Z}_2$ for $i = 0, 1$ and so the coordinates of Z are as follows:

$$\begin{array}{rcl} z_0 &=& a_1e_0 \oplus a_{01}e_1 \oplus z'_4 \\ z_1 &=& a_0e_1 \oplus a_{01}e_0 \oplus z'_5 \\ z_2 &=& a_3e_2 \oplus a_{23}e_3 \oplus z'_4 \\ z_3 &=& a_2e_3 \oplus a_{23}e_2 \oplus z'_5 \end{array} \tag{5.21}$$

where

$$\begin{array}{rcl} z'_4 &=& a_pe_{02} \oplus a_{02}e_{13} \\ z'_5 &=& a_{02}e_{02} \oplus a_{13}e_{13} \end{array} \tag{5.22}$$

Similarly, one can optimize the formulations for $W = B \times E$ by replacing the coordinates of A by the ones of B to obtain the coordinates of $W = (w_0, w_1, w_2, w_3) = (w_0\omega + w_1\omega^2)\alpha + (w_2\omega + w_3\omega^2)\alpha^4$ as:

$$\begin{array}{rcl} w_0 &=& b_1e_0 \oplus b_{01}e_1 \oplus w'_4 \\ w_1 &=& b_0e_1 \oplus b_{01}e_0 \oplus w'_5 \\ w_2 &=& b_3e_2 \oplus b_{23}e_3 \oplus w'_4 \\ w_3 &=& b_2e_3 \oplus b_{23}e_2 \oplus w'_5, \end{array} \tag{5.23}$$

where

$$\begin{array}{rcl} w'_4 &=& b_pe_{02} \oplus b_{02}e_{13} \\ w'_5 &=& b_{02}e_{02} \oplus b_{13}e_{13} \end{array} \tag{5.24}$$

Note that the two signals $e_{02} = e_0 \oplus e_2$ and $e_{13} = e_1 \oplus e_3$ are shared among the two multipliers. Instead of using AND gates, one can design the logical circuit of these multipliers using NAND gates by simply replacing all AND operations to NAND [70]. Such replacements do not change the multiplication operation because the two inputs of XOR gates are complemented which result in no change at the output of XOR gates. Using NAND gates is cheaper and faster in the ASIC implementation. We can conclude the following for the multipliers used in the improved forward S-box.

Lemma 5.4.8 *For the improved forward S-box, the output of multipliers, i.e., $W' = (w'_0, w'_1, w'_2, w'_3, w'_4, w'_5)$ and $Z' = (z'_0, z'_1, z'_2, z'_3, z'_4, z'_5)$ are represented in the RNB6 representation and so their coordinates can be obtained as follows*

$$\begin{array}{l|l} w'_0 = \overline{b_1e_0} \oplus \overline{b_{01}e_1} & z'_0 = \overline{a_1e_0} \oplus \overline{a_{01}e_1} \\ w'_1 = \overline{b_0e_1} \oplus \overline{b_{01}e_0} & z'_1 = \overline{a_0e_1} \oplus \overline{a_{01}e_0} \\ w'_2 = \overline{b_3e_2} \oplus \overline{b_{23}e_3} & z'_2 = \overline{a_3e_2} \oplus \overline{a_{23}e_3} \\ w'_3 = \overline{b_2e_3} \oplus \overline{b_{23}e_2} & z'_3 = \overline{a_2e_3} \oplus \overline{a_{23}e_2} \\ w'_4 = \overline{b_pe_{02}} \oplus \overline{b_{02}e_{13}} & z'_4 = \overline{a_pe_{02}} \oplus \overline{a_{02}e_{13}} \\ w'_5 = \overline{b_{02}e_{02}} \oplus \overline{b_{13}e_{13}} & z'_5 = \overline{a_{02}e_{02}} \oplus \overline{a_{13}e_{13}}, \end{array} \tag{5.25}$$

where each coordinate is implemented using two NAND2 and one XOR gates in a two-level of NAND2-XOR2 as shown in Figure 5.2.

As a result, one can obtain the space and time complexities of the two output multipliers in the improved forward S-box as follows.

Proposition 5.4.9 *In the improved forward S-box, the output multipliers consist of 14 XOR2 and 24 NAND2 gates with the longest propagation delay of* $D_{NAND} + 2D_{XOR2}$.

5.5 Implementation Results and Comparisons

In this section, we evaluate the implementation results of all the modified blocks along with the overall improved forward S-box. All the estimate implementation areas (in GEs) in this section are computed based on the area of individual gates in the STM 65nm technology library.

5.5.1 Complexity Analysis of the Improved AES S-box

Table 5.5: Complexity comparison of the improved and the original forward S-box designs.

Design	HW Complexity	GEs*
Reyhani et. al. [70]	48X + 7X3 + 1XN3 + 32ND + 6NR + 2O3 + 2O4 + 6NT	177.75
Improved S-box	44X + 8X3 + 1XN3 + 32ND + 6NR + 2O3 + 2O4 + 6NT	173.5

*All GE values are estimated using STM 65 technology where X is XOR2/XNOR2 = 2GEs, X3 is XOR3 = 3.75GEs, XN3 is XNOR3 = 4GEs, AD is AND2 = 1.25GEs, ND is NAND2 = 1GE, NR isNOR2 = 1GE, O3 is OAI32 = 2GEs,, O4 is OAI222 = 2.5GEs, NT is NOT = 0.75GEs.

Table 5.5 compares the hardware complexity analysis of the improved forward S-box against the original forward S-box [70].

5.5.2 ASIC Implementation Results

We use VHDL coding as a design entry to the Synopsys Design Vision® for logic synthesis. All the individual blocks as well as the improved forward S-box and the original S-box [70] are evaluated using the STM 65nm CMOS standard-cell library. Note that results are collected at conservative wire load models and tthe CPDs are reported by the CAD tool when there is no load to the external output.

5.5.2.1 Individual Blocks

We coded all blocks of the improved S-box in two different modeling methods using VHDL. The first type of modeling is the structural modeling where we code all the blocks exactly as presented in equations and/or figures, with no optimization in the gate selection by the CAD tool. Then, we use behavioral modeling by defining the input/output relationship of each block and allowing optimization by the CAD tool to select the most compact circuit. In Table 5.6, we compare the results of structural modeling against behavioral modeling for each block of the improved forward S-box. In this table, the power consumption values are included as reported by the CAD tool at relaxed constraints using a clock frequency of 100 MHz. Note that the maximum clock frequency can be obtained from the CPD. Later, we evaluate the improved forward S-box design under more tight constraints.

Table 5.6 lists synthesis results for the modified exponentiation block (Sec. 5.4.3), the modified subfield inversion block (Sec. 5.4.4) and the output multipliers (Sec. 5.4.5) using both structural modeling and behavioral modeling. The table shows that structural modeling of all the blocks results in a more compact design than behavioral modeling.

Table 5.6: ASIC synthesis results for the three blocks of the inversion and the entire inversion over the tower field at STM 65nm technology.

Block (S-box)	Imp.	Ref	Area		CPD	Pow.
			μm^2	GE	ns	μW
Exp	Str.	Fig. 5.3	**72.8**	**35**	0.144	2.63
	Beh.	(5.12)	74.88	36	0.144	2.32
sub Inv.	Str.	Fig. 5.4	**40.56**	**19.5**	0.073	1.41
	Beh.	(5.4)	43.68	21	0.083	1.22
Mult.	Str	(5.21),(5.23)	**108.16**	**52**	0.119	3.69
	Beh.	(5.21),(5.23)	108.16	52	0.120	3.74
Cascading the Three Blocks						
Tower Inv.	Str.	Fig. 5.1	**221.52**	**106.5**	0.504	19.35
	Beh.	Fig. 5.1	226.72	109	0.575	19.63

Table 5.7: ASIC Implementation results for the improved forward S-box.

S-box	Area		CPD	Power
	μm^2	GE	ns	μW
Reyhani et. al [70]	369.72	177.75	1.169	34.73
Improved S-box	**360.88**	**173.5**	1.245	34.57

5.5.2.2 Overall Design

In this section, we compare the actual ASIC implementations results of the improved forward S-box against the original compact design [70]. Table 5.7 highlights the ASIC reported results for the improved forward S-box using the STM 65nm CMOS standard-cell library. This table also shows the area in terms of GE, CPD and power consumption as reported by the CAD tool. We compared the results of the improved forward S-box architecture against the original design [70]. As seen from the table, the improved architecture of the forward S-box saves 4.25 GE over the original forward S-box proposed in [70].

In Figure 5.5, we evaluate the improved forward S-box against original work [70] under different delay constraints as an input design requirement for the CAD tool. The CAD tool is free to use gates with different output strengths to improve the delay of each gate at a slight increase of the design area. These figures show that the improved forward S-box circuit results in a more compact design than the previous work [70] across tighter delay constraints.

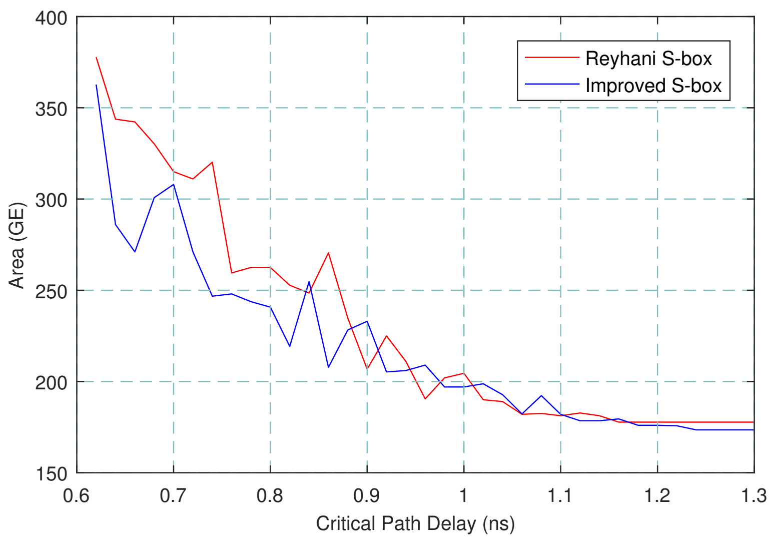


Figure 5.5: Area in GEs of the improved forward S-box circuit as compared to previous work [70] at different input delay constraints targeting the STM 65nm library.

5.6 Conclusion

In this chapter, we improved the area of AES forward S-box design in [70] by 4.25 GEs. Using the focused-search logic minimization algorithm, we have found a new input and output mapping, using the tower field $GF(((2^2)^2)^2)$ which results in more compact linear transformation blocks. For the new field selected, we have derived the formulations and re-designed the corresponding blocks. Moreover, we have verified our designs by extensive simulation codes and implemented our design along with the original one as proposed in [70]. Our analysis and the ASIC implementation results show that the improved S-box outperforms the original scheme in terms of the area.

Chapter 6

Summary and Future Work

6.1 Thesis Summary

In this thesis, we evaluated, improved and compared several forward, inverse and combined AES S-boxes available in the literature in terms of area and/or delay. For each and every code, we verified the codes using the S-box testbenches and Modelsim® and by comparing against the legitimate S-box outputs. We also introduced several improvements to the lightweight and the fast AES S-boxes proposed in [69]. The improved fast S-box, Fast 3, is currently the fastest and most efficient (measured by area $\times$ delay) S-box available in the literature, up to our knowledge. We improved the low area S-box proposed in [70] as well and re-designed the $GF(2^8)$ inversion circuit, correspondingly.

The contributions of this thesis are

- Extensive simulations and comparisons of previous forward, inverse and combined AES S-boxes.
- Improving the area of the composite field AES S-box proposed in [69].
- Optimizing the area of the tower field AES S-box proposed in [70] and re-designing the inversion circuit correspondingly.
- Improving the delay of the composite field AES fast S-box in [69].
- The proposed Fast 3 design is currently the fastest and most efficient AES S-box available in the literature, up to our knowledge.

6.2 Future Work

AES will remain a key security component for a number of sectors like banking and internet commerce. Several internet protocols, e.g. HTTPS, FTPS, SFTP, WebDAVS, OFTP, and AS2 use AES for security purposes [3]. Also, the lightweight implementation of AES-192 and AES-256 will be of greater importance in the era of quantum computers, as a larger key size will be necessary to compensate for the security reduction due to the key search attacks with Grover's algorithm [32]. AES-256 is also used in 8 candidates of the NIST Post-quantum Cryptography (PQC) project [64, 23].

In addition, in 2013, NIST initialized the Lightweight Cryptography (LWC) project [63, 55] to evaluate and standardize lightweight cryptographic algorithms to be used to secure resource-constrained devices (e.g. Radio-Frequency IDentification (RFID) tags [43, 48], IoT [35, 42], sensor networks [46], embedded systems [1], health networks, automotive applications, etc.). 32 candidates are selected to continue for Round 2 and the finalists will be announced by the end of

September 2020. Hardware implementation and benchmarking of the LWC project candidates will be very beneficial in the evaluation process.

An important research direction is the energy efficiency of cryptographic algorithms used to secure resource-constrained devices [27]. The security algorithm should guarantee the throughput requirement of the target application. Also, it should add minimal area, cost, and energy overhead to the system. Most resource-constrained devices are battery operated or depend on harvesting power from the environment. As a result, the energy efficiency of the cryptographic algorithm is of great importance in order to fit security into the energy budget of the constrained device [6, 4, 9, 10].

Another research direction is the transistor level optimization of AES. By utilizing new low power custom gates, an optimized full custom CMOS design of AES could be designed for constrained applications [2].

In the future, the following developments can be pursued

- Fabricate the S-boxes on chip and compare the actual delay and area values against the synthesis results from the CAD tool.
- Optimize at the gate/transistor level in ASIC implementations.
- Hardware implementation of a countermeasure against side-channel attacks for the AES S-box, e.g. threshold implementations [13, 24].
- Design and implementation of a full lightweight AES-128/192/256 encryption/decryption hardware core.
- Design and implementation of an energy-efficient lightweight block cipher suitable for IoT.
- Hardware implementation and benchmarking of NIST Lightweight Cryptography (LWC) projects.

Bibliography

[1] Shady Agwa, Eslam Yahya, and Yehea Ismail. Power efficient aes core for iot constrained devices implemented in 130nm cmos. In *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–4. IEEE, 2017.

[2] Nabihah Ahmad and SM Rezaul Hasan. Low-power compact composite field aes s-box/inv s-box design in 65 nm cmos using novel xor gate. *Integration*, 46(4):333–344, 2013.

[3] Fatih Balli and Subhadeep Banik. Six shades of aes. In *International Conference on Cryptology in Africa*, pages 311–329. Springer, 2019.

[4] Subhadeep Banik, Andrey Bogdanov, Tiziana Fanni, Carlo Sau, Luigi Raffo, Francesca Palumbo, and Francesco Regazzoni. Adaptable aes implementation with power-gating support. In *Proceedings of the ACM International Conference on Computing Frontiers*, pages 331–334, 2016.

[5] Subhadeep Banik, Andrey Bogdanov, Takanori Isobe, Kyoji Shibutani, Harunaga Hiwatari, Toru Akishita, and Francesco Regazzoni. Midori: A block cipher for low energy. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 411–436. Springer, 2015.

[6] Subhadeep Banik, Andrey Bogdanov, and Francesco Regazzoni. Exploring energy efficiency of lightweight block ciphers. In *International Conference on Selected Areas in Cryptography*, pages 178–194. Springer, 2015.

[7] Subhadeep Banik, Andrey Bogdanov, and Francesco Regazzoni. Atomic-aes: A compact implementation of the aes encryption/decryption core. In Orr Dunkelman and Somitra Kumar Sanadhya, editors, *Progress in Cryptology – INDOCRYPT 2016*, pages 173–190, Cham, 2016. Springer International Publishing.

[8] Subhadeep Banik, Andrey Bogdanov, and Francesco Regazzoni. Atomic-aes v 2.0. *IACR Cryptol. ePrint Arch.*, 2016:1005, 2016.

[9] Subhadeep Banik, Andrey Bogdanov, Francesco Regazzoni, Takanori Isobe, Harunaga Hiwatari, and Toru Akishita. Round gating for low energy block ciphers. In *2016 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 55–60. IEEE, 2016.

[10] Subhadeep Banik, Andrey Bogdanov, Francesco Regazzoni, Takanori Isobe, Harunaga Hiwatari, and Toru Akishita. Inverse gating for low energy encryption. In *2018 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 173–176. IEEE, 2018.

[11] Subhadeep Banik, Yuki Funabiki, and Takanori Isobe. More results on shortest linear programs. In *International Workshop on Security*, pages 109–128. Springer, 2019.

[12] Guido Bertoni, Marco Macchetti, Luca Negri, and Pasqualina Fragneto. Power-efficient asic synthesis of cryptographic sboxes. In *Proceedings of the 14th ACM Great Lakes symposium on VLSI*, pages 277–281, 2004.

[13] Begül Bilgin, Benedikt Gierlichs, Svetla Nikova, Ventzislav Nikov, and Vincent Rijmen. A more efficient aes threshold implementation. In *International Conference on Cryptology in Africa*, pages 267–284. Springer, 2014.

[14] Joan Boyar, Morris Dworkin, Rene Peralta, Meltem Turan, Cagdas Calik, and Luis Brandao. CMT: Circuit minimization team, http://www.cs.yale.edu/homes/peralta/CircuitStuff/CMT.html, last accessed on: 1st march, 2020.

[15] Joan Boyar, Magnus Find, and René Peralta. Low-depth, low-size circuits for cryptographic applications. In *Boolean Functions and their Applications BFA- The 2nd International Workshop on, Os, Hordaland, Norway, July 3-8, 2017, Proceedings*, 2017.

[16] Joan Boyar, Philip Matthews, and René Peralta. On the shortest linear straight-line program for computing linear forms. In Edward Ochmanski and Jerzy Tyszkiewicz, editors, *Mathematical Foundations of Computer Science 2008, 33rd International Symposium, MFCS 2008, Torun, Poland, August 25-29, 2008, Proceedings*, volume 5162 of *Lecture Notes in Computer Science*, pages 168–179. Springer, 2008.

[17] Joan Boyar, Philip Matthews, and René Peralta. Logic minimization techniques with applications to cryptology. *Journal of Cryptology*, 26(2):280–312, 2013.

[18] Joan Boyar and René Peralta. A new combinational logic minimization technique with applications to cryptology. In Paola Festa, editor, *Experimental Algorithms, 9th International Symposium, SEA 2010, Ischia Island, Naples, Italy, May 20-22, 2010, Proceedings*, volume 6049 of *Lecture Notes in Computer Science*, pages 178–189. Springer, 2010.

[19] Joan Boyar and René Peralta. A small depth-16 circuit for the AES S-box. In Dimitris Gritzalis, Steven Furnell, and Marianthi Theoharidou, editors, *Information Security and Privacy Research - 27th IFIP TC 11 Information Security and Privacy Conference, SEC 2012, Heraklion, Crete, Greece, June 4-6, 2012, Proceedings*, volume 376 of *IFIP Advances in Information and Communication Technology*, pages 287–298. Springer, 2012.

[20] David Canright. A very compact Rijndael S-box. Technical report, Naval Postgraduate School Technical Report: NPS-MA-05-001, 2005.

[21] David Canright. A very compact S-box for AES. In Josyula R. Rao and Berk Sunar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2005, 7th International Workshop, Edinburgh, UK, August 29 - September 1, 2005, Proceedings*, volume 3659 of *Lecture Notes in Computer Science*, pages 441–455. Springer, 2005.

[22] David Canright and Dag Arne Osvik. A more compact aes. volume 5867, pages 157–169, 08 2009.

[23] Lily Chen, Lily Chen, Stephen Jordan, Yi-Kai Liu, Dustin Moody, Rene Peralta, Ray Perlner, and Daniel Smith-Tone. *Report on post-quantum cryptography*, volume 12. US Department of Commerce, National Institute of Standards and Technology, 2016.

[24] Joan Daemen. Changing of the guards: A simple and efficient method for achieving uniformity in threshold sharing. In *International Conference on Cryptographic Hardware and Embedded Systems*, pages 137–153. Springer, 2017.

[25] Joan Daemen and Vincent Rijmen. *The Design of Rijndaels: AES - The Advanced Encryption Standard*. Information Security and Cryptography. Springer, 2002.

[26] S. N. Dhanuskodi, S. Allen, and D. E. Holcomb. Efficient register renaming architectures for 8-bit aes datapath at 0.55 pj/bit in 16-nm finfet. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, pages 1–14, 2020.

[27] William Diehl, Farnoud Farahmand, Panasayya Yalla, Jens-Peter Kaps, and Kris Gaj. Comparison of hardware and software implementations of selected lightweight block ciphers. In *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*, pages 1–4. IEEE, 2017.

[28] Morris Dworkin. Recommendation for block cipher modes of operation. methods and techniques. Technical report, National Inst of Standards and Technology Gaithersburg MD Computer security Div, 2001.

[29] M. Feldhofer, J. Wolkerstorfer, and V. Rijmen. AES implementation on a grain of sand. *IEE Proceedings - Information Security*, 152:13–20(7), October 2005.

[30] M. Feldhofer, J. Wolkerstorfer, and V. Rijmen. Aes implementation on a grain of sand. *IEE Proceedings - Information Security*, 152(1):13–20, 2005.

[31] PUB 197 FIPS. Specification for the advanced encryption standard (AES). National Institute of Standards and Technology, US Department of Commerce,, November 2001.

[32] Lov K Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 212–219, 1996.

[33] Shay Gueron and Sanu Mathew. Hardware implementation of AES using area-optimal polynomials for composite-field representation GF($(2^4)^2$) of GF(2^8). In Paolo Montuschi, Michael J. Schulte, Javier Hormigo, Stuart F. Oberman, and Nathalie Revol, editors, *23nd IEEE Symposium on Computer Arithmetic, ARITH 2016, Silicon Valley, CA, USA, July 10-13, 2016, Proceedings*, pages 112–117. IEEE Computer Society, 2016.

[34] P. Hamalainen, T. Alho, M. Hannikainen, and T. D. Hamalainen. Design and implementation of low-area and low-power aes encryption hardware core. In *9th EUROMICRO Conference on Digital System Design (DSD'06)*, pages 577–583, 2006.

[35] Chung-Wen Hung and Wen-Ting Hsu. Power consumption and calculation requirement analysis of aes for wsn iot. *Sensors*, 18(6):1675, 2018.

[36] Toshiya Itoh and Shigeo Tsujii. A fast algorithm for computing multiplicative inverses in GF(2^m) using normal bases. *Information and computation*, 78(3):171–177, 1988.

[37] Jérémy Jean, Amir Moradi, Thomas Peyrin, and Pascal Sasdrich. Bit-sliding: A generic technique for bit-serial implementations of SPN-based primitives - applications to AES, PRESENT and SKINNY. In Wieland Fischer and Naofumi Homma, editors, *Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings*, volume 10529 of *Lecture Notes in Computer Science*, pages 687–707. Springer, 2017.

[38] Jérémy Jean, Thomas Peyrin, Siang Meng Sim, and Jade Tourteaux. Optimizing implementations of lightweight building blocks. *IACR Transactions on Symmetric Cryptology*, pages 130–168, 2017.

[39] Yong-Sung Jeon, Young-Jin Kim, and Dong-Ho Lee. A compact memory-free architecture for the AES algorithm using resource sharing methods. *Journal of Circuits, Systems, and Computers*, 19(5):1109–1130, 2010.

[40] Emilia Käsper and Peter Schwabe. Faster and timing-attack resistant aes-gcm. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 1–17. Springer, 2009.

[41] Mehran Mozaffari Kermani and Arash Reyhani-Masoleh. Efficient and high-performance parallel hardware architectures for the AES-GCM. *IEEE Trans. Computers*, 61(8):1165–1178, 2012.

[42] Ho Keun Kim and Myung Hoon Sunwoo. Low power aes using 8-bit and 32-bit datapath optimization for small internet-of-things (iot). *Journal of Signal Processing Systems*, 91(11-12):1283–1289, 2019.

[43] Mooseop Kim, Jaecheol Ryou, Yongje Choi, and Sungik Jun. Low power aes hardware architecture for radio frequency identification. In *International Workshop on Security*, pages 353–363. Springer, 2006.

[44] Thorsten Kranz, Gregor Leander, Ko Stoffelen, and Friedrich Wiemer. Shorter linear straight-line programs for mds matrices. *IACR Transactions on Symmetric Cryptology*, pages 188–211, 2017.

[45] P. C. Liu, H. C. Chang, and C. Y. Lee. A 1.69 gb/s area-efficient AES crypto core with compact on-the-fly key expansion unit. In *European Solid-State Circuits Conference - ESSCIRC, Proceedings*, pages 404–407, Sept 2009.

[46] Zhenglin Liu, Yonghong Zeng, Xuecheng Zou, Yu Han, and Yicheng Chen. A high-security and low-power aes s-box full-custom design for wireless sensor network. In *2007 International Conference on Wireless Communications, Networking and Mobile Computing*, pages 2499–2502. IEEE, 2007.

[47] Samsung Electronics Co. Ltd. STD90/MDL90 0.35μm CMOS Standard Cell Library for Pure Logic/MDL Products Databook, 2000, https://www.digchip.com/datasheets/download_datasheet.php?id=935791&part-number=STD90.

[48] Adam SW Man, Edward S Zhang, Vincent KN Lau, Chi Ying Tsui, and Howard C Luong. Low power vlsi design for a rfid passive tag baseband system enhanced with an aes cryptography engine. In *2007 1st Annual RFID Eurasia*, pages 1–6. IEEE, 2007.

[49] Ben Marshall, G Richard Newell, Dan Page, Markku-Juhani O Saarinen, and Claire Wolf. The design of scalar aes instruction set extensions for risc-v. 2020.

[50] J. L. Massey and J. K. Omura. Computational method and apparatus for finite field arithmetic, 1986. US Patent 4,587,627.

[51] S. Mathew, S. Satpathy, V. Suresh, M. Anders, H. Kaul, A. Agarwal, S. Hsu, G. Chen, and R. Krishnamurthy. 340 mv-1.1 v, 289 gbps/w, 2090-gate NanoAES hardware accelerator with area-optimized encrypt/decrypt $GF(2^4)^2$ polynomials in 22 nm tri-gate CMOS. *IEEE Journal of Solid-State Circuits*, 50(4):1048–1058, 2015.

[52] S. K. Mathew, F. Sheikh, M. Kounavis, S. Gueron, A. Agarwal, S. K. Hsu, H. Kaul, M. A. Anders, and R. K. Krishnamurthy. 53 gbps native $GF(2^4)^2$ composite-field AES-encrypt/decrypt accelerator for content-protection in 45 nm high-performance microprocessor. *IEEE Journal of Solid-State Circuits*, 46(4):767–776, 2011.

[53] Alexander Maximov. Aes mixcolumn with 92 xor gates. *IACR Cryptol. ePrint Arch.*, 2019:833, 2019.

[54] Alexander Maximov and Patrik Ekdahl. New circuit minimization techniques for smaller and faster aes sboxes. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 91–125, 2019.

[55] Kerry McKay, Lawrence Bassham, Meltem Sönmez Turan, and Nicky Mouha. Report on lightweight cryptography. Technical report, National Institute of Standards and Technology, 2016.

[56] Nele Mentens, Lejla Batina, Bart Preneel, and Ingrid Verbauwhede. A systematic evaluation of compact hardware implementations for the rijndael s-box. In *Cryptographers Track at the RSA Conference*, pages 323–333. Springer, 2005.

[57] Amir Moradi, Axel Poschmann, San Ling, Christof Paar, and Huaxiong Wang. Pushing the limits: A very compact and a threshold implementation of aes. In Kenneth G. Paterson, editor, *Advances in Cryptology – EUROCRYPT 2011*, pages 69–88, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[58] Sumio Morioka and Akashi Satoh. An optimized s-box circuit architecture for low power aes design. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 172–186. Springer, 2002.

[59] Kenta Nekado, Yasuyuki Nogami, and Kengo Iokibe. Very short critical path implementation of AES with direct logic gates. In Goichiro Hanaoka and Toshihiro Yamauchi, editors, *Advances in Information and Computer Security - 7th International Workshop on Security, IWSEC 2012, Fukuoka, Japan, November 7-9, 2012, Proceedings*, volume 7631 of *Lecture Notes in Computer Science*, pages 51–68. Springer, 2012.

[60] Yasuyuki Nogami, Kenta Nekado, Tetsumi Toyota, Naoto Hongo, and Yoshitaka Morikawa. Mixed bases for efficient inversion in $F_{((2^2)^2)^2}$ and conversion matrices of subbytes of AES. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 234–247. Springer, 2010.

[61] Christof Paar. *Efficient VLSI architectures for bit parallel computation in Galios fields*. PhD thesis, University of Duisburg-Essen, Germany, 1994.

[62] Christof Paar and Martin Rosner. Comparison of arithmetic architectures for reed-solomon decoders in reconfigurable hardware. In *Proceedings. The 5th Annual IEEE Symposium on Field-Programmable Custom Computing Machines Cat. No. 97TB100186)*, pages 219–225. IEEE, 1997.

[63] NIST Lightweight Cryptography Project. https://csrc.nist.gov/projects/lightweight-cryptography, last accessed on: 1st july, 2020.

[64] NIST Post-Quantum Cryptography Project. https://csrc.nist.gov/Projects/Post-Quantum-Cryptography, last accessed on: 1st july, 2020.

[65] Arash Reyhani-Masoleh and M. Anwar Hasan. A new construction of massey-omura parallel multiplier over GF(2^m). *IEEE Trans. Computers*, 51(5):511–520, 2002.

[66] Arash Reyhani-Masoleh and M. Anwar Hasan. Efficient multiplication beyond optimal normal bases. *IEEE Trans. Computers*, 52(4):428–439, 2003.

[67] Arash Reyhani-Masoleh and M. Anwar Hasan. Low complexity bit parallel architectures for polynomial basis multiplication over GF(2^m). *IEEE Trans. Computers*, 53(8):945–959, 2004.

[68] Arash Reyhani-Masoleh, Mostafa Taha, and Doaa Ashmawy. New area record for the aes combined s-box/inverse s-box. In *2018 IEEE 25th Symposium on Computer Arithmetic (ARITH)*, pages 145–152. IEEE, 2018.

[69] Arash Reyhani-Masoleh, Mostafa Taha, and Doaa Ashmawy. Smashing the implementation records of aes s-box. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 298–336, 2018.

[70] Arash Reyhani-Masoleh, Mostafa Taha, and Doaa Ashmawy. New low-area designs for the aes forward, inverse and combined s-boxes. *IEEE Transactions on Computers*, 2019.

[71] Atri Rudra, Pradeep K. Dubey, Charanjit S. Jutla, Vijay Kumar, Josyula R. Rao, and Pankaj Rohatgi. Efficient Rijndael encryption implementation with composite field arithmetic. In Çetin Kaya Koç, David Naccache, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2001, Third International Workshop, Paris, France, May 14-16, 2001, Proceedings*, volume 2162 of *Lecture Notes in Computer Science*, pages 171–184. Springer, 2001.

[72] Akashi Satoh, Sumio Morioka, Kohji Takano, and Seiji Munetoh. A compact Rijndael hardware architecture with S-box optimization. In Colin Boyd, editor, *Advances in Cryptology - ASIACRYPT 2001, 7th International Conference on the Theory and Application of Cryptology and Information Security, Gold Coast, Australia, December 9-13, 2001, Proceedings*, volume 2248 of *Lecture Notes in Computer Science*, pages 239–254. Springer, 2001.

[73] Ko Stoffelen. Optimizing s-box implementations for several criteria using sat solvers. In *International Conference on Fast Software Encryption*, pages 140–160. Springer, 2016.

[74] Ivan Sutherland, Robert F Sproull, Bob Sproull, and David Harris. *Logical effort: designing fast CMOS circuits*. Morgan Kaufmann, 1999.

[75] Quan Quan Tan and Thomas Peyrin. Improved heuristics for short linear programs. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2020(1):203–230, Nov. 2019.

[76] Stefan Tillich, Martin Feldhofer, and Johann Großschädl. Area, delay, and power characteristics of standard-cell implementations of the aes s-box. In *International Workshop on Embedded Computer Systems*, pages 457–466. Springer, 2006.

[77] Kun-Lin Tsai, Fang-Yie Leu, Ilsun You, Shuo-Wen Chang, Shiung-Jie Hu, and Hoonyong Park. Low-power aes data encryption architecture for a lorawan. *IEEE Access*, 7:146348–146357, 2019.

[78] Rei Ueno, Naofumi Homma, Yukihiro Sugawara, Yasuyuki Nogami, and Takafumi Aoki. Highly efficient GF(2^8) inversion circuit based on redundant GF arithmetic and its application to AES design. In Tim Güneysu and Helena Handschuh, editors, *Cryptographic Hardware*

and Embedded Systems - CHES 2015 - 17th International Workshop, Saint-Malo, France, September 13-16, 2015, Proceedings, volume 9293 of *Lecture Notes in Computer Science*, pages 63–80. Springer, 2015.

[79] Rei Ueno, Sumio Morioka, Naofumi Homma, and Takafumi Aoki. A high throughput/gate AES hardware architecture by compressing encryption and decryption datapaths - toward efficient cbc-mode implementation. In Benedikt Gierlichs and Axel Y. Poschmann, editors, *Cryptographic Hardware and Embedded Systems - CHES 2016 - 18th International Conference, Santa Barbara, CA, USA, August 17-19, 2016, Proceedings*, volume 9813 of *Lecture Notes in Computer Science*, pages 538–558. Springer, 2016.

[80] Rei Ueno, Sumio Morioka, Noriyuki Miura, Kohei Matsuda, Makoto Nagata, Shivam Bhasin, Yves Mathieu, Tarik Graba, Jean Luc Danger, and Naofumi Homma. High throughput/gate aes hardware architectures based on datapath compression. *IEEE Transactions on Computers*, 2019.

[81] Andrea Visconti, Chiara Valentina Schiavo, and René Peralta. Improved upper bounds for the expected circuit complexity of dense systems of linear equations over gf (2). *Information processing letters*, 137:1–5, 2018.

[82] Charles C. Wang, Trieu-Kien Truong, Howard M. Shao, Leslie J. Deutsch, Jim K. Omura, and Irving S. Reed. VLSI architectures for computing multiplications and inverses in GF(2^m). *IEEE Trans. Computers*, 34(8):709–717, 1985.

[83] Neil HE Weste and David Harris. *CMOS VLSI design: a circuits and systems perspective*. Pearson Education India, 2015.

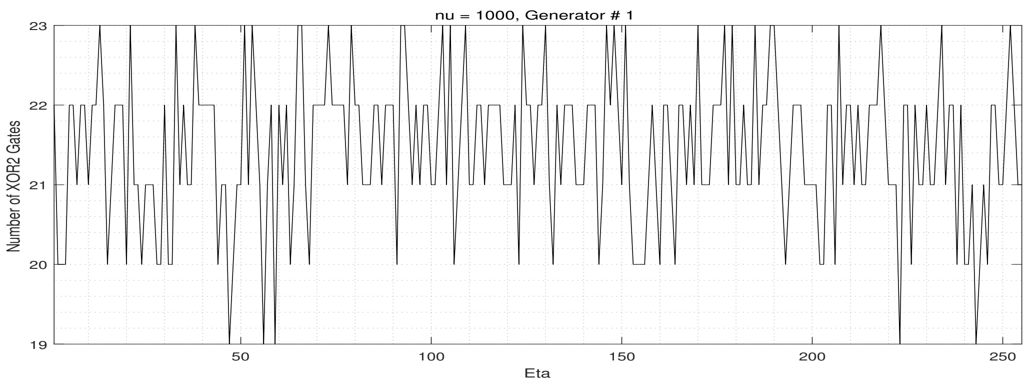
[84] Clifford Wolf. Yosys open synthesis suite.(2016). *URL http://www. clifford. at/yosys*, 2016.

[85] Johannes Wolkerstorfer, Elisabeth Oswald, and Mario Lamberger. An asic implementation of the AES SBoxes. In *Topics in Cryptology — CT-RSA 2002*, pages 67–78, 2002.

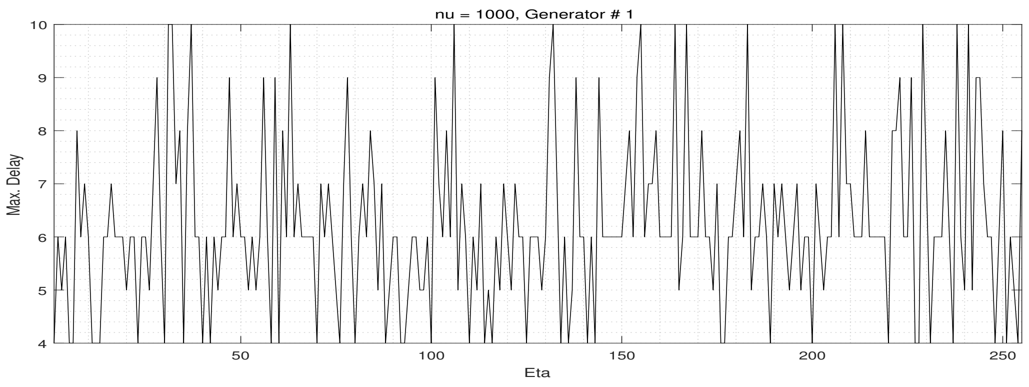
[86] W. Zhao, Y. Ha, and M. Alioto. Aes architectures for minimum-energy operation and silicon demonstration in 65nm with lowest energy per encryption. In *2015 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 2349–2352, 2015.

Appendix A

Additional Figures for Chapter 3

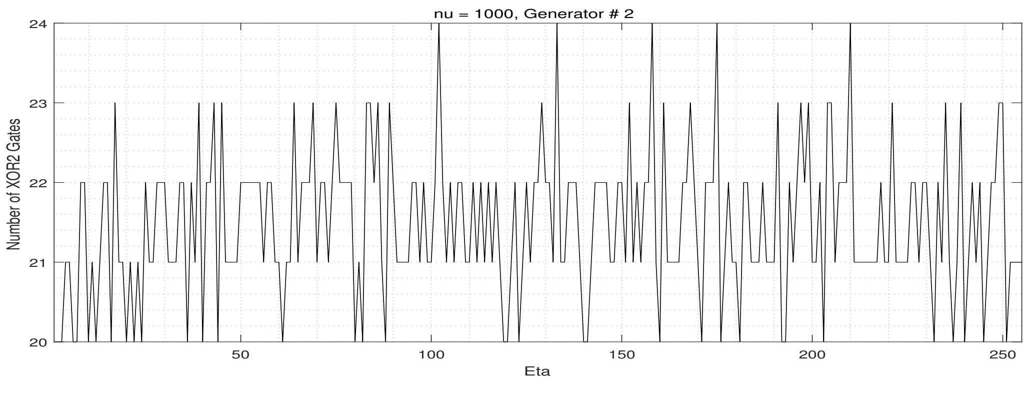


(a)

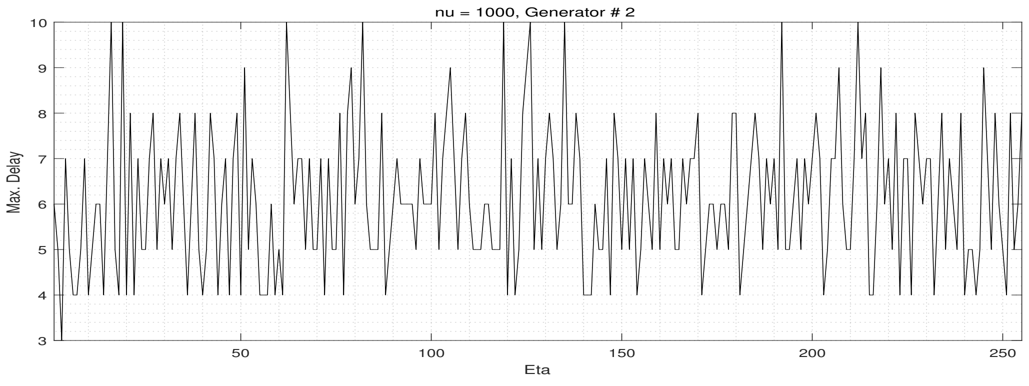


(b)

Figure A.1: (a) The number of XOR2 gates required to implement $\mathbf{T}_{in}$ as a function of η for the case of $\nu = 1000$, Generator #1.(b) The maximum delay of $\mathbf{T}_{in}$ as a function of η for the case of $\nu = 1000$, Generator #1.

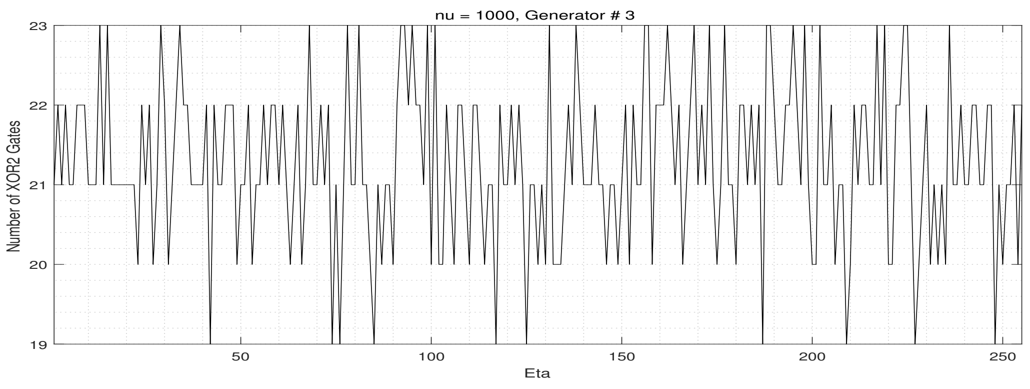


(a)

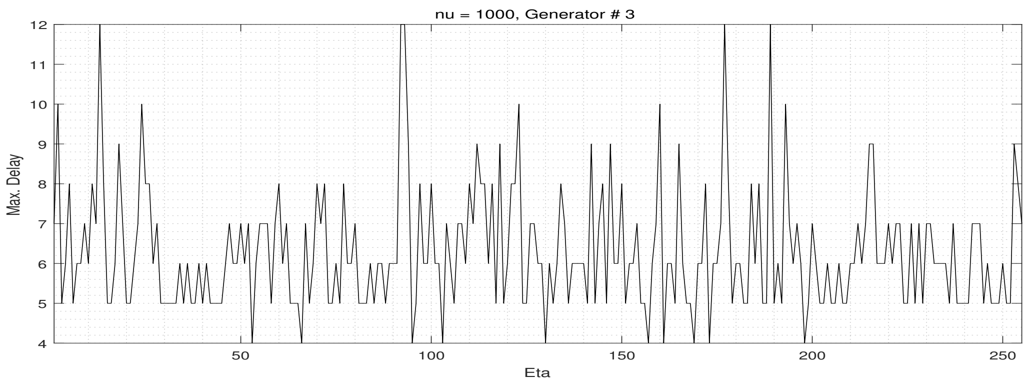


(b)

Figure A.2: (a) The number of XOR2 gates required to implement $\mathbf{T}_{in}$ as a function of η for the case of $\nu = 1000$, Generator #2.(b) The maximum delay of $\mathbf{T}_{in}$ as a function of η for the case of $\nu = 1000$, Generator #2.

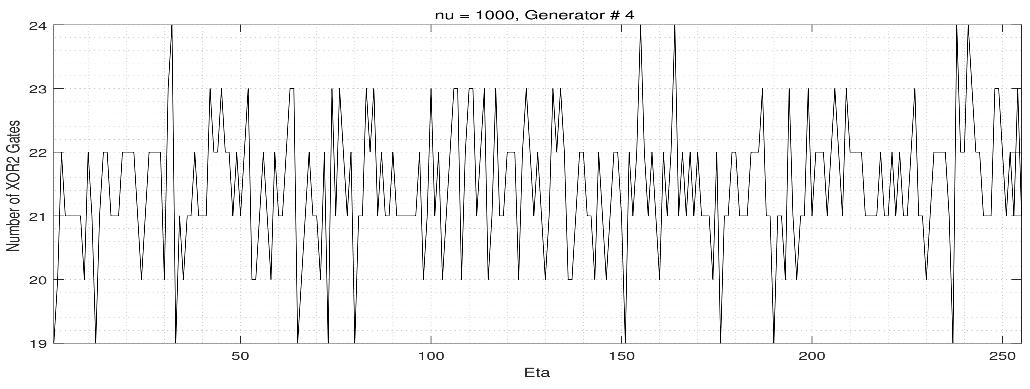


(a)

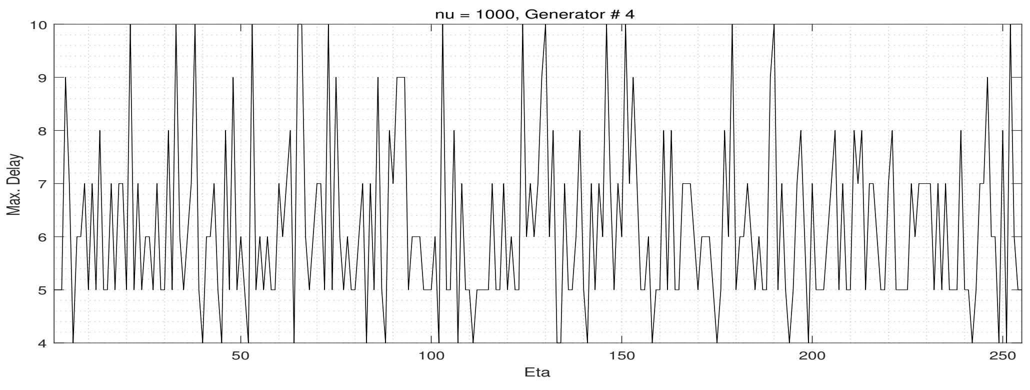


(b)

Figure A.3: (a) The number of XOR2 gates required to implement $\mathbf{T}_{in}$ as a function of η for the case of $\nu = 1000$, Generator #3.(b) The maximum delay of $\mathbf{T}_{in}$ as a function of η for the case of $\nu = 1000$, Generator #3.

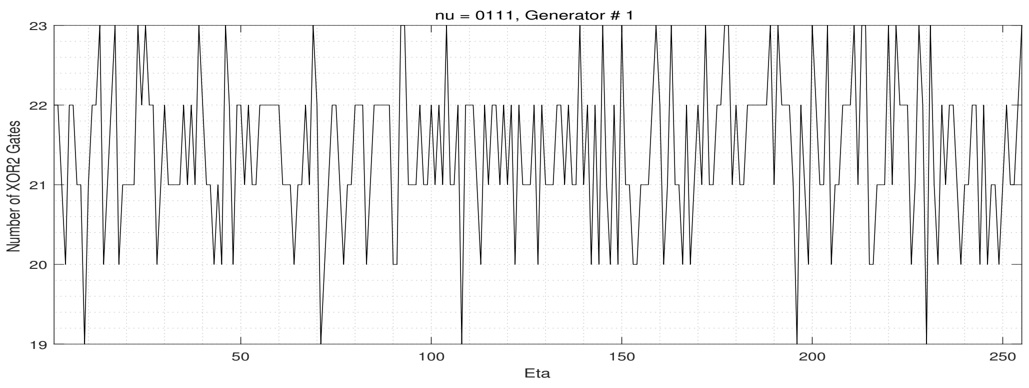


(a)

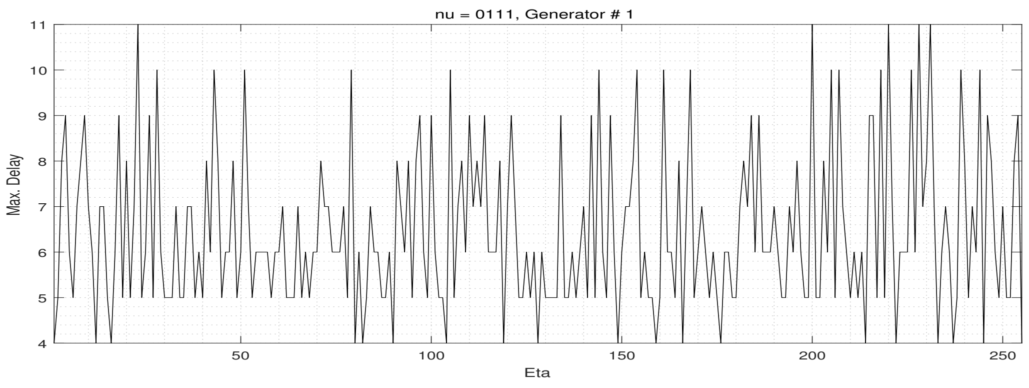


(b)

Figure A.4: (a) The number of XOR2 gates required to implement $\mathbf{T}_{in}$ as a function of η for the case of $\nu = 1000$, Generator #4.(b) The maximum delay of $\mathbf{T}_{in}$ as a function of η for the case of $\nu = 1000$, Generator #4.

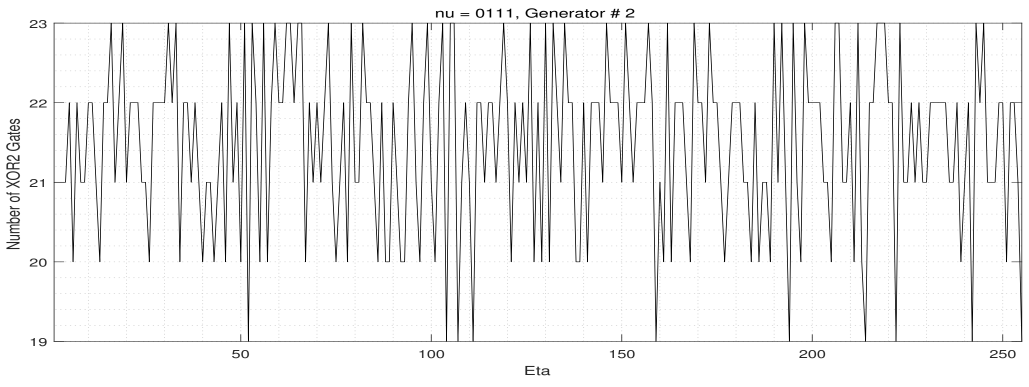


(a)

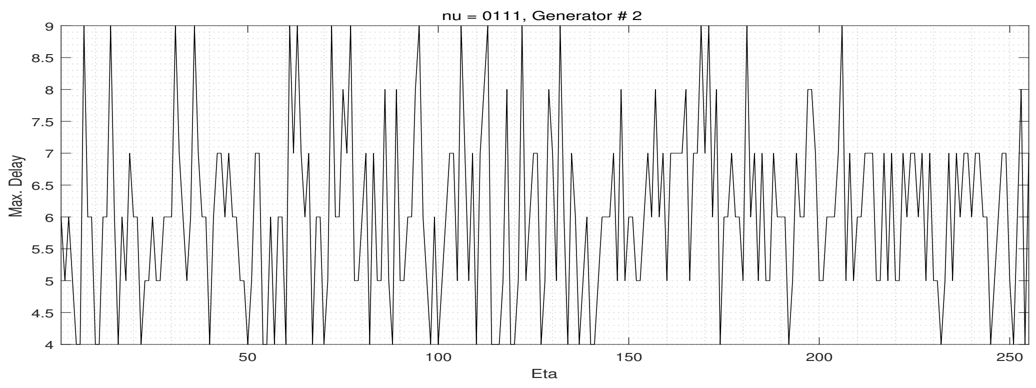


(b)

Figure A.5: (a) The number of XOR2 gates required to implement $\mathbf{T}_{in}$ as a function of η for the case of $\nu = 0111$, Generator #1.(b) The maximum delay of $\mathbf{T}_{in}$ as a function of η for the case of $\nu = 0111$, Generator #1.

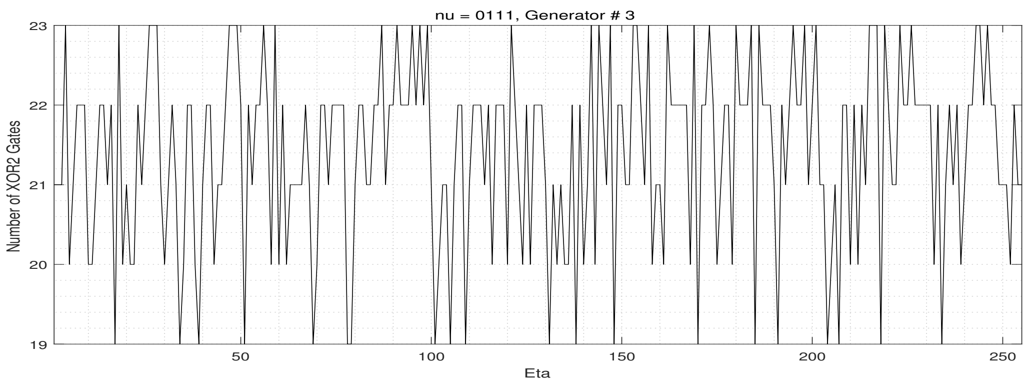


(a)

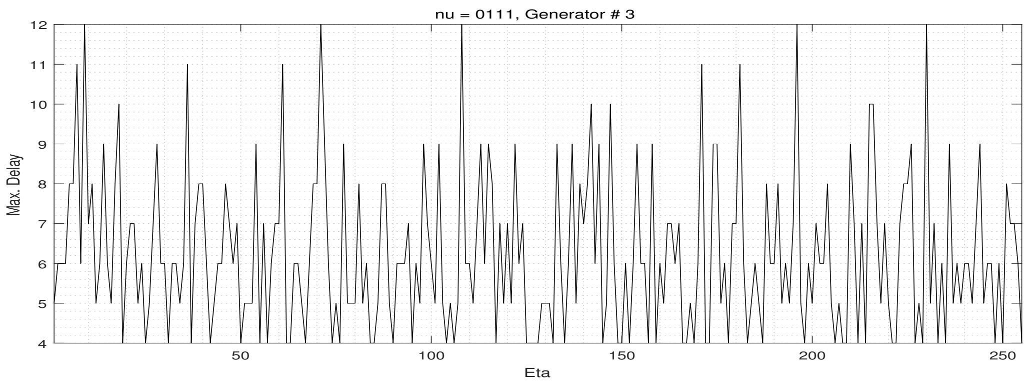


(b)

Figure A.6: (a) The number of XOR2 gates required to implement $\mathbf{T}_{in}$ as a function of η for the case of $\nu = 0111$, Generator #2.(b) The maximum delay of $\mathbf{T}_{in}$ as a function of η for the case of $\nu = 0111$, Generator #2.

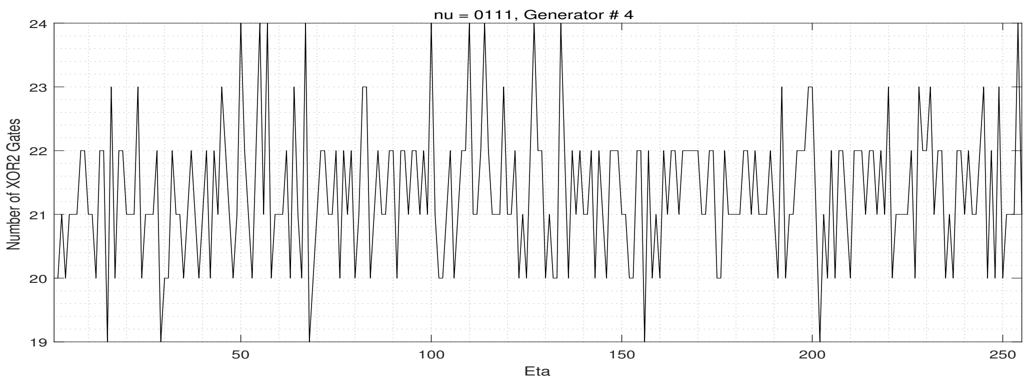


(a)

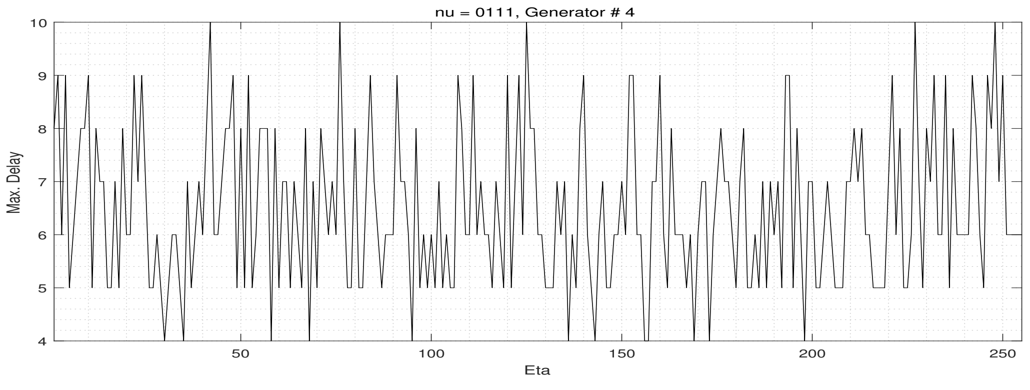


(b)

Figure A.7: (a) The number of XOR2 gates required to implement $\mathbf{T}_{in}$ as a function of η for the case of $\nu = 0111$, Generator #3.(b) The maximum delay of $\mathbf{T}_{in}$ as a function of η for the case of $\nu = 0111$, Generator #3.

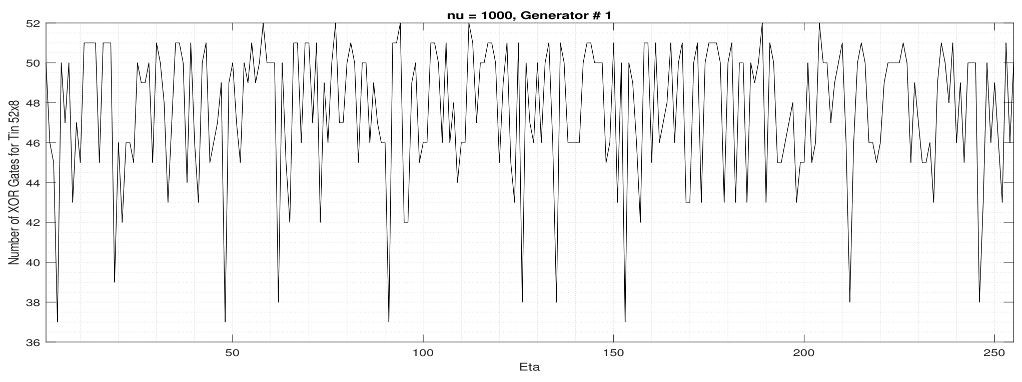


(a)

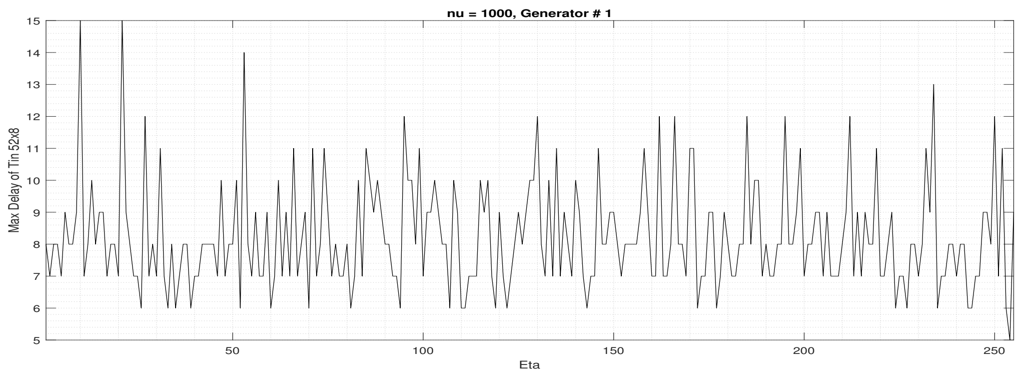


(b)

Figure A.8: (a) The number of XOR2 gates required to implement $\mathbf{T}_{in}$ as a function of η for the case of $\nu = 0111$, Generator #4.(b) The maximum delay of $\mathbf{T}_{in}$ as a function of η for the case of $\nu = 0111$, Generator #4.

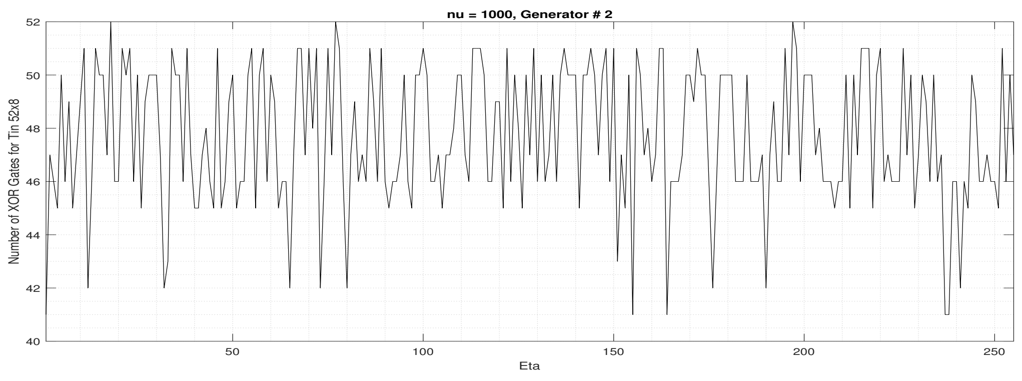


(a)

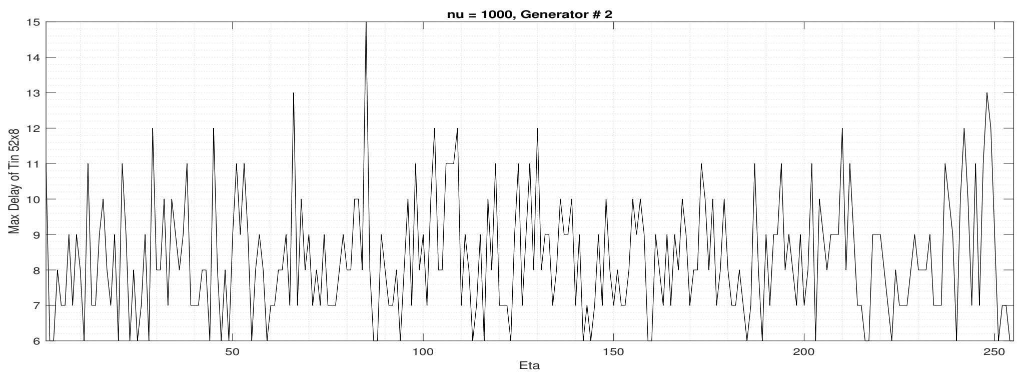


(b)

Figure A.9: (a) The number of XOR2 gates required to implement $\mathbf{T}'_{in}$ as a function of η for the case of $\nu = 1000$, Generator #1.(b) The maximum delay of $\mathbf{T}'_{in}$ as a function of η for the case of $\nu = 1000$, Generator #1.



(a)



(b)

Figure A.10: (a) The number of XOR2 gates required to implement $\mathbf{T}'_{in}$ as a function of η for the case of $\nu = 1000$, Generator #2.(b) The maximum delay of $\mathbf{T}'_{in}$ as a function of η for the case of $\nu = 1000$, Generator #2.

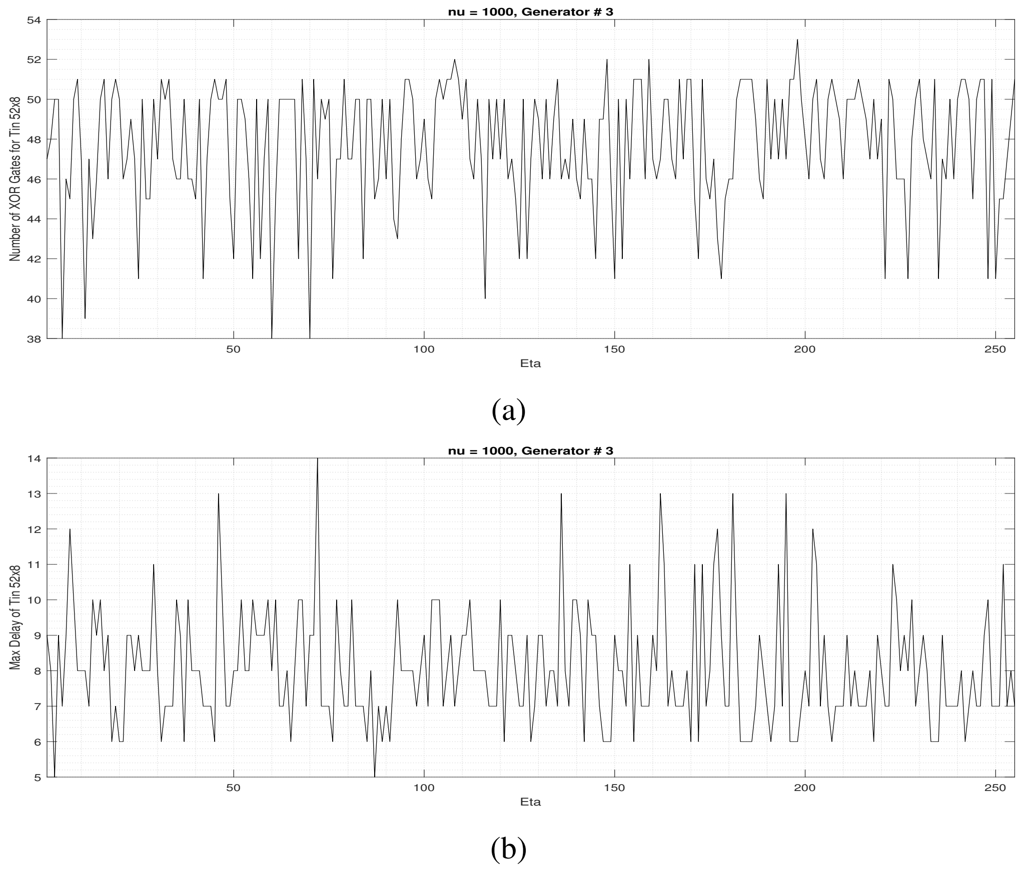


Figure A.11: (a) The number of XOR2 gates required to implement $\mathbf{T}'_{in}$ as a function of η for the case of $\nu = 1000$, Generator #3.(b) The maximum delay of $\mathbf{T}'_{in}$ as a function of η for the case of $\nu = 1000$, Generator #3.

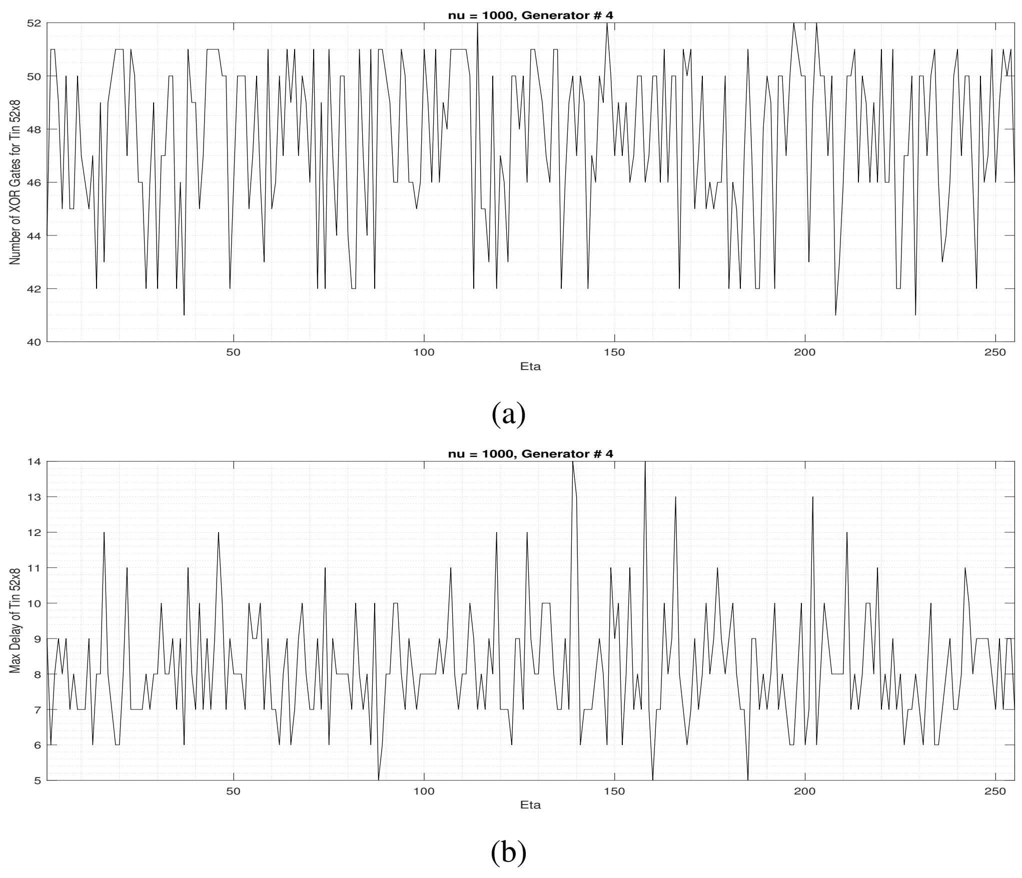
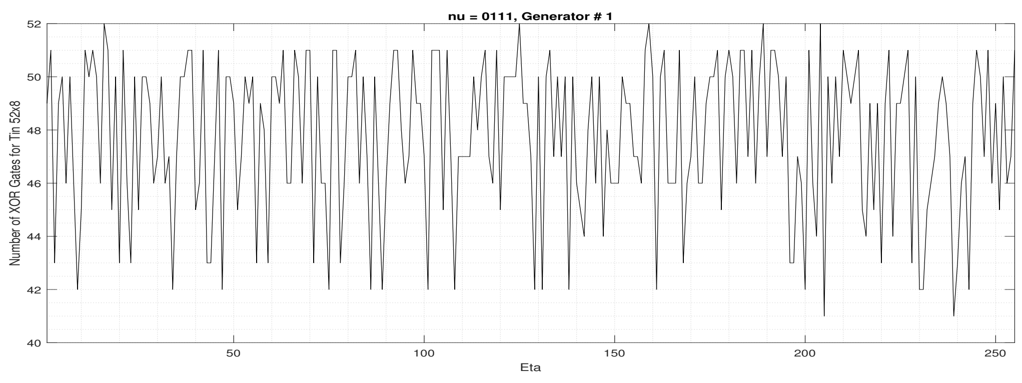
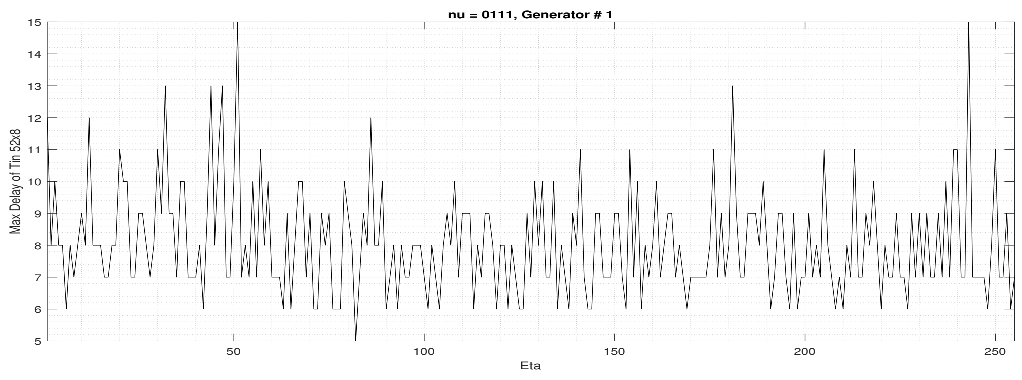


Figure A.12: (a) The number of XOR2 gates required to implement $\mathbf{T}'_{in}$ as a function of η for the case of $\nu = 1000$, Generator #4.(b) The maximum delay of $\mathbf{T}'_{in}$ as a function of η for the case of $\nu = 1000$, Generator #4.

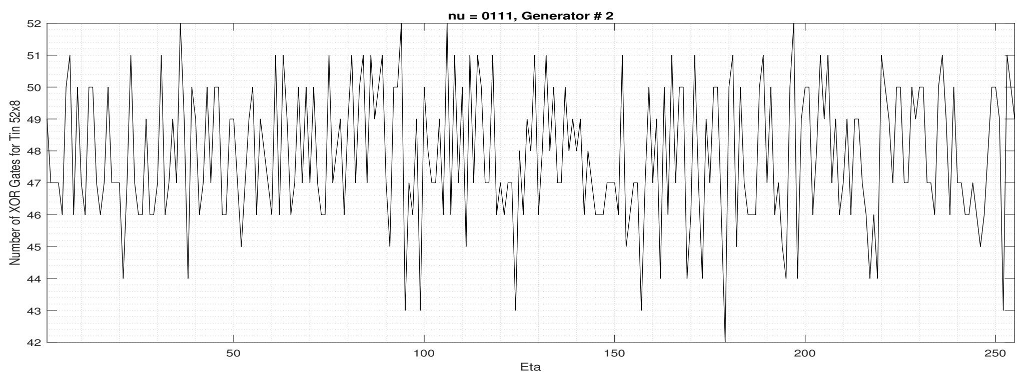


(a)

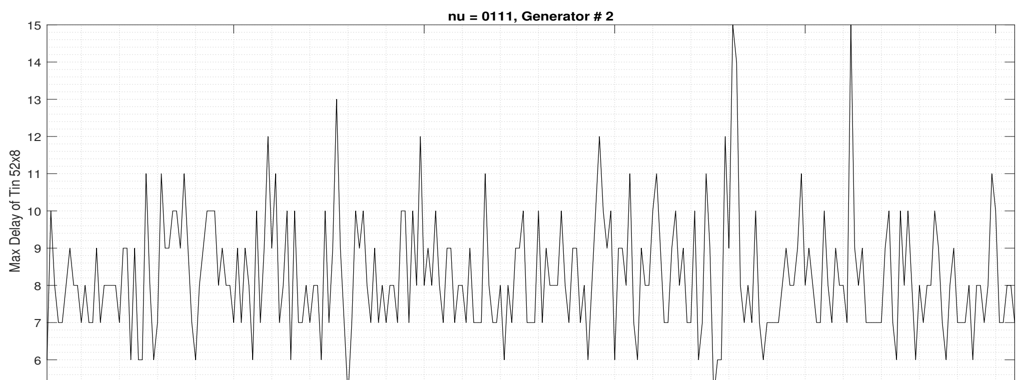


(b)

Figure A.13: (a) The number of XOR2 gates required to implement $\mathbf{T}'_{in}$ as a function of η for the case of $\nu = 0111$, Generator #1.(b) The maximum delay of $\mathbf{T}'_{in}$ as a function of η for the case of $\nu = 0111$, Generator #1.

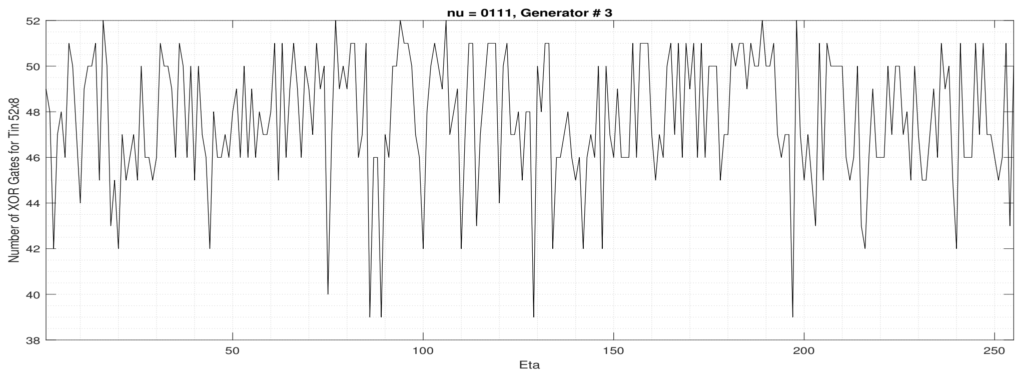


(a)

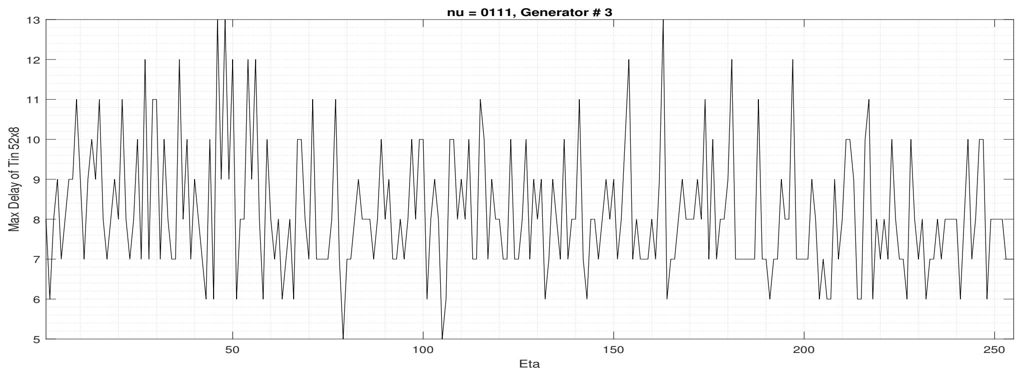


(b)

Figure A.14: (a) The number of XOR2 gates required to implement $\mathbf{T}'_{in}$ as a function of η for the case of $\nu = 0111$, Generator #2.(b) The maximum delay of $\mathbf{T}'_{in}$ as a function of η for the case of $\nu = 0111$, Generator #2.

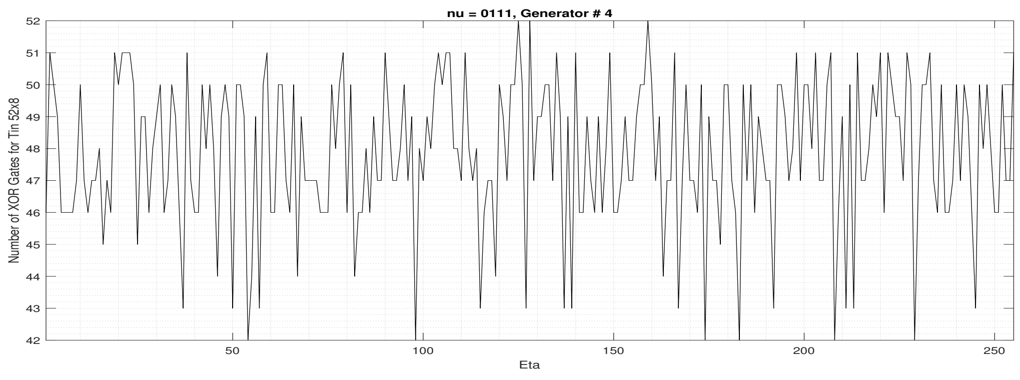


(a)

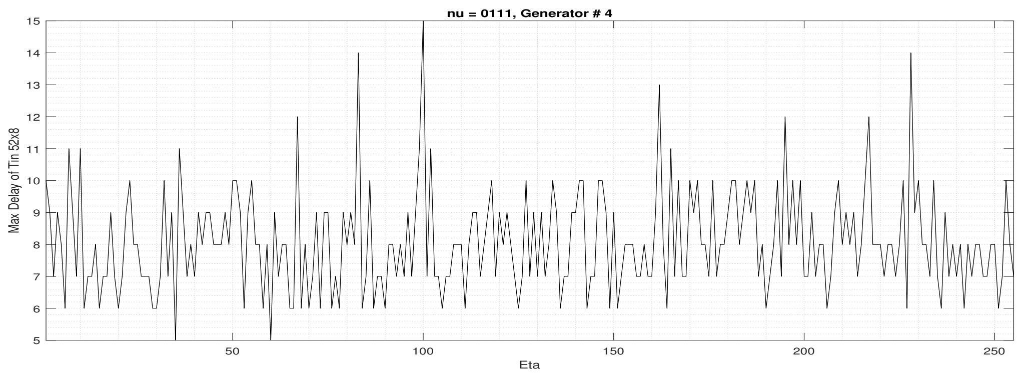


(b)

Figure A.15: (a) The number of XOR2 gates required to implement $\mathbf{T}'_{in}$ as a function of η for the case of $\nu = 0111$, Generator #3.(b) The maximum delay of $\mathbf{T}'_{in}$ as a function of η for the case of $\nu = 0111$, Generator #3.



(a)

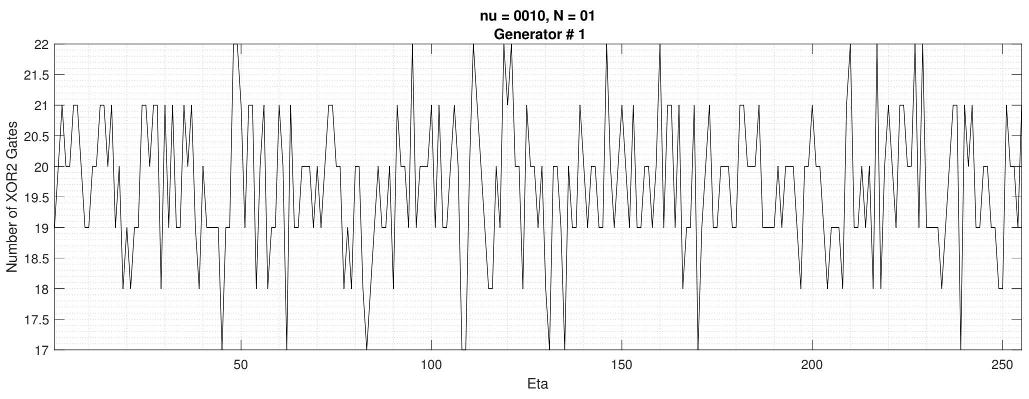


(b)

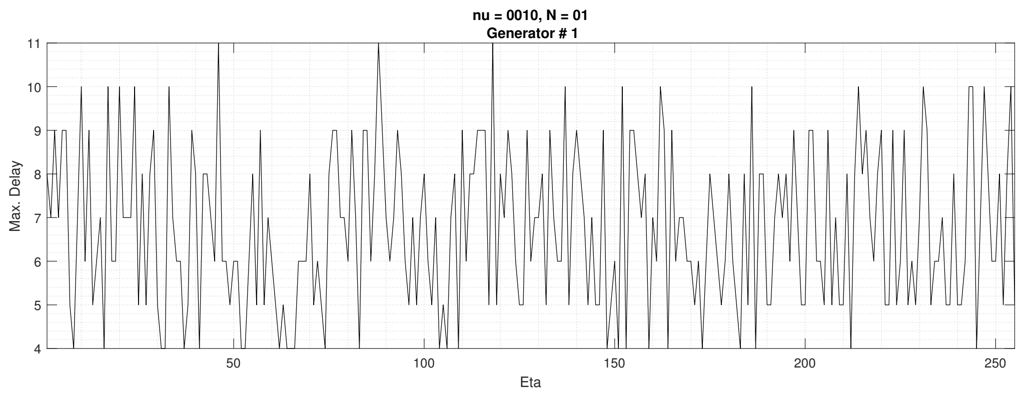
Figure A.16: (a) The number of XOR2 gates required to implement $\mathbf{T}'_{in}$ as a function of η for the case of $\nu = 0111$, Generator #4.(b) The maximum delay of $\mathbf{T}'_{in}$ as a function of η for the case of $\nu = 0111$, Generator #4.

Appendix B

Additional Figures for Chapter 4

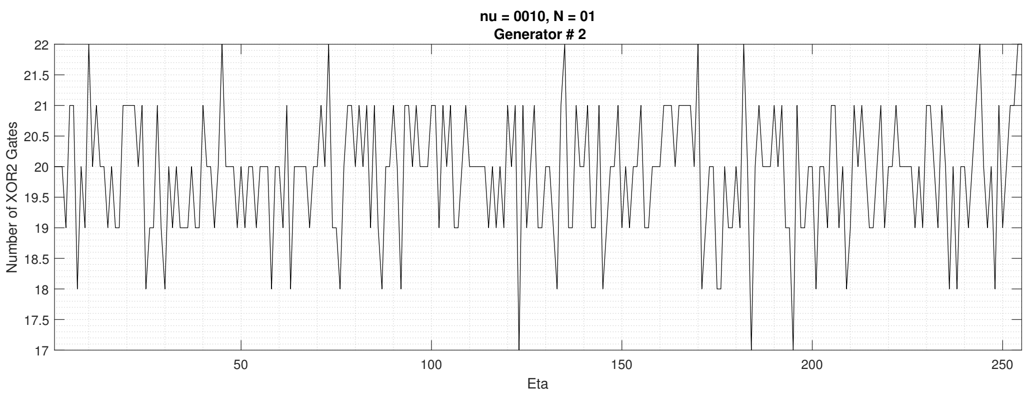


(a)

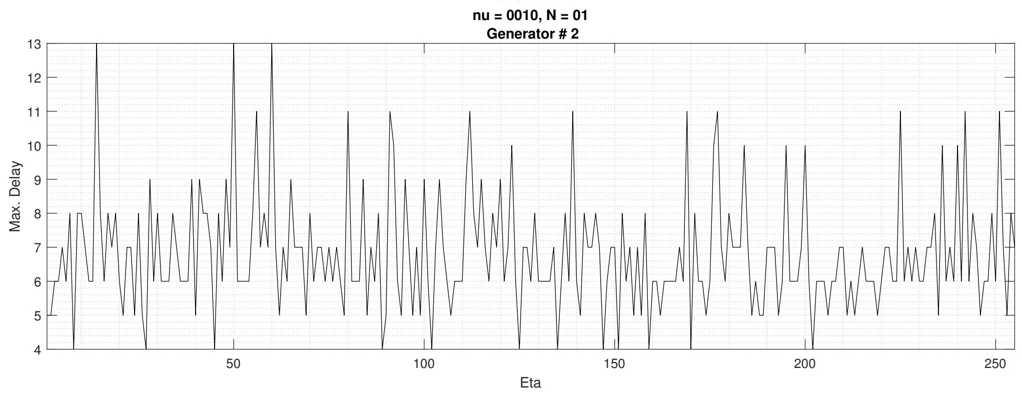


(b)

Figure B.1: (a) The number of XOR2 gates required to implement $\mathbf{T}_{in_{new}}$ as a function of η for the case of $\{\nu = [0010], N = [01]\}$, Generator #1.(b) The maximum delay of $\mathbf{T}_{in_{new}}$ as a function of η for the case of $\{\nu = [0010], N = [01]\}$, Generator #1.

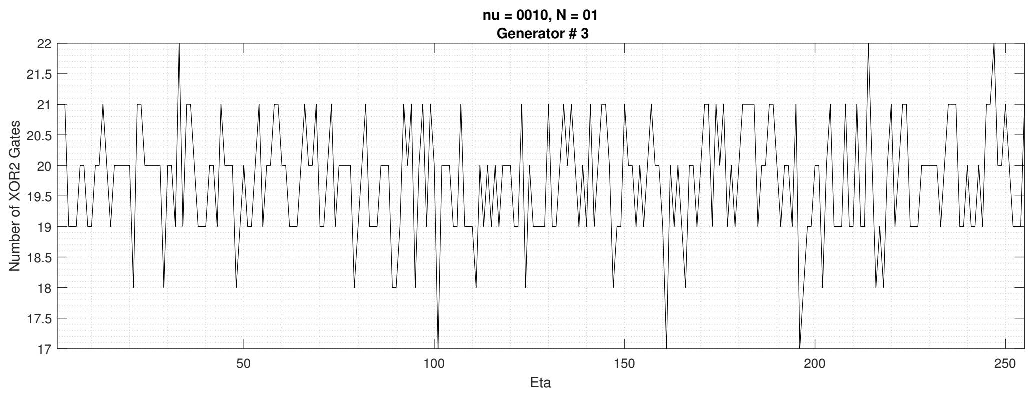


(a)

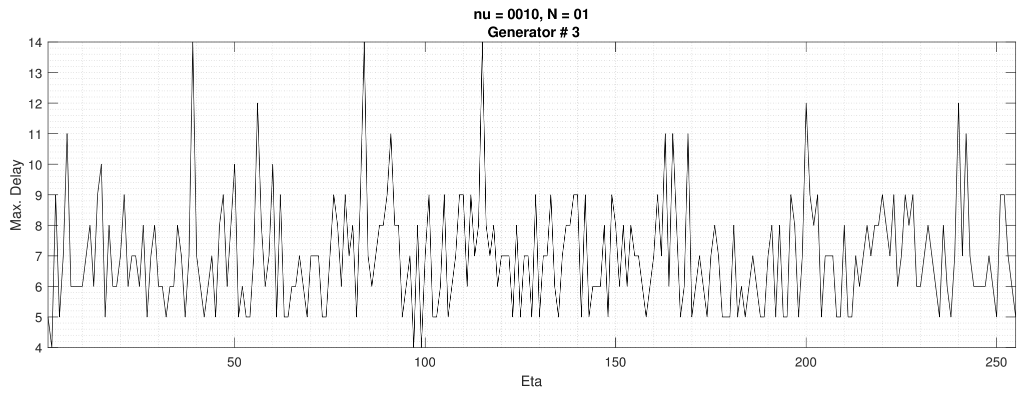


(b)

Figure B.2: (a)The number of XOR2 gates required to implement $\mathbf{T}_{in_{new}}$ as a function of η for the case of $\{\nu = [0010], N = [01]\}$, Generator #2.(b) The maximum delay of $\mathbf{T}_{in_{new}}$ as a function of η for the case of $\{\nu = [0010], N = [01]\}$, Generator #2.

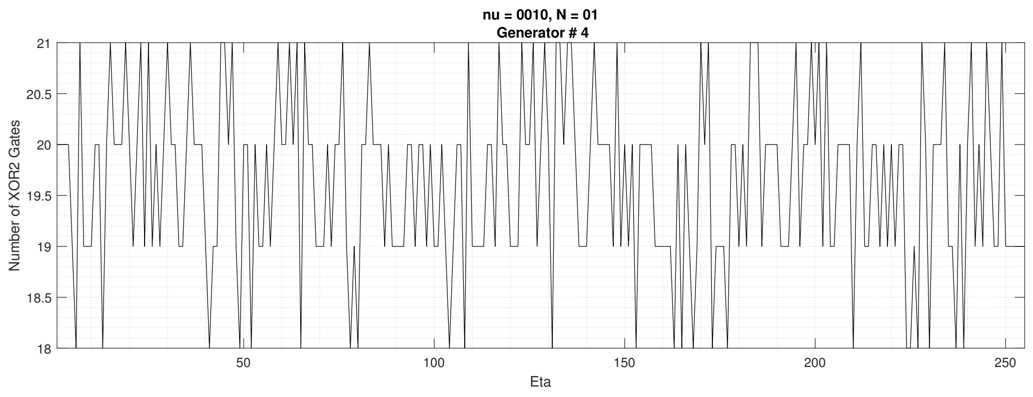


(a)

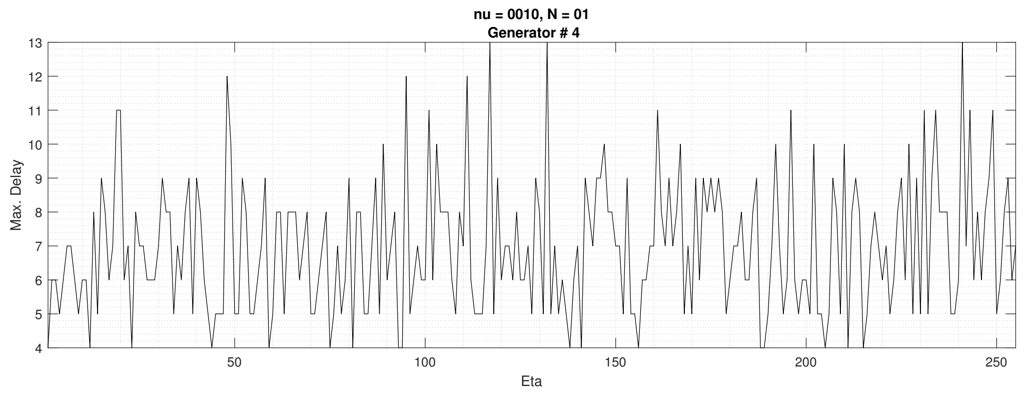


(b)

Figure B.3: (a) The number of XOR2 gates required to implement $\mathbf{T}_{in_{new}}$ as a function of η for the case of $\{\nu = [0010], N = [01]\}$, Generator #3.(b) The maximum delay of $\mathbf{T}_{in_{new}}$ as a function of η for the case of $\{\nu = [0010], N = [01]\}$, Generator #3.

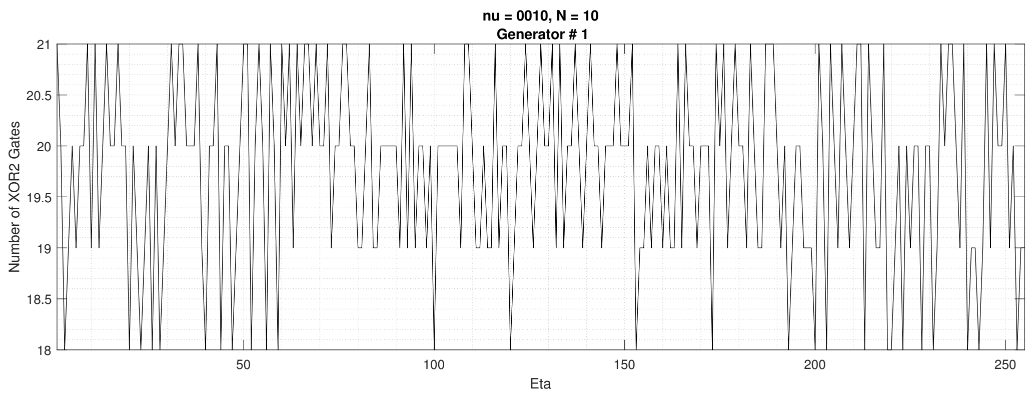


(a)

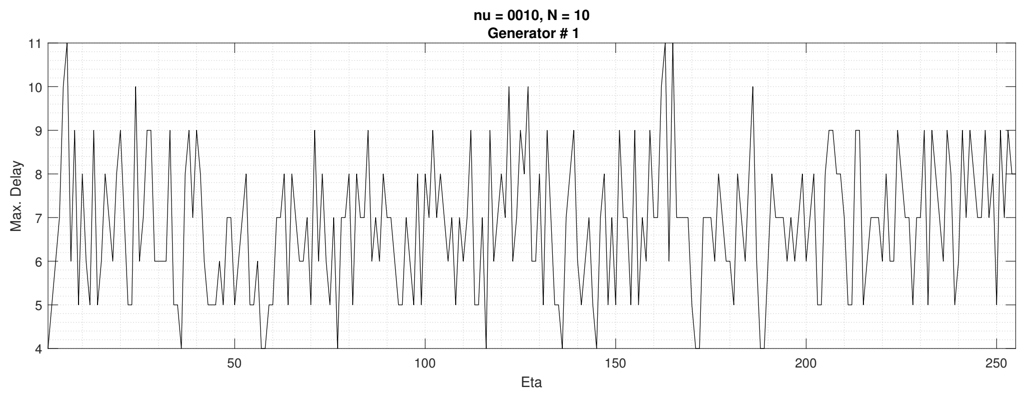


(b)

Figure B.4: (a) The number of XOR2 gates required to implement $\mathbf{T}_{in_{new}}$ as a function of η for the case of $\{\nu = [0010], N = [01]\}$, Generator #4.(b) The maximum delay of $\mathbf{T}_{in_{new}}$ as a function of η for the case of $\{\nu = [0010], N = [01]\}$, Generator #4.

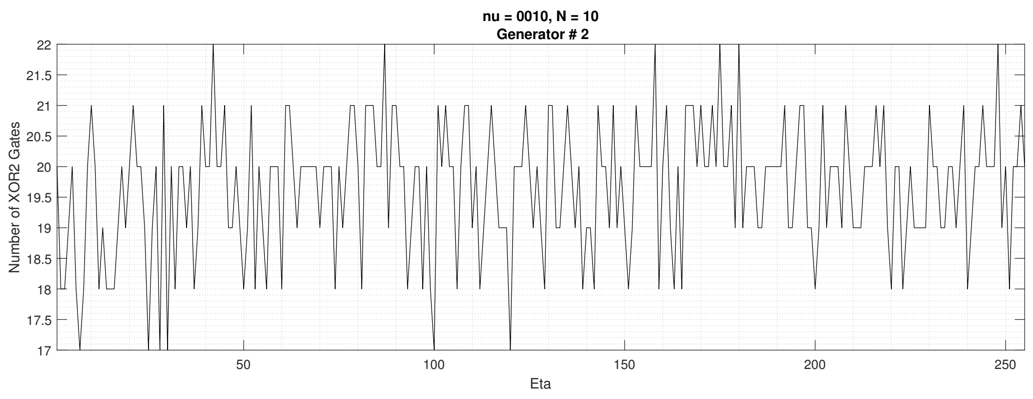


(a)

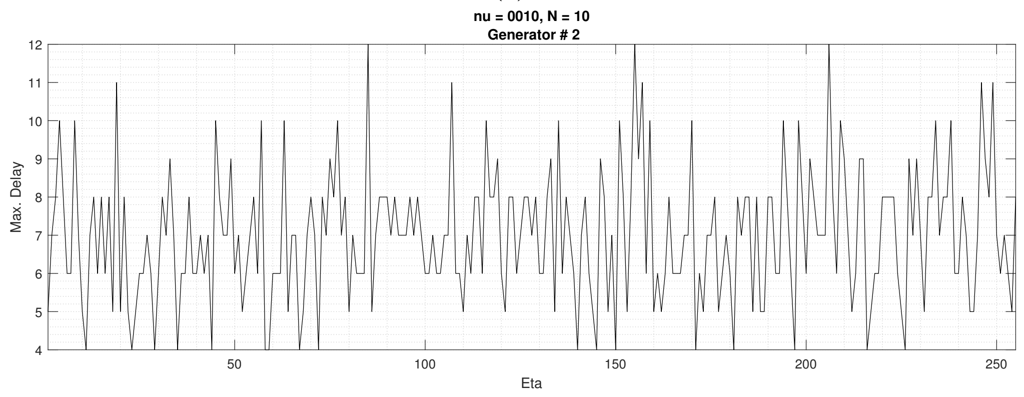


(b)

Figure B.5: (a) The number of XOR2 gates required to implement $\mathbf{T}_{in_{new}}$ as a function of η for the case of $\{\nu = [0010], N = [10]\}$, Generator #1.(b) The maximum delay of $\mathbf{T}_{in_{new}}$ as a function of η for the case of $\{\nu = [0010], N = [10]\}$, Generator #1.

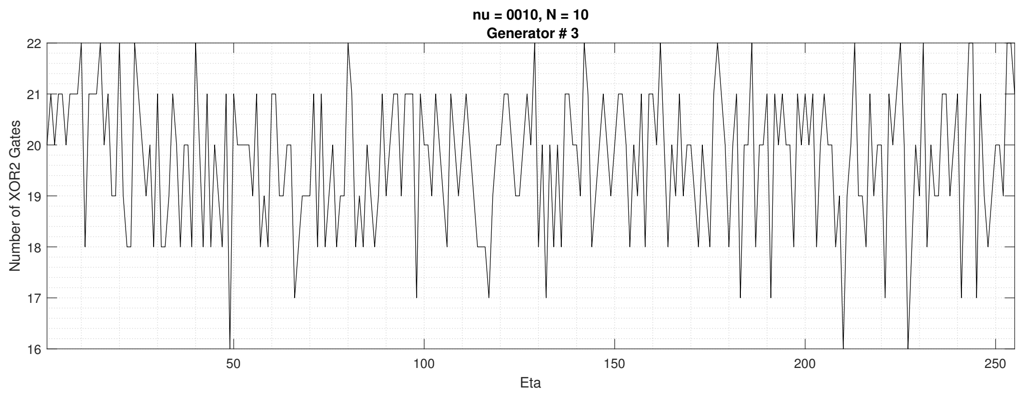


(a)

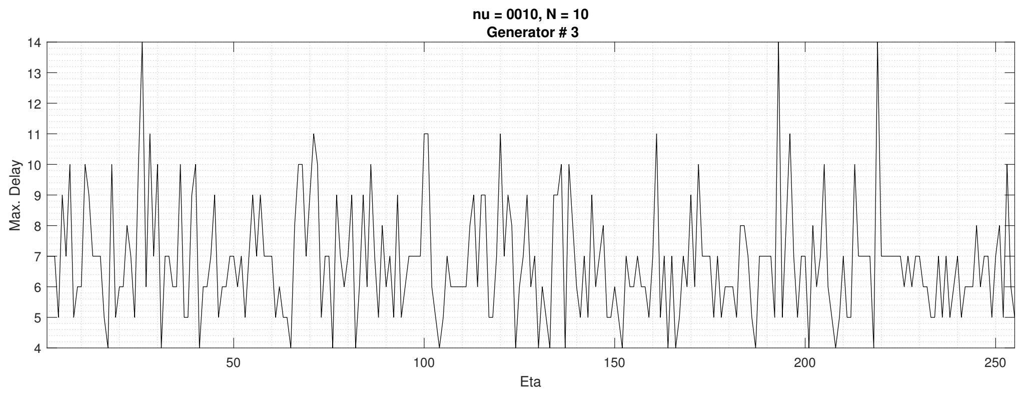


(b)

Figure B.6: (a) The number of XOR2 gates required to implement $\mathbf{T}_{in_{new}}$ as a function of η for the case of $\{\nu = [0010], N = [10]\}$, Generator #2.(b) The maximum delay of $\mathbf{T}_{in_{new}}$ as a function of η for the case of $\{\nu = [0010], N = [10]\}$, Generator #2.

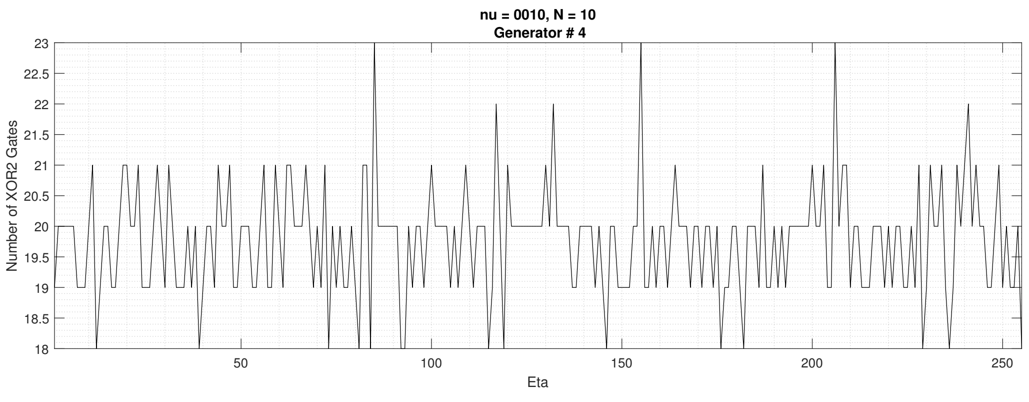


(a)

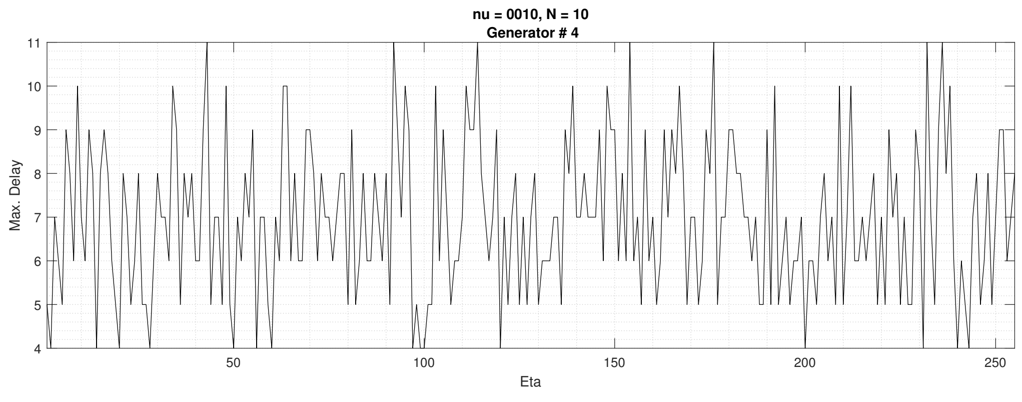


(b)

Figure B.7: (a) The number of XOR2 gates required to implement $\mathbf{T}_{in_{new}}$ as a function of η for the case of $\{\nu = [0010], N = [10]\}$, Generator #3.(b) The maximum delay of $\mathbf{T}_{in_{new}}$ as a function of η for the case of $\{\nu = [0010], N = [10]\}$, Generator #3.

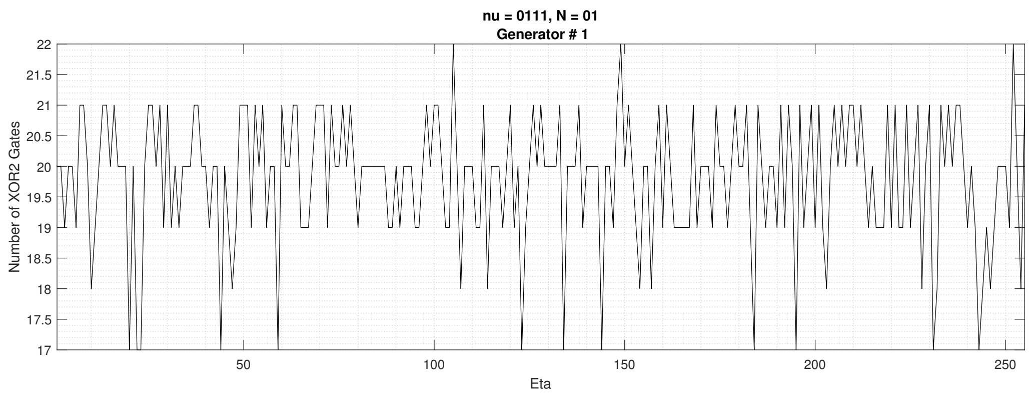


(a)

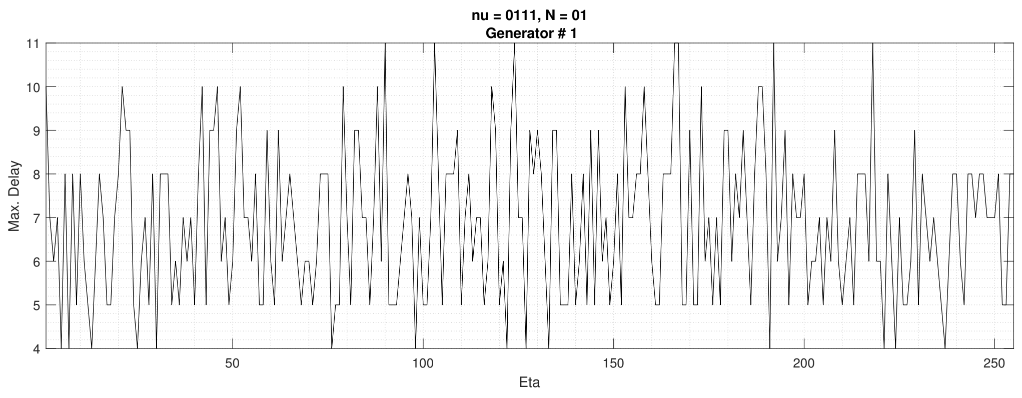


(b)

Figure B.8: (a) The number of XOR2 gates required to implement $\mathbf{T}_{in_{new}}$ as a function of η for the case of $\{\nu = [0010], N = [10]\}$, Generator #4.(b) The maximum delay of $\mathbf{T}_{in_{new}}$ as a function of η for the case of $\{\nu = [0010], N = [10]\}$, Generator #4.

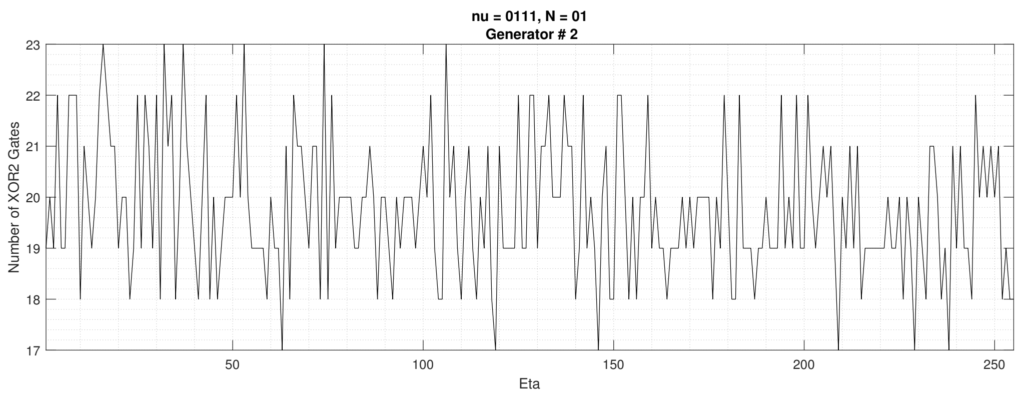


(a)

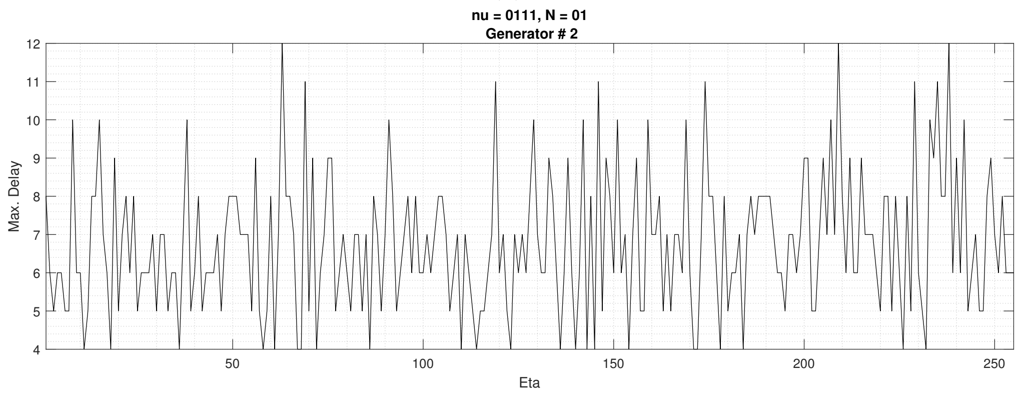


(b)

Figure B.9: (a) The number of XOR2 gates required to implement $\mathbf{T}_{in_{new}}$ as a function of η for the case of $\{\nu = [0111], N = [01]\}$, Generator #1.(b) The maximum delay of $\mathbf{T}_{in_{new}}$ as a function of η for the case of $\{\nu = [0111], N = [01]\}$, Generator #1.

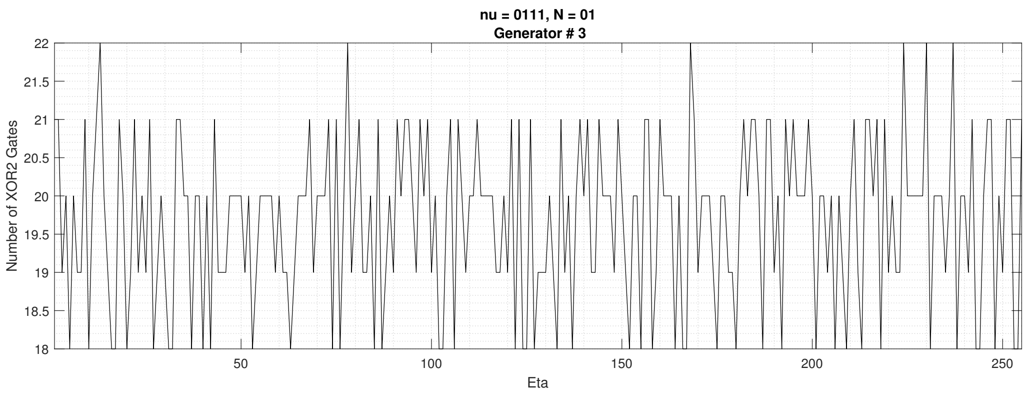


(a)

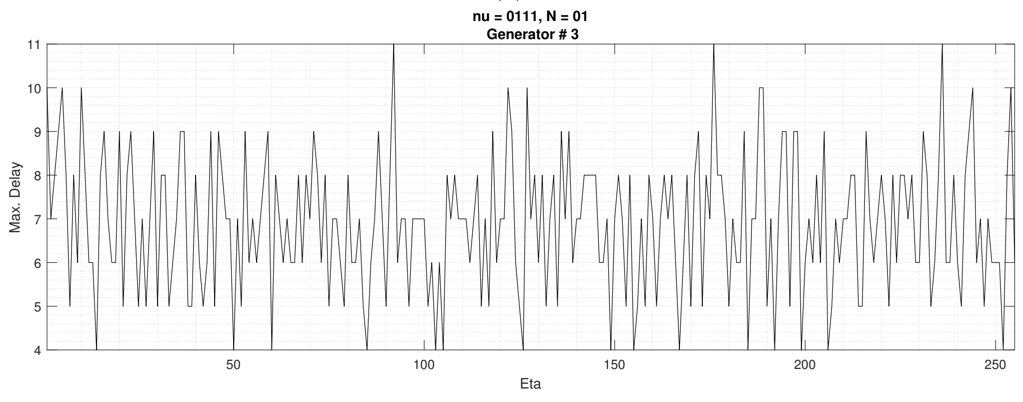


(b)

Figure B.10: (a) The number of XOR2 gates required to implement $\mathbf{T}_{in_{new}}$ as a function of η for the case of $\{\nu = [0111], N = [01]\}$, Generator #2.(b) The maximum delay of $\mathbf{T}_{in_{new}}$ as a function of η for the case of $\{\nu = [0111], N = [01]\}$, Generator #2.

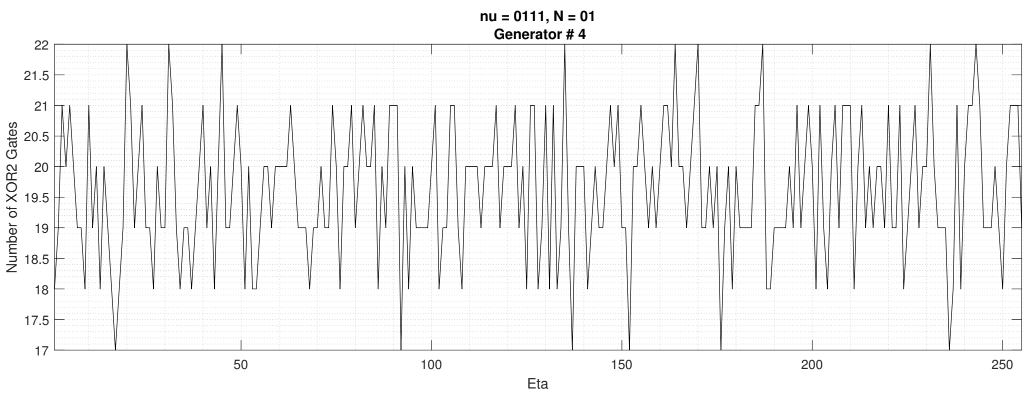


(a)

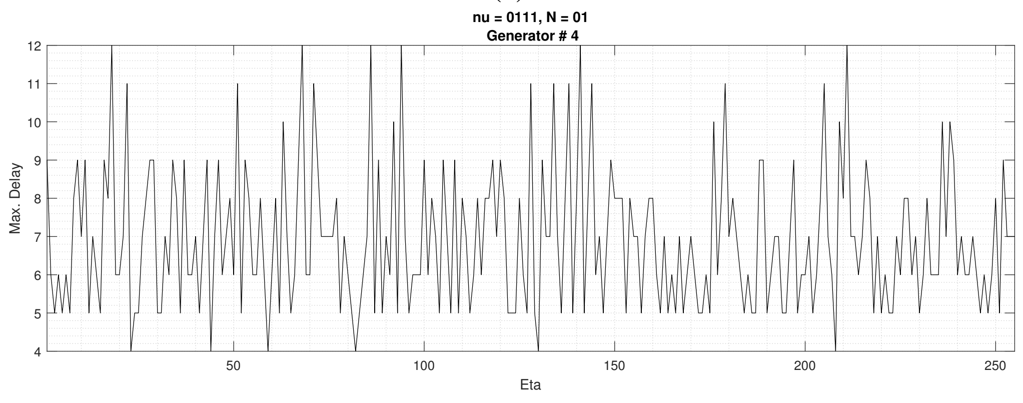


(b)

Figure B.11: (a) The number of XOR2 gates required to implement $\mathbf{T}_{in_{new}}$ as a function of η for the case of $\{\nu = [0111], N = [01]\}$, Generator #3.(b) The maximum delay of $\mathbf{T}_{in_{new}}$ as a function of η for the case of $\{\nu = [0111], N = [01]\}$, Generator #3.

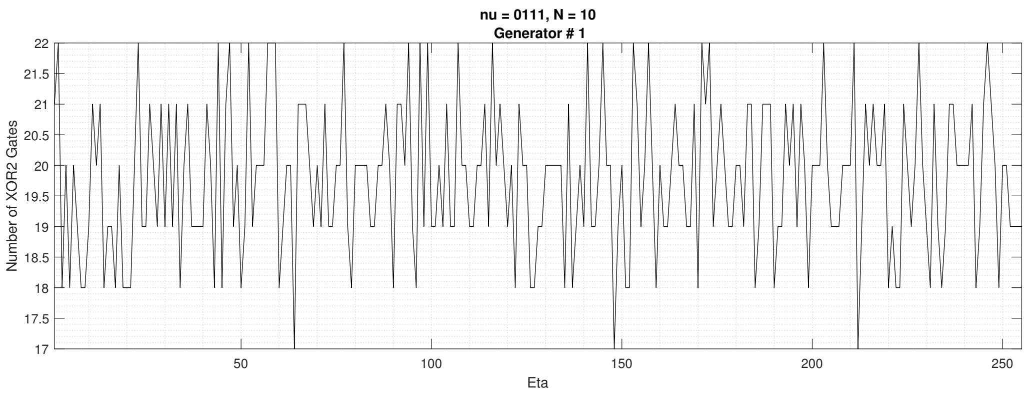


(a)

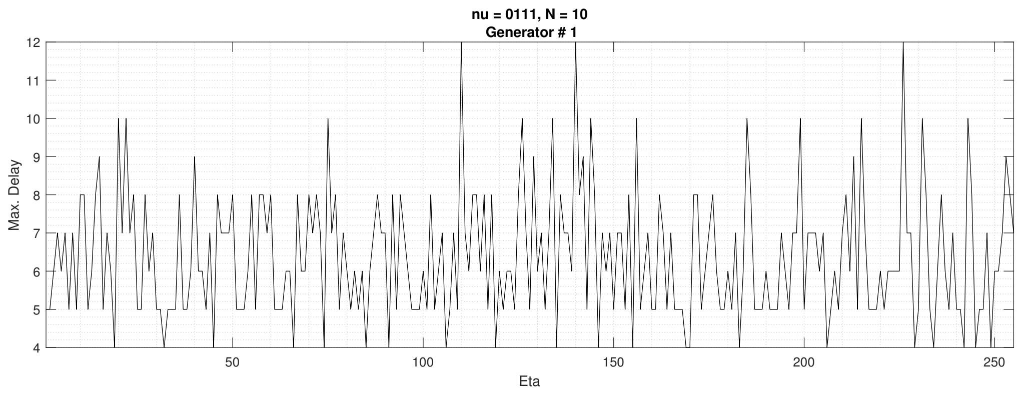


(b)

Figure B.12: (a) The number of XOR2 gates required to implement $\mathbf{T}_{in_{new}}$ as a function of η for the case of $\{\nu = [0111], N = [01]\}$, Generator #4.(b) The maximum delay of $\mathbf{T}_{in_{new}}$ as a function of η for the case of $\{\nu = [0111], N = [01]\}$, Generator #4.

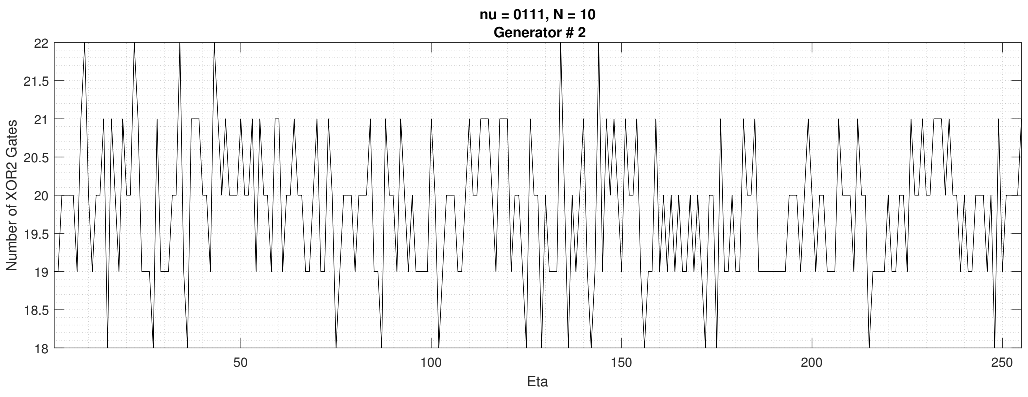


(a)

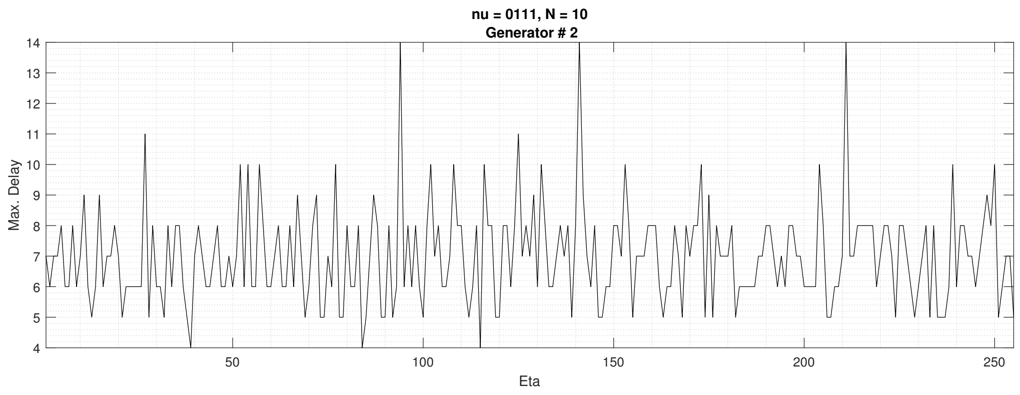


(b)

Figure B.13: (a) The number of XOR2 gates required to implement $\mathbf{T}_{in_{new}}$ as a function of η for the case of $\{\nu = [0111], N = [10]\}$, Generator #1.(b) The maximum delay of $\mathbf{T}_{in_{new}}$ as a function of η for the case of $\{\nu = [0111], N = [10]\}$, Generator #1.

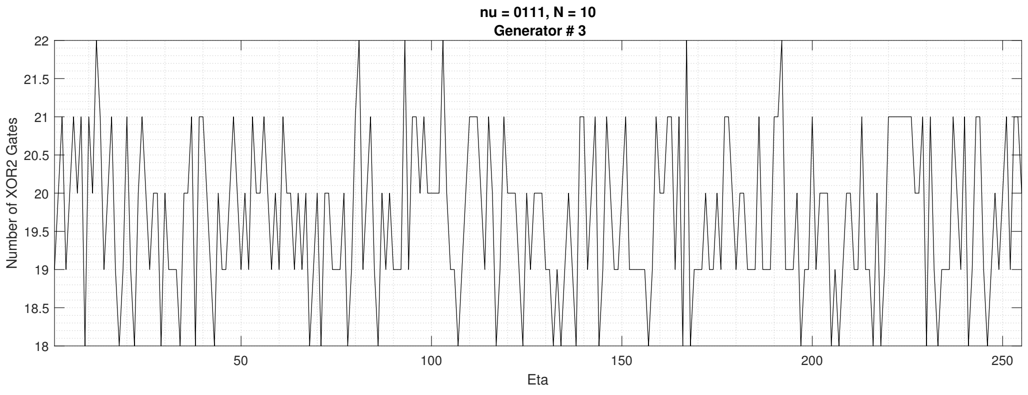


(a)

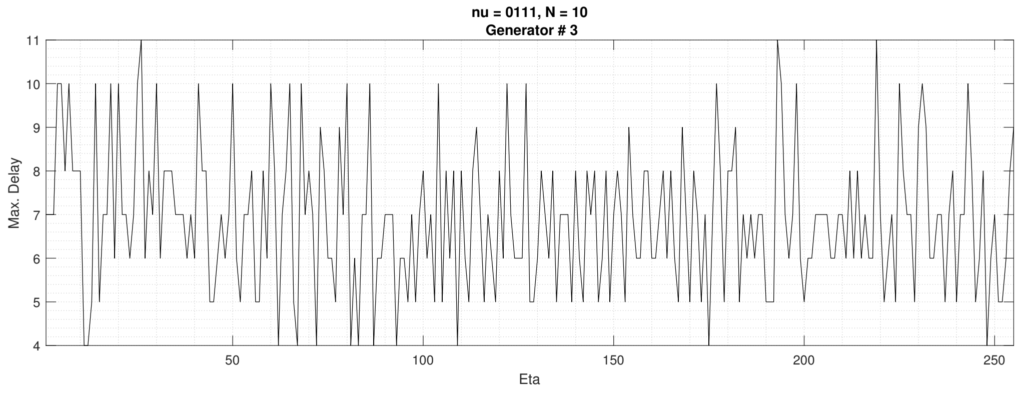


(b)

Figure B.14: (a) The number of XOR2 gates required to implement $\mathbf{T}_{in_{new}}$ as a function of η for the case of $\{\nu = [0111], N = [10]\}$, Generator #2.(b) The maximum delay of $\mathbf{T}_{in_{new}}$ as a function of η for the case of $\{\nu = [0111], N = [10]\}$, Generator #2.

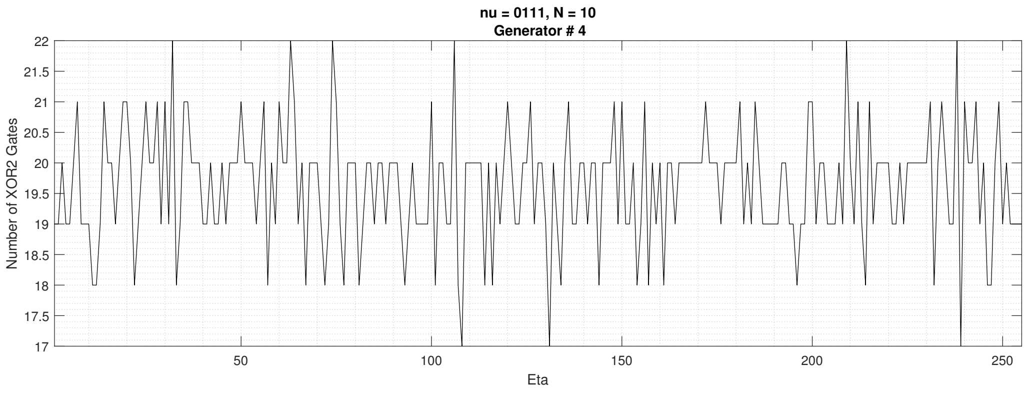


(a)

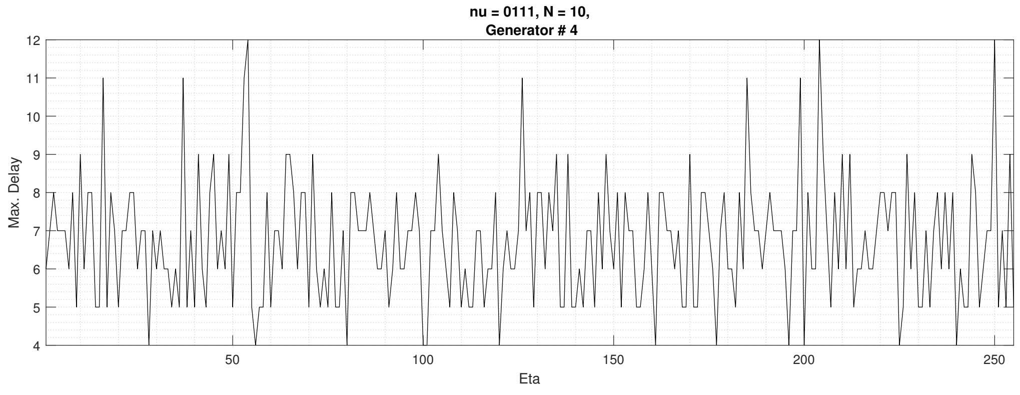


(b)

Figure B.15: (a) The number of XOR2 gates required to implement $\mathbf{T}_{in_{new}}$ as a function of η for the case of $\{\nu = [0111], N = [10]\}$, Generator #3.(b) The maximum delay of $\mathbf{T}_{in_{new}}$ as a function of η for the case of $\{\nu = [0111], N = [10]\}$, Generator #3.



(a)



(b)

Figure B.16: (a) The number of XOR2 gates required to implement $\mathbf{T}_{in_{new}}$ as a function of η for the case of $\{\nu = [0111], N = [10]\}$, Generator #4.(b) The maximum delay of $\mathbf{T}_{in_{new}}$ as a function of η for the case of $\{\nu = [0111], N = [10]\}$, Generator #4.

Curriculum Vitae

Name:	Doaa Ashmawy
Post-Secondary Education and Degrees:	University of Western Ontario London, ON, Canada 2016 Master of Engineering in ECE
Related Work Experience:	Graduate Teaching and Research Assistant University of Western Ontario 2016-2019
	Instructor Benha University 2010-2013

Publications:

[1] Arash Reyhani-Masoleh, Mostafa Taha, and Doaa Ashmawy. Smashing the implementation records of AES S-box. IACR Transactions on Cryptographic Hardware and Embedded Systems, pages 298–336, 2018.

[2] Arash Reyhani-Masoleh, Mostafa Taha, and Doaa Ashmawy. New area record for the AES combined S-box / inverse S-box. In 2018 IEEE 25th Symposium on Computer Arithmetic (ARITH), pages 145–152. IEEE, 2018.

[3] Arash Reyhani-Masoleh, Mostafa Taha, and Doaa Ashmawy. New low-area designs for the AES forward, inverse and combined -boxes. IEEE Transactions on Computers, 2019.