Western Graduate&PostdoctoralStudies

Western University
Scholarship@Western

Electronic Thesis and Dissertation Repository

6-5-2015 12:00 AM

Hardware Implementations for Symmetric Key Cryptosystems

Hayssam El-Razouk, *The University of Western Ontario*

Supervisor: Arash Reyhani-Masoleh, *The University of Western Ontario*

A thesis submitted in partial fulfillment of the requirements for the Doctor of Philosophy degree in Electrical and Computer Engineering

© Hayssam El-Razouk 2015

Follow this and additional works at: https://ir.lib.uwo.ca/etd

Part of the VLSI and Circuits, Embedded and Hardware Systems Commons

Recommended Citation

El-Razouk, Hayssam, "Hardware Implementations for Symmetric Key Cryptosystems" (2015). *Electronic Thesis and Dissertation Repository*. 2927.
https://ir.lib.uwo.ca/etd/2927

This Dissertation/Thesis is brought to you for free and open access by Scholarship@Western. It has been accepted for inclusion in Electronic Thesis and Dissertation Repository by an authorized administrator of Scholarship@Western. For more information, please contact wlswadmin@uwo.ca.

HARDWARE IMPLEMENTATIONS FOR SYMMETRIC KEY CRYPTOSYSTEMS

(Thesis format: Monograph)

by

Hayssam El-Razouk

Graduate Program in Electrical and Computer Engineering

A thesis submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy

The School of Graduate and Postdoctoral Studies
The University of Western Ontario
London, Ontario, Canada

© Hayssam El-Razouk 2015

Abstract

The utilization of global communications network for supporting new electronic applications is growing. Many applications provided over the global communications network involve exchange of security-sensitive information between different entities. Often, communicating entities are located at different locations around the globe. This demands deployment of certain mechanisms for providing secure communications channels between these entities. For this purpose, cryptographic algorithms are used by many of today's electronic applications to maintain security. Cryptographic algorithms provide set of primitives for achieving different security goals such as: confidentiality, data integrity, authenticity, and non-repudiation. In general, two main categories of cryptographic algorithms can be used to accomplish any of these security goals, namely, asymmetric key algorithms and symmetric key algorithms. The security of asymmetric key algorithms is based on the hardness of the underlying computational problems, which usually require large overhead of space and time complexities. On the other hand, the security of symmetric key algorithms is based on non-linear transformations and permutations, which provide efficient implementations compared to the asymmetric key ones. Therefore, it is common to use asymmetric key algorithms for key exchange, while symmetric key counterparts are deployed in securing the communications sessions. This thesis focuses on finding efficient hardware implementations for symmetric key cryptosystems targeting mobile communications and resource constrained applications.

First, efficient lightweight hardware implementations of two members of the Welch-Gong (WG) family of stream ciphers, the WG$(29, 11)$ and WG-16, are considered for the mobile communications domain. Optimizations in the WG$(29, 11)$ stream cipher are considered when the $GF\left(2^{29}\right)$ elements are represented in either the Optimal normal basis type-II (ONB-II) or the Polynomial basis (PB). For WG-16, optimizations are considered only for PB representations of the $GF\left(2^{16}\right)$ elements. In this regard, optimizations for both ciphers are accomplished mainly at the arithmetic level through reducing the number of field multipliers, based on novel trace properties. In addition, other optimization techniques such as serialization and pipelining, are also considered.

After this, the thesis explores efficient hardware implementations for digit-level multiplication over binary extension fields $GF(2^m)$. Efficient digit-level $GF(2^m)$ multiplications are advantageous for ultra-lightweight implementations, not only in symmetric key algorithms, but also in asymmetric key algorithms. The thesis introduces new architectures for digit-level $GF(2^m)$ multipliers considering the Gaussian normal basis (GNB) and PB representations of the field elements. The new digit-level $GF(2^m)$ single multipliers do not require loading of the two input field elements in advance to computations. This feature results in high throughput fast multiplication in resource constrained applications with limited capacity of input data-paths. The new digit-level $GF(2^m)$ single multipliers are considered for both the GNB and PB. In addition, for the GNB representation, new architectures for digit-level $GF(2^m)$ hybrid-double and hybrid-triple multipliers are introduced. The new digit-level $GF(2^m)$ hybrid-double and hybrid-triple GNB multipliers, respectively, accomplish the multiplication of three and four field elements using the latency required for multiplying two field elements. Furthermore, a new hardware architecture for the eight-ary exponentiation scheme is proposed by utilizing the new digit-level $GF(2^m)$ hybrid-triple GNB multipliers.

Keywords: Digit-Level Multipliers, Finite Fields, Finite Field Exponentiation, Gaussian Normal Basis, Hybrid-Double Multiplication, Linear Feedback Shift Registers, Normal Basis, Optimal Normal Basis, Polynomial Basis, Pseudo Random Key Generators, Serial Multiplication, Stream Ciphers, Trace Function, WG Transformation.

Co-Authorship

I would like to thank Dr. Guang Gong, from the electrical and computer engineering department at the University of Waterloo, for her constructive inputs during the different discussions and throughout the writing / revision phases of the published / accepted versions of chapters 3 and 4 of this thesis.

Dedications

To my great parents, to my lovely wife Shaima, to my little precious girl Quds, and to all my bigger family.

Acknowledgements

First of all, all praise and thanks are due to my Lord.

I would like to thank my parents, my wife Shaima, my beautiful daughter Quds, and my sisters for all their support, and prayers.

I would like to thank my supervisor, Dr. Arash Reyhani-Masoleh, for his continuous and non-stopping support and guidance throughout the course of my PhD studies.

I would also like to thank the examiners, Dr. Anestis Dounavis, Dr. Abdelkader Ouda, Dr. Marc Moreno Maza, and Dr. Majid Ahmadi, for putting the time and effort to read my PhD thesis and provide their constructive comments.

Last but not least, I would like to thank my colleges, Behdad Husseini, Depanwita Gangopadhyay, Ebrahim Hassan, and Sasan Khoshroo, for all the constructive and fruitful discussions we had.

Contents

Abstract **ii**

Co-Authorship **iv**

Dedications **v**

Acknowledgements **vi**

List of Figures **xii**

List of Tables **xiv**

Nomenclature **xvi**

1 Introduction **1**

1.1 Objectives 2

1.2 Motivations and Significance 2

1.3 Thesis Outline 4

2 Background on $GF(2^m)$ and WG Stream Ciphers **5**

2.1 Preliminaries on Algebraic Structures 5

2.2 Finite Fields 7

2.2.1 Fields 7

2.2.2 Finite Fields 7

2.3 Modular Arithmetic 7

2.4 Polynomial Rings 9

2.5 Binary Extension Fields $GF(2^m)$ 11

2.6 Polynomial Basis (PB) Representation 11

2.7 Gaussian Normal Basis (GNB) Representation 11

2.8 Addition over $GF(2^m)$ 12

2.9 Multiplication over $GF(2^m)$ 12

2.9.1 $GF(2^m)$ Multiplication in the PB Representation 13

2.9.1.1 Multiplication of Two Arbitrary $GF(2^m)$ Elements Represented in the PB 13

2.9.1.2 Efficient Multiplication of a $GF(2^m)$ Element Represented in the PB by α^q 15

2.9.1.3 Previous Work on PB Multiplication 15

2.9.2 $GF(2^m)$ Multiplication in the GNB Representation 17

2.9.2.1 Formulation for the BL-PISO $GF(2^m)$ Multiplication in the GNB Representation 17

2.9.2.2 Multiplication by the Normal Element β 18

2.9.2.3 Previous Work on GNB Multiplication 18

2.10 Exponentiation and Inverse over $GF(2^m)$ 19

2.11 Trace Mapping 20

2.12 Welch-Gong (WG) Stream Ciphers 21

2.12.1 Stream Ciphers 21

2.12.2 WG Stream Ciphers 22

2.12.2.1 A General Block Diagram 22

2.12.2.2 Phases of Operation 23

2.12.2.3 WG(29, 11) and WG-16 23

2.12.2.4 Parameters of the WG(29, 11) 24

2.12.2.5 Parameters of the WG-16 25

3 Implementations of the WG Stream Ciphers Using ONB-II 26

3.1 Optimized Hardware Design of the MOWG(29, 11, 17) Cipher 27

3.1.1 Reducing the Hardware Complexity of the MOWG Transformation . . 27

3.1.2 Improving the Critical Path of the MOWG Transform 28

3.1.2.1 Formulation 29

3.1.2.2 Modified KIA Algorithm 30

3.1.3 Architecture 30

3.1.4 The Finite State Machine 32

3.1.5 Space and Time Complexities 33

3.1.5.1 Space Complexity 33

3.1.5.2 Time Complexity 35

3.2 Low Complexity WG Cipher 36

3.2.1 Properties of the Trace Function for Type-II ONB 36

3.2.2 Optimizing the WG Transform's Hardware for the Run Phase 38

3.2.3 Serializing the Computation of the Initial Feedback Signal 39
3.2.3.1 Architecture and Operation of the Modified FSM 40
3.2.3.2 Architecture and Operation of the Serialized Key Initialization Module 42
3.2.4 Space and Time Complexities 44
3.2.4.1 Space Complexity 44
3.2.4.2 Time Complexity 46
3.3 Results and Comparisons 46
3.3.1 Results from FPGA and ASIC Implementations 46
3.3.2 Discussion 49
3.4 Conclusion 51

4 Implementations of the WG Stream Ciphers Using PB 52
4.1 Architectures of the WG(29, 11) Stream Cipher 54
4.1.1 Formulation of $WGT29$ 54
4.1.2 Design Parameters 54
4.1.2.1 Field Polynomial and Squaring Matrices 55
4.1.2.2 Characteristic Polynomial of the LFSR 55
4.1.2.3 Trace Vector 57
4.1.2.4 Trace of Multiplication of Two Field Elements 57
4.1.3 Architecture and FSM 59
4.1.3.1 Architecture of the WG(29, 11) Cipher 59
4.1.3.2 The Finite State Machine (FSM) 62
4.1.4 Serialized Implementation of the PB Based WG(29, 11) 62
4.1.4.1 Architecture of the Serialized WG(29, 11) 62
4.1.4.2 FSM for the Serialized PB based WG(29, 11) 64
4.1.5 Pipelined Implementation of the PB Based WG(29, 11) 66
4.1.5.1 Architecture of the Pipelined PB Based WG(29, 11) 66
4.1.5.2 FSM for the Pipelined PB Based WG(29, 11) 67
4.2 Architectures of the WG-16 Stream Cipher 68
4.2.1 Formulations of $WGP16$ and $WGT16$ 69
4.2.2 Squaring Matrices and Trace Vector 70
4.2.2.1 Squaring Matrices 70
4.2.2.2 Trace Vector 71
4.2.3 Trace of the Multiplication of Two Field Elements for the PB Based WG-16 71

4.2.4 Architecture and FSM 71
4.2.4.1 Architecture of the WG-16 Cipher 71
4.2.4.2 The Finite State Machine 74
4.2.5 Serialized Implementation of the PB Based WG-16 74
4.2.5.1 Architecture of the Serialized WG-16 74
4.2.5.2 FSM for the Serialized WG-16 75
4.2.6 Pipelined Implementation of the PB Based WG-16 77
4.2.6.1 Architecture 77
4.2.6.2 FSM for the Pipelined WG-16 78
4.3 Implementation Results and Comparisons 78
4.3.1 ASIC Implementations 79
4.3.2 Results and Comparisons 79
4.4 Conclusion 81

5 Digit-Level Architectures for $GF(2^m)$ Multiplication in the GNB 83
5.1 Proposed DL-FSIPO Single GNB Multipliers 86
5.1.1 Proposed MSD DL-FSIPO Single GNB Multiplier 86
5.1.1.1 Formulations 86
5.1.1.2 Architecture 89
5.1.1.3 Space and Time Complexities 90
5.1.1.4 Bit-Level Case 92
5.1.2 Proposed LSD DL-FSIPO Single GNB Multiplier 93
5.1.2.1 Formulations 93
5.1.2.2 Architecture 96
5.1.2.3 Space and Time Complexities 97
5.2 Proposed DL-PISO Single GNB Multiplier 98
5.2.1 Architecture 98
5.2.2 Space and Time Complexities 100
5.3 Proposed Digit-Level Hybrid-Double and Hybrid-Triple GNB Multipliers . . . 101
5.3.1 Proposed MSD DL-SIPO Hybrid-Double GNB Multiplier 101
5.3.2 Proposed DL-PIPO Hybrid-Triple GNB Multiplier 103
5.3.3 Space and Time Complexity Analysis 105
5.3.4 Hybrid Versus Single Digit-Level GNB Multipliers 106
5.4 Proposed Architecture for Field Exponentiation 107
5.5 Conclusion 109

6 Digit-Level Architectures for $GF(2^m)$ Multiplication in the PB 110

6.1 Proposed MSD DL-FSIPO PB Multiplier . . . 111

6.1.1 Formulations . . . 111

6.1.2 Architecture . . . 114

6.1.3 Space and Time Complexities . . . 116

6.2 Proposed LSD DL-FSIPO PB Multiplier . . . 118

6.2.1 Formulations . . . 119

6.2.2 Architecture . . . 124

6.2.3 Space and Time Complexities . . . 126

6.3 Comparisons . . . 131

6.4 Conclusion . . . 136

7 Summary and Future Work 137

7.1 Summary of Contributions . . . 137

7.2 Future Work . . . 138

Bibliography 139

Curriculum Vitae 146

List of Figures

2.1 A stream cipher is used for providing privacy over an insecure channel between two communicating entities. 21
2.2 A general block diagram of a WG stream cipher. 22

3.1 Proposed MOWG transformation. 28
3.2 Proposed design of the MOWG$(29, 11, 17)$ cipher. 31
3.3 FSM of the MOWG. 32
3.4 The proposed design of the WG transformation. 39
3.5 Modified FSM after adding the new 3-bit one-hot counter. 41
3.6 Block diagram of the SKIM module. 42
3.7 The proposed WG transformation after integration with the SKIM module. 43
3.8 Serial Implementation of MOWG/WG Stream Ciphers. 49

4.1 Contributions of this work. 53
4.2 The matrix **S** for WG$(29, 11)$. 56
4.3 Architecture of the WG$(29, 11)$ stream cipher. 60
4.4 Architecture of the $2^{10} - 1$ module. 61
4.5 FSM for the PB based implementation of the WG$(29, 11)$ stream cipher. 62
4.6 Architecture of the serial $WGP29/WGT29$ implementation. 64
4.7 a) Architecture of the FSM for the serialized implementation of the WG$(29, 11)$. b) Generating the Clock Enable Control Signals and the Multiplexers' Selectors. 65
4.8 Pipelined version of the $WGT29$. 66
4.9 Architecture of the FSM for the pipelined version of the WG$(29, 11)$. 67
4.10 Clock enable control signals for the pipelined version of the WG$(29, 11)$. 68
4.11 The matrix **S** for WG-16. 70
4.12 a) Architecture of the WG-16. b) Generation of the signal Y^s ($s = 2^5 - 1$). c) Generation of the signal $(A_{i+31})^{1057}$. 72
4.13 Architecture of the serial implementation for the PB based design of the WG-16. 75
4.14 Generating the Clock Enable Control Signals and the Multiplexers' Selectors for the serial version of the WG-16. 76

4.15 Pipelined version of the WG-16 transform. 77
4.16 Generating the clock enable signals and, ctrl0 and ctrl1 signals for the pipelined version of the WG-16. 79

5.1 Summary of contributions. 84
5.2 (a) Architecture of the proposed MSD DL-FSIPO single GNB multiplier. (b) Architecture of ∇_j. (c) Architecture of β_j. 89
5.3 Architecture of the proposed LSD DL-FSIPO single GNB multiplier. 96
5.4 (a) The proposed architecture of the MSD DL-PISO single GNB multiplier. . . 99
5.5 Architectures of the proposed MSD DL-SIPO hybrid-double GNB multiplier. (a) Low area design. (b) High speed design. 102
5.6 Architectures of the proposed MSD DL-PIPO hybrid-triple GNB multiplier. (a) Low area design. (b) High speed design. 104
5.7 Architecture of the proposed eight-ary field exponentiation scheme. 108

6.1 (a) Architecture of the proposed MSD DL-FSIPO PB multiplier. (b) Detailed architecture of $\triangle_j$. (c) Architecture of $\otimes$ module. 115
6.2 The state of the corresponding $GF\left(2^3\right)$ MSD DL-FSIPO PB multiplier for Example 6.1.3, throughout the different iterations of the computation. (a) initial state. $i = 0$. (b) state after first clock cycle. $i = 1$. (c) state after second clock cycle. $i = 2$. (d) state after third clock cycle. 117
6.3 (a) Architecture of the proposed LSD DL-FSIPO PB multiplier. (b) Detailed architecture of $\triangle_j^{'}$ at i-th iteration. 125
6.4 The state of the corresponding $GF\left(2^3\right)$ LSB BL-FSIPO PB multiplier for Example 6.2.3, throughout the different iterations of the computation. (a) initial state. $i = 0$. (b) state after first clock cycle. $i = 1$. (c) state after second clock cycle. $i = 2$. (d) state after third clock cycle. 126
6.5 Multiplying an arbitrary $GF(2^m)$ element by the constant α^{-q} where$p(x) = x^m + \sum_{i=1}^{\omega-2} x^{t_i} + 1$ is the field's generating irreducible polynomial with ω nonzero terms and $q \leq t_1$ (condition of (6.11)). 129
6.6 Normalized throughput as a function of the digit size for the serial inputs loading case. 132
6.7 Normalized throughput as a function of the digit size for the parallel inputs loading case. 133

List of Tables

1.1 Comparison of strength of AES, RSA, DSA, and ECDSA [18]. 1

2.1 Addition over $GF(5)$. 9
2.2 Subtraction over $GF(5)$. 9
2.3 Multiplication over $GF(5)$. 10

3.1 Phase of operation in the proposed MOWG as a function of the state of the 2-bit binary counter. 33
3.2 Count of 1-bit registers and logic gates in the different components of the proposed MOWG design. 34
3.3 Signals s_0 and s_1 as a function of the output of the 3-bit one-hot counter. 42
3.4 Multiplexers outputs and next states of Register 1 and Register 2 as a function of s_0 and s_1. 44
3.5 Count of 1-bit registers and logic gates in the components of the proposed WG$(29, 11)$. 45
3.6 Results obtained from ASIC implementations. 47
3.7 Results obtained from FPGA implementations. 47

4.1 The space and time complexities of the different squaring matrices used in the WG$(29, 11)$. 57
4.2 Computation of the $IF = WGP29$ signal over 3 clock cycles during the initialization phase. 61
4.3 Phase of operation in the proposed PB based WG designs as a function of the state of the 2-bit binary counter. 63
4.4 Steps for computing the $WGP29$ and $WGT29$ in the serial implementation of the WG$(29, 11)$ design. 63
4.5 Space and propagation delay complexities of the different squaring matrices used in the WG-16. 70
4.6 Computation of the $WGP16$ signal over 3 clock cycles. 73
4.7 Computing $WGP16$ and $WGT16$ in the serial implementation of WG-16. . . . 76

4.8 Results obtained for area and speed from the ASIC implementations. 82

5.1 Steps for multiplication of the two $GF\left(2^3\right)$ elements $A = B = \beta^{2^2} = (0, 0, 1)$. . . 88
5.2 Space complexity of digit-level single GNB multipliers. 91
5.3 Time complexity of digit-level single GNB multipliers. 91
5.4 Space and time complexity readings for the case of type-4 GNB of $GF\left(2^{163}\right)$ digit-level single multipliers. 92
5.5 Steps for multiplication of the two $GF\left(2^3\right)$ elements $A = B = \beta^{2^2} = (0, 0, 1)$. . . 95
5.6 Space complexity of the digit-level hybrid-double and hybrid-triple GNB multipliers. 105
5.7 Time complexity of the digit-level hybrid-double and hybrid-triple GNB multipliers. 105

6.1 Example 6.1.3 for multiplying the two $GF\left(2^3\right)$ elements $A = \alpha = (0, 1, 0)$ and $B = \alpha^2 = (1, 0, 0)$ using (6.1) and (6.2). 114
6.2 Example 6.2.3 for multiplying the two $GF\left(2^3\right)$ elements $A = \alpha = (0, 1, 0)$ and $B = \alpha^2 = (1, 0, 0)$ using (6.6) and (6.7). 122
6.3 Space and time complexities of the different digit-level $GF(2^m)$ PB multipliers. 131
6.4 Space and time complexities for the NIST recommended field $GF\left(2^{233}\right)$ defined by the irreducible trinomial $x^{233} + x^{t_1} + 1$, where $t_1 = 74$ and the digit size is $d = 1$. 134
6.5 Space and time complexity estimates for the multipliers which are listed in Table 6.4 based on on the standard 65nm CMOS library measures. 135

Nomenclature

3GPP	The 3rd generation partnership project
4G	Fourth generation mobile communications domain
AES	Advanced encryption standard
ASIC	Application specific integrated circuits
CMOS	Complementary metal-oxide-semiconductor technology
DL	Digit-level
Double field multiplication	Multiplication of three field elements
DSA	Digital signature algorithm
DSS	Digital signature standard
ECDSA	Elliptic curve digital signature algorithm
FF	Flip-Flop
FLT	Fermat's Little Theorem
FPGA	Field programmable gate array
FSIPO	Fully-serial-in-parallel-out
FSM	Finite state machine
GF(2)	The binary finite field with elements 0 and 1
$GF(2^m)$	Galois Field with 2^m elements
GNB	Gaussian normal basis

i.e.	That is
IP	Inner product
IP-networks	Internet protocol networks
LFSR	Linear feedback shift register
LSB	Least significant bit first
LSD	Least significant digit first
LTE	Long term evolution
m-sequence	Maximal length sequence
MOWG	Multiple output-bits version of the WG stream cipher
MPD	Maximum propagation delay
MSB	Most significant bit first
MSD	Most significant digit first
MUX	Multiplexer
NB	Normal Basis
NIST	National institute of standards and technology
ONB-II	Optimal normal basis of type 2
PB	Polynomial Basis
PD	Propagation delay
PIPO	Parallel-in-parallel-out
PISO	Parallel-in-serial-out
PRSG	Psuedo random sequence generator
RFID	Radio frequency identification
ROM	Read-only memory
Single field multiplication	Multiplication of two field elements

SIPO	Serial-in-parallel-out
SNOW 3G	The stream cipher SNOW 3G, which is included in the 4G network domain's cipher suite
SSL	Secure sockets layer
TLS	Transport layer security
TP	Throughput: number of output bits per second
Tr(A)	The trace of a field element A, which is a mapping from $GF(2^m)$ to either 0 or 1
Triple field multiplication	Multiplication of four field elements
w.r.t	With respect to
WEP	Wired equivalent privacy
WG	Welch-Gong
WG Stream Cipher	Welch-Gong transform based Stream Cipher
WG(29, 11)	WG stream cipher with an LFSR of length 11 over $GF(2^{29})$
WG(m, l)	WG stream cipher with an LFSR of length l over $GF(2^m)$
WGP	Welch-Gong (WG) permutation
WGT	Welch-Gong (WG) transform
WPA	Wi-Fi protected access
XOR	Logical binary exclusive OR operation
XST	Xilinx synthesis tool
ZUC	The stream cipher ZUC, which is included in the 4G network domain's cipher suite

Chapter 1

Introduction

Cryptographic algorithms play essential role in communications systems security. In general, the different deployed cryptosystems are divided into two categories of asymmetric key and symmetric key [82]. Schemes from the former category require relatively high space and time complexities for their hardware implementations. Hence, asymmetric key cryptosystems are usually deployed in key set-up mechanisms. On the other hand, schemes from the latter category offer hardware implementations which require relatively lower complexities in terms of space and time. Hence, symmetric key cryptosystems are usually used for providing security services during communications sessions. The lower implementation cost of symmetric key systems is due to the simpler underlying mathematical constructs, in addition to the smaller key sizes required by these cryptosystems to achieve certain security levels compared to the asymmetric key ones. Table 1.1 presents a comparison of key sizes required by AES, RSA, DSA, and ECDSA in order to achieve some security levels (in terms of bits of security). In this table, k indicates the key size for AES, while it refers to the size of the modulus in RSA. l and n, respectively, indicate the size of public and private keys for DSA. For ECDSA, f is considered to be the key size.

Bits of Security	AES	RSA	DSA	ECDSA
128	$k = 128$	$k = 3072$	$l = 3072$, $n = 256$	$256 \leq f \leq 383$
192	$k = 192$	$k = 7680$	$l = 7680$, $n = 384$	$384 \leq f \leq 511$
256	$k = 256$	$k = 15360$	$l = 15360$, $n = 512$	$f \geq 512$

Table 1.1: Comparison of strength of AES, RSA, DSA, and ECDSA [18].

Efficient hardware implementations of the deployed cryptosystems are necessary for their practical use. In other words, the implementation of a given cryptosystem should comply with the performance requirements of the underlying communications system. Therefore, this research focuses on finding efficient hardware implementations for symmetric key cryptosystems, targeting mobile and resource constrained applications, as it is stated in the following section.

1.1 Objectives

The goal of this research is to introduce efficient hardware implementations for symmetric key cryptosystems targeting the mobile communications domain and resource constrained applications, as follows.

First, this research aims for lightweight hardware implementations for the two classes of the Welch-Gong (WG) family of stream ciphers, $\mathrm{WG}(29, 11)$ and WG-16. The targeted lightweight implementations will provide trade-offs between space and time complexities for a set of lightweight applications. Specifically, the WG-16 implementations are intended for the Long term evaluation (LTE) 4G mobile communications domain.

After this, the research focuses on finding new finite field constructs for higher throughput in resource constrained applications. New architectures for higher throughput digit-level field multiplications will be investigated for resource constrained applications. The issue of reduced throughput in digit-level $GF(2^m)$ multiplication of two elements (single multiplication), due to inputs preloading in applications where the input data-path has limited capacity and the value of m (dimension of the binary extension field) is large, will be addressed. This issue will be considered for both GNB and PB representations. Furthermore, new architectures for higher throughput concurrent multiplications of three (hybrid-double multiplication) and four (hybrid-triple multiplication) field elements will also be explored for the GNB representations. New field exponentiation architectures based on the eight-ary scheme [42] will be considered as a practical application of the hybrid-triple multipliers.

The following section highlights the motivations and significance of the research.

1.2 Motivations and Significance

This section states the importance and significance of the underlying research.

Stream ciphers are symmetric key cryptosystems which are attractive for protecting the wireless communications domain [24]. This is because stream ciphers prevent error propagation at the receiving end. In this context, a stream cipher is required to provide the desired security, generate key-stream sequences with good randomness properties, and show efficient performance [24]. In addition, stream ciphers can also be used as random number generators for other algorithms (for example, generating random numbers for the Digital signature standard ”DSS” [12]). In this type of applications, the randomness of the generated key sequences is very critical. This thesis considers the two classes of WG(29, 11) and WG-16 stream ciphers. In addition of resistance to all known attacks, to the best of the author knowledge, these two ciphers provide a set of desired randomness properties which can not be offered by other existing ciphers [40, 68]. Therefore, new lightweight hardware implementations of the WG(29, 11) and WG-16 stream ciphers are introduced. The new designs provide trade-off between randomness properties and performance for a selection of cryptosystems. In particular, the new hardware implementations of the WG-16 cipher provide different space options while complying with the throughput requirements of the 4G domain. This, makes the WG-16 cipher an interesting candidate for securing the 4G mobile domain.

The thesis then considers efficient digit-level $GF(2^m)$ multipliers. Digit-level field multipliers trade-off space complexity with lower throughput. Hence, digit-level field multipliers are important for resource constrained applications. Any improvement in such operation is considered of great value to a wide range of applications, such as: symmetric and asymmetric crypto algorithms, error coding, random number generation, and digital signal processing. In this context, this research proposes new architectures for single (multiply two elements), hybrid-double (multiply three elements), and hybrid-triple (multiply four elements) digit-level $GF(2^m)$ multipliers, as follows.

New GNB and PB single multipliers are introduced, targeting higher throughput for digit-level field multiplications in resource constrained applications. In particular, for cases where the input data-path has limited capacity, higher throughput is achieved through removing the requirement for inputs preloading.

In addition, new hybrid-double and hybrid-triple GNB multipliers are introduced. The new hybrid architectures improve the throughput of concurrent multiplications where three and four field elements are multiplied at the same time. Also, these hybrid multiplier architectures are advantageous for improving throughput of the important operations of field inversion [51]. It is noted that, the hybrid-triple multiplier is proposed for the first time in the literature.

As another practical application for the hybrid-triple multipliers, field exponentiation is an essential operation in asymmetric key cryptography (such as Diffie-Hellman key exchange algorithm [29]) and symmetric key cryptography (WG for example). The proposed hybrid-triple GNB multipliers are utilized in constructing new eight-ary exponentiation schemes. This results in hardware exponentiation architectures which run at same latencies as the existing eight-ary designs, however, without requiring any initial phase for precomputations, or any storage of intermediate variables.

The following section outlines the rest of the thesis.

1.3 Thesis Outline

The thesis is outlined as follows. Chapter 1 highlights the objectives, motivations, and significance, and, outlines the remaining chapters. Chapter 2 presents a brief overview about binary extension fields and WG stream ciphers, as it suffices for understanding the rest of this thesis. Chapter 3 introduces efficient hardware implementations for the WG$(29, 11)$ stream cipher based on the Optimal Normal Basis Type-II (ONB-II) representation of the feild elements. Chapter 4 introduces efficient hardware implementations for the WG$(29, 11)$ and WG-16 stream ciphers based on the Polynomial Basis (PB) representation of the feild elements. Chapter 5 proposes new architectures for digit-level single, hybrid-double, and hybrid-triple field multiplications and exponentiation based on the Gaussian Normal Basis (GNB) representation. Chapter 6 proposes new architectures for digit-level single field multiplications based on the Polynomial Basis (PB) representation. Chapter 7 highlights the contributions of this thesis and lists some future works.

Chapter 2

Background on $GF(2^m)$ and WG Stream Ciphers

This chapter starts by a brief introduction on binary extension fields and Welch-Gong (WG) stream ciphers.

Arithmetic operations over binary extension fields are extensively encountered in the next four chapters. This chapter presents necessary background about binary extension fields, as it suffices for the purpose of clarifying contents of the remaining chapters. Other references can be consulted for more reading about finite fields, for example, [82, 59]. For the purpose of this work, in the following sections, some preliminary definitions are first given. This is followed by introducing finite fields, modular arithmetic, polynomial rings, and binary extension fields. After this, the two representations of Polynomial basis (PB) and Gaussian normal basis (GNB) of binary extension fields elements are discussed. Then, field operations over binary extension fields are reviewed. At the end of the sections which are dedicated for reviewing $GF(2^m)$, from this chapter, the trace mapping is presented. It is noted that, the material presented throughout the above-mentioned sections is a summary based on reviews done over [82, 59, 5].

After introducing $GF(2^m)$, the chapter introduces WG Stream Ciphers. First, a brief introduction to stream ciphers is given. This is followed by an overview on WG stream ciphers, where the chapter talks about general WG stream ciphers block diagrams and phases of operation, after which the two classes of WG(29, 11) and WG-16 are discussed.

2.1 Preliminaries on Algebraic Structures

Before starting the presentation about finite fields, the following definitions are required.

Definition 2.1.1 *The algebraic structure* $(G, \star)$ *which consists of a set G and a binary opera-*

tion $\star$ is said to be a semigroup if:

- *$\star$ is closed over G, that is: for any $g_1, g_2 \in G$, then, $g_1 \star g_2 \in G$.*
- *$\star$ is associative over G, that is: for any $g_1, g_2, g_3 \in G$, then, $(g_1 \star g_2) \star g_3 = g_1 \star (g_2 \star g_3)$.*

Definition 2.1.2 *The algebraic structure $(G, \star)$ forms a group if it is a semigroup, and:*

- *G contains an identity element e with respect to $\star$, where: for any $g_1 \in G$, then, $g_1 \star e = e \star g_1 = g_1$.*
- *For each element $g_1 \in G$, there exists an inverse element $g_2 \in G$ with respect to $\star$, where: $g_1 \star g_2 = g_2 \star g_1 = e$.*

Definition 2.1.3 *The algebraic structure $(G, \star)$ is called an abelian group if it is a group, and:*

- *$\star$ is commutative over G, that is: for any $g_1, g_2 \in G$, then, $g_1 \star g_2 = g_2 \star g_1$.*

Definition 2.1.4 *The algebraic structure $(R, +, \cdot)$ which consists of a set R and the two binary operations + (additive) and $\cdot$ (multiplicative) is called a ring if:*

- *$(R, +)$ is an abelian group. The additive identity element is usually denoted by 0. The additive inverse of any element $r_1 \in R$ is denoted by $-r_1$.*
- *$(R, \cdot)$ is a semigroup.*
- *$\cdot$ is distributive over +, that is: for any $r_1, r_2, r_3 \in R$, then, $r_1 \cdot (r_2 + r_3) = r_1 \cdot r_2 + r_1 \cdot r_3$, and, $(r_2 + r_3) \cdot r_1 = r_2 \cdot r_1 + r_3 \cdot r_1$.*

Definition 2.1.5 *A homomorphism is a map between two algebraic structures (such as groups, rings, and so on) through which the operations are preserved. For example, for the two groups $(G, \star)$ and $(H, \triangle)$, the mapping $\phi : G \to H$ is a homomorphism if $\forall g_1, g_2 \in G \implies \phi(g_1 \star g_2) = \phi(g_1) \triangle \phi(g_2)$. If in addition, ϕ is surjective*[1] *(onto) and one-to-one, then ϕ is an isomorphism.*

The following section introduces finite fields.

---

[1] every element in H has at least one corresponding element in G

2.2 Finite Fields

2.2.1 Fields

An algebraic structure $(F, +, \cdot)$, constructed from a set F and the two binary operations $+$ and $\cdot$ forms a field if:

- $(F, +)$ is an abelian group.
- $(F - \{0\}, \cdot)$ is an abelian group. The multiplicative identity element is usually denoted by 1. The multiplicative inverse of any element $f_1 \in F$ is denoted by f_1^{-1}.
- Multiplication is distributive over addition.
- There exist no zero divisors over F, that is, for any $f_1, f_2 \in F$, if $f_1 \cdot f_2 = 0$, then, either $f_1 = 0$ or $f_2 = 0$.

2.2.2 Finite Fields

A finite field, known as *Galois Field*, and denoted by $GF(q)$ (or $\mathbb{F}_q$), is a field $\left(F_q, +, \cdot\right)$, where F_q is a set with finitely q elements. The order of $GF(q)$, that is the number of field elements q, is a positive integer which is either a prime or a power of a prime (including powers of 2). In $GF(q)$, there is a zero element, while the remaining $q - 1$ elements form the multiplicative group of the field (all elements which have multiplicative inverses). For a given order q, there might be more than one representation of the corresponding finite field $GF(q)$. However, all finite fields of a given order are isomorphic (have same structure).

In simple words, a finite field is a set with finitely many elements over which one can perform the operations of addition, subtraction, multiplication, and division, while staying in the same set.

The following section introduces modular arithmetic and highlights the relation between the algebraic structure $\left(\mathbb{Z}_p, +, \cdot\right)$ and finite fields of the form $GF(p)$, where $\mathbb{Z}$ denotes the set of integers, p is a prime number, and $\mathbb{Z}_p$ is the set of positive integers less than p (including 0).

2.3 Modular Arithmetic

This section gives a brief presentation about modular arithmetic. Modular addition and multiplication are carried out as they are done over the set of integers $\mathbb{Z}$, however, followed by reducing the result modulo an integer $m > 0$. The integer m is referred to as the modulus.

Applying the ”modulo m” operator to a given integer $v \in \mathbb{Z}$, written as $v \bmod m$, returns the remainder out of dividing v by m (long division). For example, let $v = 7$ and $m = 3$. Then, $7 \bmod 3 = 1$, since $7 = 2 \times 3 + 1$. If there exists a $v^{'} \in \mathbb{Z}$ such that $v \bmod m = v^{'} \bmod m$, one can also write $v \equiv v^{'} \pmod{m}$. The latter expression is read as: v is equivalent (or is congruent) to $v^{'}$, modulo m. In this expression, $\equiv$ is known as the equivalence (or congruence) operator. If $v \equiv 0 \pmod{m}$, that is, $v \bmod m = 0$, then, m divides v, which is simply written as $m \mid v$. If $v \equiv w \pmod{m}$, that is, $v \bmod m = w \bmod m$, then, $m \mid |v - w|$, where $|\cdot|$ denotes the absolute value operator. The following are some properties of congruences modulo an integer $m > 0$:

- $\forall v \in \mathbb{Z}, v \equiv v \pmod{m}$.
- $\forall v, w \in \mathbb{Z}, v \equiv w \pmod{m} \implies w \equiv v \pmod{m}$.
- $\forall v, w, x \in \mathbb{Z}$, if $v \equiv w \pmod{m}$ and $w \equiv x \pmod{m}$, then $v \equiv x \pmod{m}$.
- $\forall v, w, x, y \in \mathbb{Z}$, if $v \equiv w \pmod{m}$ and $x \equiv y \pmod{m}$, then $v \pm w \equiv x \pm y \pmod{m}$.
- If n is a non-zero positive integer such that $n \mid m$, therefore $\forall v, w \in \mathbb{Z}$ where $v \equiv w \pmod{m}$, then $v \equiv w \pmod{n}$.
- Let n be a non-zero positive integer such that the greatest common divisor of m and n is 1, that is, $\gcd(m, n) = 1$. Therefore, $\forall v, w \in \mathbb{Z}$, if $v \equiv w \pmod{m}$ and $v \equiv w \pmod{n}$, then $v \equiv w \pmod{mn}$.

Now, denote by $\mathbb{Z}_m$ the set of residue classes modulo m. That is, $\mathbb{Z}_m = \{0, \ldots, m - 1\}$ consists of integers modulo m. In general, not all elements of $\mathbb{Z}_m$ have multiplicative inverses. This is the reason why $(\mathbb{Z}_m, +, \cdot)$ forms a commutative ring and not a finite field. Only elements of $\mathbb{Z}_m$ which are relatively prime to m have multiplicative inverses. Therefore, if $m = p$ is a prime, then, multiplicative inverses exist for all non-zero elements in $\mathbb{Z}_p$. In this case, $\mathbb{Z}_p$ is an abelian group under multiplication, and hence, $\left(\mathbb{Z}_p, +, \cdot\right)$ forms a finite field (since $\left(\mathbb{Z}_p, +\right)$ is also an abelian group, $\cdot$ is distributive over $+$, and there are no zero divisors in $\mathbb{Z}_p$). Operations over the finite field $\left(\mathbb{Z}_p, +, \cdot\right)$ are isomorphic to those over $GF(p)$. The following is an illustrative example for arithmetic operations over $GF(5)$. After this, next section introduces polynomial rings.

Example 2.3.1 *The field* $GF(5)$ *contains the elements* $\{0, 1, 2, 3, 4\}$. *The additive and multiplicative identities are* 0 *and* 1, *respectively. Arithmetic operations are performed modulo* 5, *as it is shown for the cases of addition, subtraction, and multiplication, in Tables 2.1, 2.2, and 2.3, respectively. Notice from Table 2.3 that all non-zero elements have multiplicative inverses.*

+	0	1	2	3	4
0	0	1	2	3	4
1	1	2	3	4	0
2	2	3	4	0	1
3	3	4	0	1	2
4	4	0	1	2	3

Table 2.1: Addition over $GF(5)$.

−	0	1	2	3	4
0	0	4	3	2	1
1	1	0	4	3	2
2	2	1	0	4	3
3	3	2	1	0	4
4	4	3	2	1	0

Table 2.2: Subtraction over $GF(5)$.

2.4 Polynomial Rings

The algebraic structure $(R, +, \cdot)$ forms a ring if $(R, +)$ is an abelian group, $(R, \cdot)$ is a semigroup, and multiplication is distributive over addition. A polynomial ring is a structure $(R[x], +, \cdot)$ such that $R[x] = \left\{\sum_{i=0}^{n} r_i x^i \mid n \geq 0, r_i \in R\right\}$ represents the set of all polynomials in the variant x with coefficients from R, where addition and multiplication, respectively, are defined as follows

$$\sum_{i=0}^{n'} r'_i x^i + \sum_{i=0}^{n''} r''_i x^i = \sum_{i=0}^{\max\left(n', n''\right)} \left(r'_i + r''_i\right) x^i,$$

and

$$\left(\sum_{i=0}^{n'} r'_i x^i\right) \cdot \left(\sum_{i=0}^{n''} r''_i x^i\right) = \sum_{i=0}^{n'+n''} \left(\sum_{j+k=i} r'_j \cdot r''_k\right) x^i.$$

Notice that, while exact division applies to fields, only long division is applicable to rings. This is because field elements have inverses, while this is not the case for elements of a ring. Therefore, in order to have long division in a polynomial ring, coefficients need to be from a field and not a ring. Otherwise, long division might not be possible over a polynomial ring. The following is an example showing operations done over elements of the polynomial ring $(GF(2)[x], +, \cdot)$.

$\cdot$	0	1	2	3	4
0	0	0	0	0	0
1	0	1	2	3	4
2	0	2	4	1	3
3	0	3	1	4	2
4	0	4	3	2	1

Table 2.3: Multiplication over $GF(5)$.

Example 2.4.1 *Let* $f(x) = x^4+1$ *and* $g(x) = x^3+x+1$ *be two elements from the polynomial ring* $(GF(2)[x], +, \cdot)$. $GF(2)$ *is the binary finite field with elements* 0 *and* 1*, which is isomorphic to* $(\mathbb{Z}_2, +, \cdot)$*. Notice that, over* $GF(2)$*:* $1+1 = 0+0 = 0$ *and* $0+1 = 1+0 = 1$*, while* $0 \cdot 0 = 0 \cdot 1 = 1 \cdot 0 = 0$ *and* $1 \cdot 1 = 1$*. Then, one conducts addition, multiplication, and division over these two elements as follows:*
1) Addition:

$$\begin{aligned} f(x) + g(x) &= \left(x^4 + 1\right) + \left(x^3 + x + 1\right) \\ &= x^4 + x^3 + x. \end{aligned}$$

2) Multiplication:

$$\begin{aligned} f(x) \cdot g(x) &= \left(x^4 + 1\right) \cdot \left(x^3 + x + 1\right) \\ &= x^7 + x^5 + x^4 + x^3 + x + 1. \end{aligned}$$

3) Division (long division):

$$x^4 + 1 = \left(x^3 + x + 1\right) \cdot (x) + \left(x^2 + x + 1\right),$$

where $q(x) = x$ *is the quotient of* ${}^{f(x)}/_{g(x)}$ *and* $r(x) = x^2 + x + 1$ *is its remainder.*

Notice that in the latter operation, the degree of $r(x)$ *is less than that of* $g(x)$*, written as* $\deg(r(x)) < \deg(g(x))$*. In general,* $0 \leq \deg(r(x)) < \deg(g(x))$*. If there exists a* $g(x)$ *such that* $r(x) = 0$*, then, in this case* $g(x) \mid f(x)$*, and hence,* $f(x)$ *is reducible over* $GF(2)$*. If there is no such* $g(x)$ *which divides* $f(x)$*, then,* $f(x)$ *is an irreducible polynomial over* $GF(2)$*.*

The following section introduces binary extension fields $GF(2^m)$ and shows how irreducible polynomials are used to construct such fields.

2.5 Binary Extension Fields $GF\left(2^m\right)$

An extension field over the finite field $GF(p)$ is referred to as $GF(p^m)$. $GF(p)$ is denoted as the ground field. Specifically, for $p = 2$, $GF(2^m)$ is denoted as the binary extension field (over $GF(2)$). A $GF(2^m)$ can be viewed as a vector space over $GF(2)$ of dimension m. Hence, a $GF(2^m)$ is isomorphic to ${}^{\mathbb{Z}_2[x]}/_{p(x)}$ under polynomial addition and multiplication, where ${}^{\mathbb{Z}_2[x]}/_{p(x)}$ denotes the set of polynomials in variant x with coefficients from $\mathbb{Z}_2$ taken modulo an irreducible polynomial $p(x)$ of degree m. There are 2^m elements in $GF(2^m)$. Each one of these elements is uniquely represented by m bits (elements of $GF(2)$ with value 0 or 1) with respect to a basis. A basis is a set of m linearly-independent elements $\mathit{\Gamma} = \{\gamma_i \in GF(2^m) \mid 0 \le i < m\}$ [59]. Then, an element $A \in GF(2^m)$ is represented with respect to $\mathit{\Gamma}$ as $A = \sum_{i=0}^{m-1} a_i\gamma_i$, where $a_i \in GF(2)$ for $0 \le i < m$ are the binary coordinates of A w.r.t $\mathit{\Gamma}$. The following two sections introduce two of the most common representations of the elements of $GF(2^m)$.

2.6 Polynomial Basis (PB) Representation

The most straight forward representation for the elements of $GF(2^m)$ is obtained from the isomorphism of $GF(2^m)$ to ${}^{\mathbb{Z}_2[x]}/_{p(x)}$. Here, $p(x)$ is an irreducible polynomial of degree m over $\mathbb{Z}_2$. That is, coefficients of the terms of $p(x)$ are either 0 or 1. Therefore, following this construction, elements of $GF(2^m)$ include the binary representations of all polynomials in x of degree less than or equal to $m - 1$.

A PB follows the form $\left\{\alpha^{m-1}, \ldots, \alpha, 1\right\}$ and is constructed by finding a root $\alpha \in GF(2^m)$ of an irreducible polynomial $p(x)$ of degree m over $GF(2)$ [72]. Then, using this PB, an element A in $GF(2^m)$ is represented as $A = \sum_{i=0}^{m-1} a_i\alpha^i$, where a_i is either 0 or 1 for $0 \le i < m$. In vector representation, the element A can also be refered to as $A = (a_{m-1}, \ldots, a_0)$ [5].

2.7 Gaussian Normal Basis (GNB) Representation

On the other hand, a Normal basis (NB) is constructed by finding an element $\beta \in GF(2^m)$ such that the m elements β^{2^0} through $\beta^{2^{m-1}}$ are linearly-independent [59]. Then, the set $\left\{\beta^{2^0}, \ldots, \beta^{2^{m-1}}\right\}$ forms a NB where β is called a normal element. In the NB, an element A is represented as $A = \sum_{i=0}^{m-1} a_i\beta^{2^i}$, with $a_i \in \{0, 1\}$ representing the i-th coordinate of A with respect to the NB, for $0 \le i < m$. In vector representation, the element A can also be represented as $A = (a_0, \ldots, a_{m-1})$ [5].

Gaussian normal bases (GNBs) is a special subset of NBs which offer field operations with smaller area and time overhead compared to the general NB, when realized in hardware. A

GNB exists for all $GF(2^m)$ which satisfy the following conditions [15, 52, 70]:

1. m is not divisible by 8, and
2. there exists a prime integer $p = Tm + 1$, with $T > 0$ is an integer, such that $\gcd\left(\frac{Tm}{k}, m\right) = 1$, and k is the order of 2 modulo-p (that is $2^k \equiv 1 \pmod{p}$).

and hence, T is called the type of the GNB. It is noted that for odd values of m the type T should be even (since p is an odd prime). It is also noted that, smaller values of T results in more efficient hardware implementations of field multiplications. Similar to the NB representation, any element $A \in GF(2^m)$ can be represented w.r.t the GNB as $A = \sum_{i=0}^{m-1} a_i \beta^{2^i} = (a_0, \ldots, a_{m-1})$, where $a_i \in \{0, 1\}$.

Common $GF(2^m)$ arithmetic operations include addition, multiplication, exponentiation, and inversion. The following three sections present more details about arithmetic operations over $GF(2^m)$ considering the two cases of PB and GNB representations.

2.8 Addition over $GF(2^m)$

Field addition of two arbitrary $GF(2^m)$ elements, say A and B, is accomplished by a bit-wise Exclusive-OR (XOR) operation on the corresponding coordinates of the added elements, regardless whether a PB or a GNB representation is used. That is:

$$A + B = \sum_{i=0}^{m-1} (a_i + b_i)\,\gamma_i,$$

where a_i and b_i are the binary coordinates of A and B with respect to a given basis $\Gamma = \{\gamma_0, \ldots, \gamma_{m-1}\}$.

2.9 Multiplication over $GF(2^m)$

On the other hand, the field multiplication of two arbitrary $GF(2^m)$ elements A and B is accomplished as follows:

$$AB = \sum_{i=0}^{m-1} \sum_{j=0}^{m-1} a_i b_j \gamma_i \gamma_j,$$

which is more expensive than field addition, and its complexity depends on the underlying representation $\Gamma = \{\gamma_0, \ldots, \gamma_{m-1}\}$. It is known that the PB representation offers efficient hardware implementations of field multiplications, compared to the NB / GNB representation [71]. The following two sections give brief literature reviews of $GF(2^m)$ multiplications in the PB and GNB representations, respectively, as it suffices for the purpose of this thesis.

2.9.1 $GF(2^m)$ Multiplication in the PB Representation

Two popular schemes for the multiplication of two $GF(2^m)$ elements in the PB representation are: the two-step classic multiplication scheme and the Matrix-vector scheme [28]. The first scheme starts by performing polynomial multiplication of the two input field elements, then, the result is reduced modulo the irreducible defining field polynomial [89]. The following is an example illustrating this multiplication scheme.

Example 2.9.1 *This example constructs the field* $GF\left(2^3\right)$ *using the irreducible polynomial* $p(x) = x^3 + x + 1$ *over* $GF(2)$. *Denote by* α *the root of* $p(x)$ *over* $GF\left(2^3\right)$. *By using* α, *the polynomial basis* $\left\{\alpha^2, \alpha, 1\right\}$ *is constructed. Based on this polynomial basis, any element* A *from the* $2^3 = 8$ *elements of* $GF\left(2^3\right)$ *is defined by a unique set of* 3 *binary coordinates as* $A = a_2\alpha^2 + a_1\alpha + a_0$. *For example, by considering the* $GF\left(2^3\right)$ *elements* $\alpha^2 + \alpha + 1$ *and* α, *then, one has* $\left(\alpha^2 + \alpha + 1\right) \times (\alpha) = \alpha^3 + \alpha^2 + \alpha = \alpha^2 + 1$ *over* $GF\left(2^3\right)$. *The latter result is obtained after reducing* $x^3 + x^2 + x$ *by* $p(x) = x^3 + x + 1$ *(also, it can be obtained by noticing that* $p(\alpha) = 0$, *which results in* $\alpha^3 = \alpha + 1$*).*

In the second scheme, known as the Mastrovito multiplier [62, 84, 44, 72], one performs the field multiplication in terms of vector by matrix multiplication, in which both steps of the former scheme are combined into a single step. In the following, the multiplication of two arbitrary $GF(2^m)$ elements represented in the PB, based on the vector by matrix method, is first briefly reviewed. This is followed by reviewing efficient ways for the hardware realizations of the fixed multiplication of an arbitrary $GF(2^m)$ element by α^q, where q is a positive integer. After this, a quick review over previous work on PB multiplication is presented.

2.9.1.1 Multiplication of Two Arbitrary $GF(2^m)$ Elements Represented in the PB

This section, reviews the multiplication of two arbitrary $GF(2^m)$ elements represented in the PB based on the vector by matrix method. The formulations presented in this section are utilized in Section 6.2.1 to accomplish the multiplication of an arbitrary $GF(2^m)$ element by the constant α^{m-1}. Here, α is a root of the irreducible polynomial $p(x) = x^m + \sum_{i=1}^{\omega-2} x^{t_i} + 1$ which generates the field $GF(2^m)$. $\omega = H(p(x))$ is the Hamming weight of the field polynomial, denoting the number of nonzero terms in $p(x)$. First, the following notations are defined.

Definition 2.9.2 *Define* $\mathbf{v}[\uparrow i]$ *and* $\mathbf{v}[\downarrow i]$ *to be the operations of up and down i-bit shifts, respectively, of a given m-bit vertical vector* $\mathbf{v} = \left[\begin{array}{ccc} v_0 & \dots & v_{m-1} \end{array}\right]^T$, *where the emptied positions are filled with zeros (T denotes vector transposition). That is,*

$$v[\downarrow i] = \left[\begin{array}{cccccc} 0 & \dots & 0 & v_0 & \dots & v_{m-i-1} \end{array}\right]^T$$

and

$$v\left[\uparrow i\right] = \begin{bmatrix} v_i & \dots & v_{m-1} & 0 & \dots & 0 \end{bmatrix}^T.$$

Let $C = (c_{m-1}, \dots, c_0)$ denotes the result of multiplying two arbitrary $GF\left(2^m\right)$ elements $A = (a_{m-1}, \dots, a_0)$ and $B = (b_{m-1}, \dots, b_0)$, represented in the PB. Therefore, the m binary coordinates of $C = AB \bmod p\left(\alpha\right)$, represented by the vertical m-bit vector $\mathbf{c} = \begin{bmatrix} c_0 & \dots & c_{m-1} \end{bmatrix}^T$, are obtained as follows [75]

$$\mathbf{c} = \mathbf{d} + \sum_{j=0}^{\omega-2} \mathbf{e}' \left[\downarrow t_j\right], \tag{2.1}$$

where

$$\mathbf{d} = \begin{bmatrix} a_0 & 0 & \dots & 0 \\ a_1 & a_0 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ a_{m-1} & a_{m-2} & \dots & a_0 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_{m-1} \end{bmatrix}, \tag{2.2}$$

$$\mathbf{e}' = \sum_{i=0}^{n-1} \mathbf{e}\left[\uparrow l_i\right],$$

and

$$\mathbf{e} = \begin{bmatrix} 0 & a_{m-1} & \dots & a_2 & a_1 \\ 0 & 0 & \dots & a_3 & a_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & 0 & a_{m-1} \\ 0 & 0 & \dots & 0 & 0 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_{m-2} \\ b_{m-1} \end{bmatrix}. \tag{2.3}$$

Here, $t_0 = 0$, n is the number of nonzero entries in column zero of the $(m-1) \times m$ binary reduction matrix $\mathbf{Q}$ [72], and l_i denotes the row location of the i-th nonzero entry in this column, $0 \le i < n$. The following are some remarks on the values of n and l_i.

Remark 2.9.3 *Let* $p\left(x\right) = x^m + \sum_{i=1}^{\omega-2} x^{t_i} + 1$ *be the generator irreducible polynomial of* $GF\left(2^m\right)$ *with* ω *nonzero elements, then [33, 75]:*

- $l_0 = 0$ *regardless of the structure of* $p\left(x\right)$*.*
- *If* $p\left(x\right)$ *is a trinomial of the form* $x^m + x + 1$*, then:* $n = 1$ *and* $l_0 = 0$*.*
- *If* $p\left(x\right)$ *is a trinomial of the form* $x^m + x^{t_1} + 1$ *with* $1 < t_1 \le \frac{m+1}{2}$*, then:* $n = 2$ *with* $l_0 = 0$ *and* $l_1 = m - t_1$*.*
- *If* $p\left(x\right)$ *is a general irreducible polynomial with* $t_{\omega-2} \le \frac{m+1}{2}$*, then:* $n = \omega - 1 - \left\lfloor \frac{1}{t_1} \right\rfloor$*,* $l_0 = 0$*, and* $l_i = m - t_i$ *for* $1 \le i \le \omega - 2 - \left\lfloor \frac{1}{t_1} \right\rfloor$*.*

2.9.1.2 Efficient Multiplication of a $GF(2^m)$ Element Represented in the PB by α^q

It is noted that, one can obtain a general formulation for the multiplication of an arbitrary element $A = (a_{m-1}, \ldots, a_0) \in GF(2^m)$, represented in the PB, by α^q, using (2.1), (2.2), and (2.3) (see Section 6.2.1). However, this section lists some conditions, originally presented in [81] and [57], for the efficient hardware realization of such constant field multiplication. Here, q is a positive integer and α is the root of the field's irreducible polynomial $p(x) = x^m + \sum_{i=1}^{\omega-2} x^{t_i} + 1$ with ω nonzero terms.

Theorem 2.9.4 *[81] Assume $p(x) = x^m + \sum_{i=1}^{\omega-2} x^{t_i} + 1$ is the field irreducible polynomial which defines $GF(2^m)$. Let α denotes the root of $p(x)$. Therefore, for $q < m - t_{\omega-2}$, the coordinates of α^{m+q} are obtained as follows*

$$\alpha^{m+q} \; mod \; p(\alpha) = \left(\sum_{i=1}^{\omega-2} \alpha^{t_i} + 1 \right) \alpha^q. \tag{2.4}$$

Theorem 2.9.5 *[57] Assume $p(x) = x^m + \sum_{i=1}^{\omega-2} x^{t_i} + 1$ is the field irreducible polynomial which defines $GF(2^m)$. Denote by α the root of $p(x)$. Let $A = (a_{m-1}, \ldots, a_0)$ be an arbitrary $GF(2^m)$ element represented in the PB. Therefore, for $q \leq m - t_{\omega-2}$, the coordinates of $A\alpha^q \; mod \; p(\alpha)$ are obtained in a single step using $q(\omega - 2)$ two-inputs XOR gates with a propagation delay equivalent to $\lceil \log_2 (q+1) \rceil$ XOR gate delays, as follows:*

$$\begin{aligned} A\alpha^q \; mod \; p(\alpha) &= \sum_{i=0}^{m-1} a_i \alpha^{i+q} \; mod \; p(\alpha) \\ &= \sum_{i=0}^{m-q-1} a_i \alpha^{i+q} + \\ &\quad \sum_{i=m-q}^{m-1} a_i \left(\sum_{j=1}^{\omega-2} \alpha^{t_j} + 1 \right) \alpha^{i-(m-q)}. \end{aligned} \tag{2.5}$$

2.9.1.3 Previous Work on PB Multiplication

In general, the different proposed designs for implementing PB multiplication fall under one of the two categories of parallel and serial computations. For achieving high throughput, the parallel implementation is used where all the output bits of the multiplication are generated in a single clock cycle [19, 62, 84, 44, 72, 27, 22]. For achieving low space complexity, digit-level serial computations are considered. In digit-level serial multiplication schemes, the space complexity is reduced at the expense of increasing the number of clock cycles required for generating the m output bits (computational latency) to $k = \left\lceil \frac{m}{d} \right\rceil$ clock cycles (in general), where d is the digit size [78, 20, 46, 75, 79, 50, 66].

In a digit-level serial implementation, the multiplication input / output bits are entered / generated either in parallel, or serially in the order of one digit per a clock cycle. For example, digit-level serial-in-parallel-out (DL-SIPO) multipliers generate the output bits in parallel after k clock cycles [20]. In this DL-SIPO scheme, one input is loaded in parallel (in advance to computations), while the other input enters serially one digit per a clock cycle during computations. The serial input of the DL-SIPO multiplier enters in either a most-significant-digit first (MSD) or least-significant-digit first (LSD) order. Parallel-in-serial-out is another digit-level multiplication scheme (DL-PISO), in which both inputs are preloaded in parallel in advance to computations [75]. After this, the output digits of the DL-PISO multiplier are generated over k clock cycles, one digit per a clock cycle. A third digit-level multiplication scheme is known as parallel-in-parallel-out (PIPO) requires preloading of both inputs in advance to computations. The output of the PIPO multiplier is generated in parallel after a number of clock cycles from inputs preloading. For example, the PIPO PB multiplication architecture presented in [50] has a latency of $2t_{\omega-2} + 1$ clock cycles to generate the m output bits in parallel, where $t_{\omega-2}$ denotes the second highest nonzero term of the field irreducible polynomial $p(x) = x^m + \sum_{i=1}^{\omega-2} x^{t_i} + 1$ with ω nonzero terms. The authors of [50] show that their serial PIPO PB multiplier offers the lowest latency for cases where $m \geq 2t_{\omega-2} - 1$, however, the corresponding space complexity is quadratic in m.

In addition, serial-serial finite field multipliers with two serially-entered inputs and serial output have also been proposed. For example, in 1992, the authors of [46] presented a bit-level most-significan-bit-first (MSB) serial-serial PB multiplier which generates the m output bits serially over $2m$ clock cycles. In [46], the inputs to the multiplier enter serially bit-by-bit, starting with the MSB, over the first m clock cycles. After reading the serial inputs, the m output bits are then generated serially, one bit per a clock cycle, starting with the MSB. In 2009, the authors of [14] proposed a generic serial-serial multiplication/reduction architecture for $GF(q)$, where q can be a prime p, a power of a prime p^m, and where it is possible to have $p = 2$. For the case of $GF(2^m)$, which is the focus of this thesis, the serial-serial multiplication/reduction scheme proposed in [14] reads both of its multiplication inputs serially, one digit at a time, in either least or most-significant first order. The final result is generated digit-by-digit without using any dedicated parallel-in-serial-out register, starting with the $(k + 1)$-th clock cycle, where an additional correction step is required in case of the least-significant first input order. Then, using the scheme in [14], all the m output bits are produced serially, after a total of $2k$ clock cycles. It is noted that, the serial-serial multiplier in [14] is not a dedicated multiplication scheme in the sense that it works for any irreducible polynomial by reading it as one of its inputs. In order to allow for scalability, and to make the field multiplication generic, the serial-serial multiplier in [14] requires additional multiplexers and storage Flip-Flops (FF), in addition to a number of control signals.

2.9.2 $GF(2^m)$ Multiplication in the GNB Representation

Although multiplication is realized more efficiently in hardware based on the PB representation [71]; however, NBs are considered advantageous for use in the hardware designs of binary extension fields' arithmetic [71] due to the free cost of squaring operations, which are implemented as cyclic shifts [5]. In particular, the special subset of Gaussian normal bases (GNBs) offer field operations with smaller area and time overhead compared to the general NB. Hence, GNBs are often used for efficient hardware implementations of field multiplication, for example see the IEEE standard [5] and the National institute of standards and technology (NIST) standard [12].

The original multiplication scheme under the NB representation has been proposed by Massey and Omura [61]. In this scheme, all the output bits of the product are computed, one bit at a time, through applying some function to different cyclic shifts of the two input elements. This scheme is referred to as bit-level (BL) parallel-in-serial-out (PISO) multiplication. This section, briefly reviews formulations for the BL-PISO multiplication of two $GF(2^m)$ elements represented in the GNB. Also, this section shows how the multiplication of a field element by the normal element β is accomplished. In addition, a brief summary about existing GNB multiplication is given.

The following, starts by reviewing formulations for the BL-PISO GNB multiplication of two arbitrary $GF(2^m)$ elements.

2.9.2.1 Formulation for the BL-PISO $GF(2^m)$ Multiplication in the GNB Representation

Here, the formulations for accomplishing bit-level PISO multiplication of two $GF(2^m)$ elements represented in the GNB are presented. As mentioned earlier, by finding an element $\beta \in GF(2^m)$ such that $\mathcal{N} = \left\{\beta^{2^0}, \ldots, \beta^{2^{m-1}}\right\}$ is a basis, then, $\mathcal{N}$ is a NB and β is a normal element. For any $m > 1$ not divisible by 8, if there exists a prime number $p = mT + 1$ such that $\gcd({}^{mT}/_{k}, m) = 1$ where $2^k \equiv 1 \,(\mathrm{mod}\, p)$, then, $\mathcal{N}$ is a Gaussian normal basis (GNB) of type T, where T is an even integer if m is odd. Any element $A \in GF(2^m)$ can be represented w.r.t the GNB as $A = \sum_{i=0}^{m-1} a_i \beta^{2^i} = (a_0, \ldots, a_{m-1})$, where $a_i \in \{0, 1\}$.

Let $P_A(V) = AV = (p_0, \ldots, p_{m-1})$ denotes the result of multiplying A by $V = (v_0, \ldots, v_{m-1})$. Then, by using the following formulation, one obtains the l-th coordinate of $P_A(V)$, for $0 \le l <$

m [70]

$$p_l = a_l v_{((l+1))} + \sum_{i=1}^{m-1} a_{((l+i))} \left(\sum_{j=1}^{T} v_{((l+R[i,j]))} \right), \tag{2.6}$$

where $((q)) = q \bmod m$ and $0 \leq R[i,j] < m$, for $1 \leq i < m$ and $1 \leq j \leq T$, is an integer entry of an $(m-1) \times T$ matrix $\mathbf{R}$ which corresponds to the position of the j-th 1 in the i-th row of the GNB's multiplication matrix $\mathbf{M}$ [70]. This scheme for computing the l-th coordinate of the field multiplication requires m AND gates and at most $(m-1)\,T$ XOR gates, with a propagation delay of $T_A + (\lceil \log_2 m \rceil + \lceil \log_2 T \rceil)\, T_X$ [16], where T_A and T_X denote the propagation delay in a two-inputs AND gate and a two-inputs XOR gate, respectively.

The following section, shows how the GNB multiplication by the normal element β is accomplished.

2.9.2.2 Multiplication by the Normal Element β

Here, the formulation for accomplishing field multiplication of an arbitrary $GF(2^m)$ element $V = (v_0, \ldots, v_{m-1})$ represented in the Gaussian normal basis $\left\{\beta, \ldots, \beta^{2^{m-1}}\right\}$ of type T by the normal element $\beta = (1, 0, \ldots, 0)$ is presented. By substituting for $(a_0, \ldots, a_{m-1})$ with $(1, 0, \ldots, 0)$ in (2.6), and considering all values of $l = 0, \ldots, m-1$, one obtains [70]

$$P_\beta(V) = v_1 \beta + \sum_{i=1}^{m-1} \left(\sum_{j=1}^{T} v_{((i+R[m-i,j]))} \right) \beta^{2^i}, \tag{2.7}$$

which requires at most $(m-1)(T-1)$ XOR gates, with a propagation delay of $\lceil \log_2 T \rceil\, T_X$. It is noted that, one can reduce the number of XOR gates required for realizing (2.6) or (2.7) by a value $\triangle_X$ through applying signal reuse techniques (see [69, 25] for example), where $\triangle_X$ is obtained through simulation.

2.9.2.3 Previous Work on GNB Multiplication

Massey and Omura [61] proposed the original scheme for multiplication in the NB representation. After this, a number of designs were proposed in an attempt to optimize the throughput and / or space complexities of the NB multiplier [86, 47, 55, 71, 73, 13]. Generally, the different proposed designs can be divided into three categories of parallel, bit-level, and digit-level computations. For high throughput usage, the parallel implementation generates all the output bits of the multiplication in one clock cycle [86, 47, 55, 71]. In $GF(2^m)$, this is achieved by a gate complexity which is quadratic in m. For area critical applications, bit-level schemes are adopted [86, 36, 13]. In this scheme, the space complexity is generally proportional to m, while the multiplication process requires m clock cycles to generate the final output. To trade-off between space and throughput, digit-level multipliers are deployed [73, 37]. In a digit-level scheme, the space complexity is traded-off with the number of required clock cycles in such a way that d-bits, $2 \leq d < m$, are processed in parallel during each one of the $k = \left\lceil \frac{m}{d} \right\rceil$ clock cycles of computations.

Similar to the PB digit-level multipliers, there are three schemes in terms of types of inputs and output for the digit-level (DL) GNB multipliers. The first scheme of serial multipliers is the parallel-in-parallel-out (PIPO) [38, 13, 73]. In this scheme, the inputs are preloaded to the input registers first, and then, the m output bits are produced in parallel after k clock cycles.

In the remaining two schemes, one or both input(s) / output are fed / generated serially during each iteration of computations, where the serial input(s) / output follow either a least significant digit first (LSD), or a most significant digit first (MSD) order. The second scheme is the serial-in-parallel-out (SIPO) [20, 36]. There are two variants of this scheme, one with only one serial input [20], while the other has two serial inputs [36]. For clarity of reference, the two serial inputs variant is denoted as fully-serial-in-parallel-out (FSIPO). Both of the SIPO and the FSIPO multipliers generate the m output bits in parallel after k clock cycles. The SIPO requires to preload one of its inputs in advance to computations, while the other input enters the multiplier during computations. On the other hand, a FSIPO multiplier does not require any preloading of the operands, since it reads both inputs as computations are carried out.

The third scheme is the parallel-in-serial-out (PISO), in which the two operands are preloaded into the input registers before the computation starts, followed by generating the k output digits serially, one digit per a clock cycle [61, 37, 76].

By combining one DL-PISO and one DL-SIPO architectures, a DL-PIPO hybrid-double GNB multiplier has been recently proposed by the authors of [16], which performs two field multiplications using the same latency required for a single field multiplication (i.e. k iterations). It is noted that the authors of [16] have shown the hybrid-double multiplier to be useful for applications where two dependent field multiplications are involved, such as double exponentiation.

2.10 Exponentiation and Inverse over $GF(2^m)$

Field exponentiation and inversion, are usually realized in the form of repeated rounds of "square and multiply" operations [5]. Squaring is the operation of multiplying an element by itself. Field exponentiation of an element A, say A^e, is computed as [42]:

$$A^e = \Pi_{i=0}^{m-1} A^{e_i 2^i},$$

where the integer exponent e, $2 \leq e < 2^m - 1$, is represented in its radix two expansion as $\sum_{i=0}^{m-1} e_i 2^i$ with $e_i \in \{0, 1\}$ for all $i = 0, 1, \ldots, m-1$. Field inversion is a special case of the exponentiation, in which the exponent has a fixed value $e = 2^m - 2$, according to Fermat's Little Theorem (FLT) [30]. As it is mentioned earlier, the NB representation requires larger space overhead, compared to the PB, in order to realize field multiplications in hardware. However, NBs offer free of cost squaring operations, which is not the case for PBs. In NB representation, for any $A = (a_0, \ldots, a_{m-1}) \in GF(2^m)$, one simply obtains $A^2 = (a_{m-1}, a_0 \ldots, a_{m-2})$. In hardware, this is realized as a simple right cyclic shift. Therefore, many hardware designs favor using NBs for implementing exponentiation and inversion over PBs. More specifically, the subclass of Gaussian normal basis representation (GNB) which offers more efficient hardware implementations for the field multiplications than the general NBs [5], is usually deployed for exponentiation and inversion.

The next section introduces trace mappings of $GF(2^m)$ elements.

2.11 Trace Mapping

Trace, is a mapping, which maps a $GF(2^m)$ element, say A, to the ground field $GF(2)$, and is denoted by $Tr(A)$. The Trace of the element A is computed using the following formulation [5]:

$$Tr(A) = \sum_{i=0}^{m-1} A^{2^i}. \tag{2.8}$$

In the NB representation, (2.8) is reduced to the modulo-2 sum of the coordinates of A, that is [5]:

$$Tr(A) = \sum_{i=0}^{m-1} a_i.$$

On the other hand, in the PB representation, (2.8) is computed in the form of an inner product of the row vector representing A with the constant column vector $\tau = (\tau_{m-1}, \ldots, \tau_0)$, as follows [5]:

$$Tr(A) = \sum_{i=0}^{m-1} a_i \tau_i.$$

The coordinates of the constant vector τ are precomputed as $\tau_i = Tr\left(\alpha^i\right)$, for $0 \leq i < m$, with α representing the root of the defining irreducible polynomial $p(x)$ of $GF(2^m)$.

2.12 Welch-Gong (WG) Stream Ciphers

2.12.1 Stream Ciphers

Stream ciphers are symmetric key cryptosystems which are used for providing privacy through applying encryption and decryption mechanisms to a given message's text. Stream ciphers are attractive for implementing protection in the wireless air-link domain, due to the individual processing of the input message digits, which results in preventing error propagation at the receiving end. For example, stream ciphers are used in different wireless communications applications, such as, blue-tooth [8], network protocols (WEP and WPA) [43], and 3GPP Long Term Evolution (LTE) security suite [11, 7]. To accomplish individual processing of the input message's digits, stream ciphers encrypt (or decrypt) an input message by bit-wise XORing the corresponding bits of the message with a generated key-stream bits, bit by bit, where the key-stream is generated by means of a Pseudo random sequence generator (PRSG). Figure 2.1 presents two entities communicating over an insecure channel where a stream cipher is used for accomplishing privacy of transmitted data.

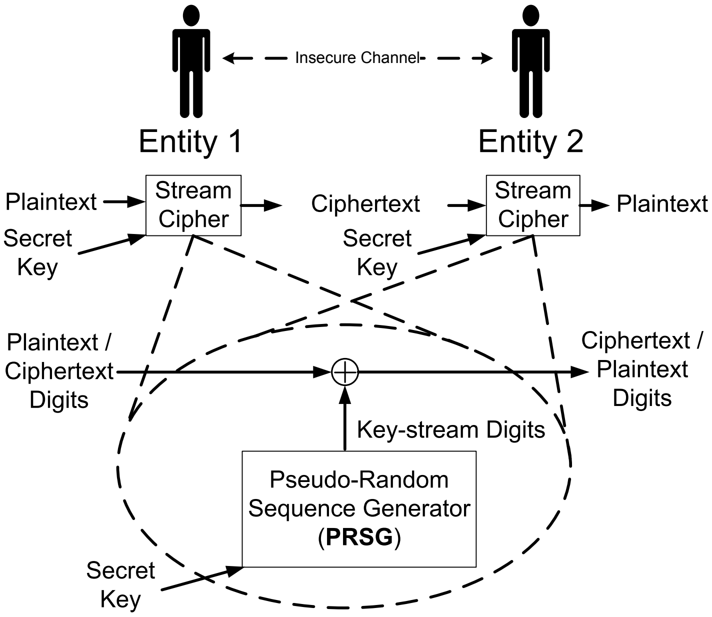


Figure 2.1: A stream cipher is used for providing privacy over an insecure channel between two communicating entities.

2.12.2 WG Stream Ciphers

The Welch-Gong (WG) stream ciphers, is a family of stream ciphers with good randomness properties [39, 68, 67, 24]. The randomness properties provided by the member ciphers of this family are proved mathematically, which include long period, balanced 0-1 distribution, ideal tuple distribution, exact linear complexity, cross correlation with an m-sequence has only three values, delta like autocorrelation functions, and high non linearity, for which no other existing ciphers could provide [40, 68].

2.12.2.1 A General Block Diagram

Figure 2.2 presents a block diagram showing an architecture of a general WG stream cipher. As it is shown in this figure, a WG stream cipher is built from a Finite state machine (FSM), a Linear feedback shift register (LFSR) which consists of l elements from the field $GF(2^m)$, and a WG transform ($WGTm$). Hence, the WG cipher is denoted by either WG(m, l) or WG-m. The FSM controls the operation of the cipher. The linear feedback function, which is represented by the LFSR's characteristic polynomial $C(Z)$ in the figure, is primitive over $GF(2^m)$, and therefore, the LFSR generates m-sequences having periods of $2^{ml} - 1$. The output of the LFSR, which is taken from the leftmost cell, is filtered by an m-bit WG transform $WGTm$. Notice that, the LFSR output might first go through decimation (that is exponentiation) before entering the transform. The WG transform consists of a permutation module ($WGPm$) followed by a trace mapping.

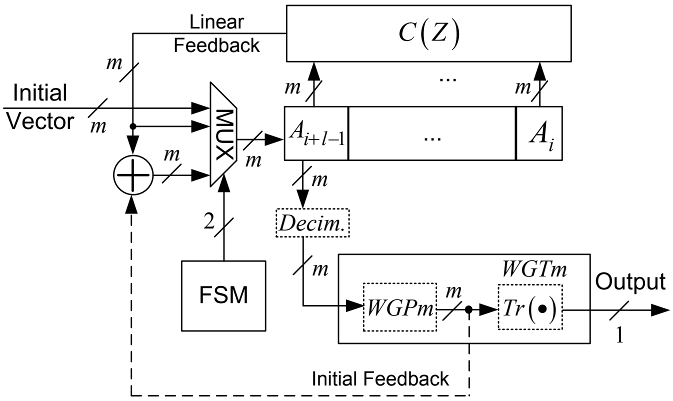


Figure 2.2: A general block diagram of a WG stream cipher.

2.12.2.2 Phases of Operation

There are three phases of operation in a WG stream cipher: loading phase, initialization phase, and run phase. During the loading phase, which takes l clock cycles to complete, the initial state is written to the cells of the LFSR, where the only input to the LFSR is "Initial Vector" in figure 2.2. Then, the initialization of the cipher starts and continues for $2l$ clock cycles, during which the input to the LFSR is the bitwise XOR of the "Linear Feedback" and "Initial Feedback" signals in figure 2.2. After this, the cipher enters the run phase, where a single key-stream bit is generated at each clock cycle. The only input to the LFSR during the run phase is the "Linear Feedback" signal.

2.12.2.3 WG$(29, 11)$ and WG-16

The eSTREAM project [6] is the most significant effort for finding secure stream ciphers [67]. The WG$(29, 11)$ [39] is a stream cipher submitted to the hardware profile of phase 2 of this project. The WG$(29, 11)$ offers the proved randomness properties of the WG family of ciphers [40, 39, 68, 24]. The two attacks [88, 77] were launched on WG$(29, 11)$ during this project. However, it is noted that the revised version of the cipher [68] does not suffer the chosen IV (Initial Value) attack in [88, 64]. Also, as per design, the number of key-stream bits per a single key/IV pair is strictly less than the number of key-stream bits required to perform a linear span attack introduced in [77], [68].

In the literature, there is a number of proposed WG$(29, 11)$ hardware designs [39, 67, 68, 56]. The original submission uses normal basis (NB) representation [39] and hence all of presented designs until now have used the NB representation [39, 68, 56, 58]. The authors of [39] adopt a direct design using computation in the Optimal normal basis (ONB), which requires 7 multiplications and an inversion over $GF\left(2^{29}\right)$. The inversion using Itoh-Tsujii algorithm requires $(\lfloor \log_2 (28) \rfloor + H(28) - 1) = 4 + 3 - 1 = 6$ multiplications and 28 squarings in $GF\left(2^{29}\right)$, where $H(28)$ denotes the Hamming weight of 28 [45]. In [68], the authors replaced the inversion operation with a computation of the power $2^k - 1$ which requires 4 multiplications for $k = \left\lceil \frac{29}{3} \right\rceil = 10$ and reduced the other 7 multiplications of the WG transformation in [39] by one through signal reuse. In [56], the author uses a look-up table based approach which uses 2^{29} bits of ROM. In [58], the authors propose a multiple-bit output version of the WG cipher, called MOWG. The MOWG reduces the hardware cost through signal reuse by removing one multiplier from the WG permutation in [68], while it generates $d \leq 17$ output bits. Furthermore, [58] improves the hardware cost and throughput of the cipher through pipelining with reuse techniques. The keystream sequences generated by the MOWG cipher possess many of the WG keystream randomness properties [58].

Another initiative for designing secure stream ciphers is the LTE mobile technology. LTE is being established as the fourth generation (4G) mobile technology, where a flat all Internet Protocol infrastructure has been adopted [34]. This has changed the threat model of the 4G mobile domain to include the security issues which are applied to the IP-networks [34]. Accordingly, there is a continuous effort demonstrated by the security specification group of the third generation partnership project (3GPP-TSG) [1] to address these security threats [34]. The cipher suite of 4G LTE consists of two stream ciphers, SNOW 3G and ZUC, and the block cipher AES in the counter mode [11, 7]. It is noted that the randomness of the key-streams generated by the 4G LTE cryptographic algorithms is hard to analyze and, more importantly, some weaknesses concerning these ciphers have already been discovered [87, 21]. Furthermore, some security flaws in the LTE integrity protocols have been recently recognized [90]. The authors of [34] propose confidentiality and integrity protection schemes for securing the 4G network domain against the attack in [90]. These schemes are based on the WG-16 stream cipher. The WG-16 offers the proved randomness properties of the WG family of ciphers [34]. In addition, it is secure and resists to all known attacks [34]. The only WG-16 hardware design, which uses NB, is presented in [35]. This design is based on composite field arithmetic and properties of the trace function in the tower field representation.

2.12.2.4 Parameters of the WG$(29, 11)$

The permutation for the WG$(29, 11)$ is

$$\begin{aligned} WGP29 = 1 \oplus Y \oplus Y^{2^{10}+1} \oplus Y^{2^{20}+2^{10}+1} \oplus \\ Y^{2^{20}-2^{10}+1} \oplus Y^{2^{20}+2^{10}-1}, \end{aligned} \tag{2.9}$$

where $Y = 1 \oplus A_{i+10}$ and A_{i+10} is the LFSR's output. The WG transform is given as follows [39, 68, 58]

$$WGT29 = Tr\left(WGP29\right). \tag{2.10}$$

The linear feedback characteristic polynomial of the WG$(29, 11)$

$$C(Z) = Z^{11} \oplus Z^{10} \oplus Z^{9} \oplus Z^{6} \oplus Z^{3} \oplus Z \oplus \beta, \tag{2.11}$$

is a primitive polynomial of degree 11 over $GF\left(2^{29}\right)$, where $\beta = \alpha^{464730077}$ is the generator of the Type-II Optimal NB (ONB-II, that is GNB of type 2) and α is a root of the defining polynomial of $GF\left(2^{29}\right)$ given by [68]

$$\begin{aligned} g\left(x\right) = x^{29} + x^{28} + x^{24} + x^{21} + x^{20} + x^{19} + x^{18} + x^{17} \\ + x^{14} + x^{12} + x^{11} + x^{10} + x^{7} + x^{6} + x^{4} + x + 1. \end{aligned} \tag{2.12}$$

2.12.2.5 Parameters of the WG-16

The WG-16 permutation is [34]

$$\begin{aligned} WGP16 =& 1 \oplus Y \oplus Y^{2^{11}+1} \oplus Y^{2^{11}+2^{6}+1} \\ & \oplus Y^{-2^{11}+2^{6}+1} \oplus Y^{2^{11}+2^{6}-1}, \end{aligned} \tag{2.13}$$

where $Y = (A_{i+31})^{1057} \oplus 1$ and A_{i+31} is the output of the LFSR. In [35], $WGP16$ is computed as

$$1 \oplus Y \oplus Y^{2^{11}+1} \oplus Y^{2^{11}(2^{11}-1)+1} \oplus Y^{2^{6}} \left(Y^{2^{11}+1} \oplus Y^{2^{11}-1} \right), \tag{2.14}$$

where

$$Y^{2^{11}-1} = Y^{\left((1+2)\left(1+2^{2}\right)+2^{4}\right)\left(1+2^{5}\right)+2^{10}}.$$

It is noted that (2.14) requires 10 multiplications (including 2 for computing $(A_{i+31})^{1057}$). The WG transform is $WGT16 = Tr(WGP16)$. The characteristic polynomial of the WG-16's LFSR[2] is [34]

$$C(Z) = Z^{32} \oplus Z^{31} \oplus Z^{22} \oplus Z^{9} \oplus \omega^{11} \tag{2.15}$$

which is primitive over $GF\left(2^{16}\right)$, where ω is the root of the $GF\left(2^{16}\right)$'s field polynomial

$$g(x) = x^{16} + x^{5} + x^{3} + x^{2} + 1. \tag{2.16}$$

---

[2] For the field polynomial (2.16), the multiplication with the constant ω^{11} in (2.15) requires only 33 XOR gates and a delay of $2T_X$.

Chapter 3

Implementations of the WG Stream Ciphers Using ONB-II

In this chapter, a novel method for computing the trace of a product of two field elements is presented, when the representation is the type-II ONB. Also, two designs are proposed. One for the MOWG$(29, 11, 17)$ cipher (where 29 corresponds to $GF\left(2^{29}\right)$, 11 is the number of stages in the LFSR, and 17 is the number of output bits) and the other one for the WG$(29, 11)$ cipher (which was initially proposed in [39]), demonstrated by ASIC and FPGA implementations. The proposed designs optimize the area by reducing the number of multiplications in the MOWG/WG transforms. This is done through signal reuse for the MOWG$(29, 11, 17)$ and through utilizing the new trace properties for the WG$(29, 11)$. The ASIC and FPGA implementations of the proposed WG$(29, 11)$ design show significant area and power consumption reduction and an improved speed compared to [68]. Notice that, in an FPGA implementation one has a predetermined space resources. In this context, reducing area consumption in an FPGA implementation is in terms of decreasing the number of used look-up tables. This in return would leave more resources for implementing other modules on the FPGA chip.

Throughout this chapter, $\oplus$ represents the bit-wise addition operator (XOR) in $GF\left(2^m\right)$. $A^{2^p} = A \gg p$ and $A^{2^{-p}} = A \ll p$, represent the right and left cyclic shift, respectively, of the coordinates of $A = (a_0, \ldots, a_{m-1}) \in GF\left(2^m\right)$, w.r.t NB, p-times. In the NB representation, the addition of $1 = (1, \ldots, 1) \in GF\left(2^m\right)$ to another $GF\left(2^m\right)$ element can be done by complementing the bits of that element. $C\left(Z\right) = Z^l \oplus \sum_{i=0}^{l-1} C_i Z^i$, $C_i \in GF\left(2^m\right)$ is the characteristic polynomial of an l-stages LFSR over $GF\left(2^m\right)$, from which the recurrence relation is obtained as

$$A_{j+l} = \sum_{i=0}^{l-1} C_i A_{i+j}, \tag{3.1}$$

where $j \geq 0$, $A_i \in GF(2^m)$, and $(A_0, A_1, \ldots, A_{l-1})$ is the initial state of the LFSR.

Also, throughout this chapter, the 29-bit WG transformation and permutation introduced in Chapter 2 are rewritten as follows

$$WGT29\left(A_{i+10} \oplus 1\right) = Tr\left(WGP29\left(A_{i+10} \oplus 1\right)\right), \tag{3.2}$$

and

$$\begin{aligned} WGP29(X) =& 1 \oplus X \oplus X^{r_1} \oplus X^{r_2} \oplus X^{r_3} \oplus X^{r_4} \\ =& \left(1 \oplus X \oplus X^{2^k+1} \oplus X^{2^{2k}+(2^k+1)} \oplus X^{2^k(2^k-1)+1} \oplus X^{2^{2k}+(2^k-1)}\right) \end{aligned} \tag{3.3}$$

where $r_1 = 2^k + 1$, $r_2 = 2^{2k} + 2^k + 1$, $r_3 = 2^{2k} - 2^k + 1$, $r_4 = 2^{2k} + 2^k - 1$, and $k = \left\lceil \frac{29}{3} \right\rceil$ [58].

It is noted that a version of this chapter appears in [31]. The chapter is organized as follows. Sections 3.1 and 3.2 presents the new hardware designs of the MOWG(29, 11, 17) cipher and the WG(29, 11) cipher, respectively. Results based on FPGA and ASIC implementations of the new designs are discussed in Section 3.3. Section 3.4 concludes the chapter.

3.1 Optimized Hardware Design of the MOWG(29, 11, 17) Cipher

This section presents a hardware design of the MOWG(29, 11, 17) cipher. In this design, the MOWG transform uses 7 multipliers, compared to 8 multipliers in [58]. Also, in an attempt to improve the overall speed of the cipher, the LFSR is reconstructed in order to remove the inverters from the critical paths during the run phase/initialization phase. In what follows, the reduced area MOWG transform design is first introduced, followed by presenting the LFSR and key initialization algorithm (KIA) changes for speed improvement. Then, the proposed architecture and finite state machine are discussed, and the section ends up by deriving formulations for the space and time complexities.

3.1.1 Reducing the Hardware Complexity of the MOWG Transformation

The hardware cost of the MOWG(29, 11, 17) cipher is dominated by its transform's field multipliers. Any decrease in the number of these multipliers would minimize the area of the overall cipher. This section presents the architecture of the MOWG transform, where the number of field multipliers is reduced by 1 through signal reuse, compared to [58].

The architecture of the proposed MOWG transform is shown in Figure 3.1. In this figure, $X = A_{i+10} + 1$ is the bit-wise complement of the LFSR's output, $r_1 = 2^k + 1$, $r_2 = 2^{2k} + 2^k + 1$, $r_3 = 2^{2k} - 2^k + 1$, $r_4 = 2^{2k} + 2^k - 1$, and $k = \left\lceil \frac{29}{3} \right\rceil = 10$. By taking $X^{2^{2k}}$ as a common factor of the exponent terms $2^{2k} + \left(2^k + 1\right)$ and $2^{2k} + \left(2^k - 1\right)$ in equation (3.3), the architecture in this figure can easily be obtained, where the WG permutation given by (3.3) is now computed as follows

$$WGP29 = \left(1 \oplus X \oplus X^{2^k+1} \oplus X^{2^k(2^k-1)+1} \oplus X^{2^{2k}} \left(X^{(2^k+1)} \oplus X^{(2^k-1)}\right)\right). \tag{3.4}$$

In the MOWG$(29, 11, 17)$, $k = 10$ and, hence, the signal X^{2^k-1} requires 4 multiplications and 4 squaring operations (which is free of cost in ONB) [58]. Also, in addition to the multiplication operations involved in computing the signal $X^{(2^k-1)}$, (3.4) requires three more multiplications to generate the signals X^{2^k+1}, $X^{2^k(2^k-1)+1}$, and $X^{2^{2k}} \left(X^{(2^k+1)} \oplus X^{(2^k-1)}\right)$. Therefore, the architecture of Figure 3.1 requires a total of 7 $GF\left(2^{29}\right)$ multiplications. The inverter symbol denoted by (1) in this figure requires 29 NOT gates to generate $X = A_{i+10} \oplus 1$ from the LFSR's output signal A_{i+10}. The signal $X \oplus X^{r_1} \oplus X^{r_2} \oplus X^{r_3} \oplus X^{r_4}$ is obtained as the addition in $GF(2^{29})$ of X, $X^{r_1} = X^{2^k+1}$, $X^{r_2} \oplus X^{r_4} = X^{2^{2k}} \left(X^{(2^k+1)} \oplus X^{(2^k-1)}\right)$, and $X^{r_3} = X^{2^k(2^k-1)+1}$. The signals X^{2^k} and $X^{2^{2k}}$ are obtained by right cyclic shifts of X, k and $2k$ times, respectively. X^{2^k+1} is generated by multiplying X with X^{2^k} in $GF\left(2^{29}\right)$. $X^{2^k(2^k-1)}$ is the right cyclic shift of $X^{(2^k-1)}$, k times, and $X^{2^k(2^k-1)+1}$ is generated by multiplying $X^{2^k(2^k-1)}$ with X in $GF\left(2^{29}\right)$. In Figure 3.1, the coordinates of the output of $X \oplus X^{r_1} \oplus X^{r_2} \oplus X^{r_3} \oplus X^{r_4}$ in $GF(2^{29})$ are complemented by the inverter symbol denoted by (2) to generate all 29 bits of the $WGP29$ function of (3.4), which forms the Initial Feedback. Seventeen bits of the $WGP29$ are the output of the MOWG$(29, 11, 17)$ in the run phase [58].

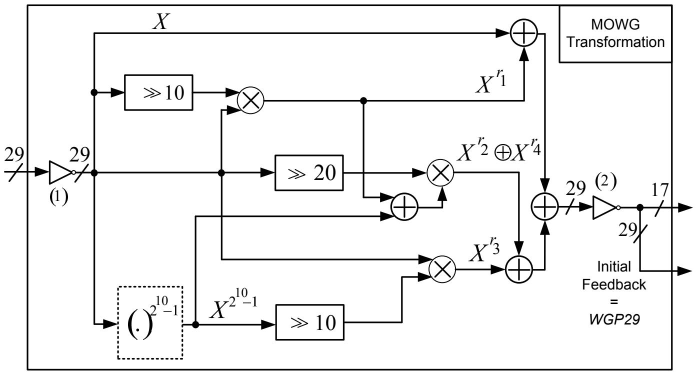


Figure 3.1: Proposed MOWG transformation.

3.1.2 Improving the Critical Path of the MOWG Transform

The time delay through the MOWG transform dominates the delay of the overall cipher (see Section 3.1.5.2). This section shows how to slightly reduce the delay through this transform.

This is accomplished by removing inverter (1), and by reallocating inverter (2) away from the critical paths of the run phase and key initialization phase. This reduces the delay of the critical path by an amount equivalent to the delay of two inverters. However, the MOWG transform delay is still the dominant, due to the delays of 5 serially connected field multipliers. First, the required mathematical formulation is derived, then required changes to the KIA algorithm are presented.

3.1.2.1 Formulation

During the key initialization phase and the run phase, inverter (1) in Figure 3.1 generates the complement of A_{i+10}. Notice that this cell holds the feedback from the LFSR during the run phase, and the bit-wise XOR of the LFSR feedback and the MOWG transform feedback during the key initialization phase. Therefore, to remove inverter (1), it requires the direct storage of the complement of these values in both phases. In other words, it is required to reconstruct the LFSR such that it generates a sequence $\underline{B} = \left\{B_i = 1 \oplus A_i,\ 0 \le i < 2^{319} - 1\right\}$, where $B_i \in GF\left(2^{29}\right)$ and $\{A_i\}$ is the sequence generated by (2.11) over $GF\left(2^{29}\right)$. Sequence $\underline{B}$ is referred to as the complement sequence of $\{A_i\}$. The following proposition shows how this is accomplished for an LFSR with a general feedback polynomial of degree l over $GF\left(2^m\right)$.

Proposition 3.1.1 *Let* $\underline{B}$ *be the complement sequence of a sequence* $\underline{A} = \left\{A_i,\ 0 \le i < 2^{ml} - 1\right\}$, *where* $A_i \in GF\left(2^m\right)$ *and* $\underline{A}$ *is generated by* (3.1). *Then,* $\underline{B}$ *is generated by the following recurrence relation*

$$B_{j+l} = \left(\sum_{i=0}^{l-1} C_i B_{i+j}\right) \oplus \left(\left(\sum_{i=0}^{l-1} C_i\right) \oplus 1\right), \tag{3.5}$$

where $j \ge 0$, *and the initial state of* $\underline{B}$ *is* $B_i = 1 \oplus A_i$, *for* $0 \le i \le l - 1$.

Proof By definition

$$B_{j+l} = A_{j+l} \oplus 1, \tag{3.6}$$

$j \ge 0$. Using (3.1) in (3.6), one gets $B_{j+l} = \sum_{i=0}^{l-1} C_i A_{i+j} \oplus 1$, and by noticing $2C_i = 0$ one obtains

$$\begin{aligned} B_{j+l} &= \sum_{i=0}^{l-1} C_i (A_{i+j} \oplus 1) \oplus \sum_{i=0}^{l-1} C_i \oplus 1 \\ &= \sum_{i=0}^{l-1} C_i B_{i+j} \oplus \sum_{i=0}^{l-1} C_i \oplus 1. \end{aligned}$$

Thus, the assertion is true.

By noticing that $X = 1 \oplus A_{i+10}$ in (3.4), then, from Proposition 3.1.1, one can see that X is B_{i+10}. Notice that the term $\left(\sum_{i=0}^{l-1} C_i\right) \oplus 1$ in (3.5) is a constant term. Hence, its addition in $GF\left(2^{29}\right)$ is realized with a number of NOT gates equal to its Hamming weight. For the LFSR of the MOWG(29, 11, 17), replacing the coefficients of (2.11) in (3.5) gives $\left(\sum_{i=0}^{l-1} C_i\right) \oplus 1 = \beta \oplus 1$, which has a Hamming weight equal to 28.

Inverter (2), on the other hand, realizes the addition of the field element 1 in (3.4). Notice that this addition of the term 1 can be implemented in different ways. One way is to add it to one of the terms X, X^{r_1}, $X^{r_2} \oplus X^{r_4}$, or X^{r_3} prior to the summation of these terms. Doing so would reallocate inverter (2) from its current position. However, it is required that this reallocation does not result in a delay higher than the current maximum delay of the MOWG transform. For this reason, the inverter is relocated to complement X before it is added to X^{r_1}. This is the path at the top of Figure 3.1, which has the lowest delay with only two $GF\left(2^{29}\right)$ adders between inverters (1) and (2).

The following section presents necessary changes required in the KIA algorithm of the MOWG(29, 11, 17) cipher.

3.1.2.2 Modified KIA Algorithm

Modifying the LFSR of MOWG(29, 11, 17) according to (3.5), requires its left most stage to hold the complement of the Initial Vector during the loading phase. Therefore, it is required to complement the Initial Vector input before it is loaded to the modified LFSR. This can easily be implemented by inserting 29 inverters at the multiplexer's input which receives the Initial Vector in Figure 2.2.

Next, the proposed architecture of the MOWG(29, 11, 17) cipher is presented.

3.1.3 Architecture

Here, the overall proposed architecture of the MOWG(29, 11, 17) cipher is presented, as shown in Figure 3.2. In this figure, a double-headed arrow, under a component, corresponds to a 29-bit register which is inserted for pipelining purposes (see Section 3.3.2 for more details). The Finite State Machine (FSM) controls the input to the LFSR for each phase of operation. In the same figure, due to the bit-wise complement operator denoted by (a), the LFSR receives the complemented Initial Vector during the loading phase. Hence, after 11 clock cycles, the initial state of this LFSR, $(B_0, B_1, \ldots, B_{10})$, is basically the complement of the initial state of the LFSR in Figure 2.2, i.e. $B_i = A_i \oplus 1$, $0 \leq i < 11$. When the key initialization phase starts, the bit-wise XOR of the Initial Feedback and the Linear Feedback applies to the input of the LFSR. Note that the Linear Feedback in Figure 3.2 is generated by (3.5), which is equivalent

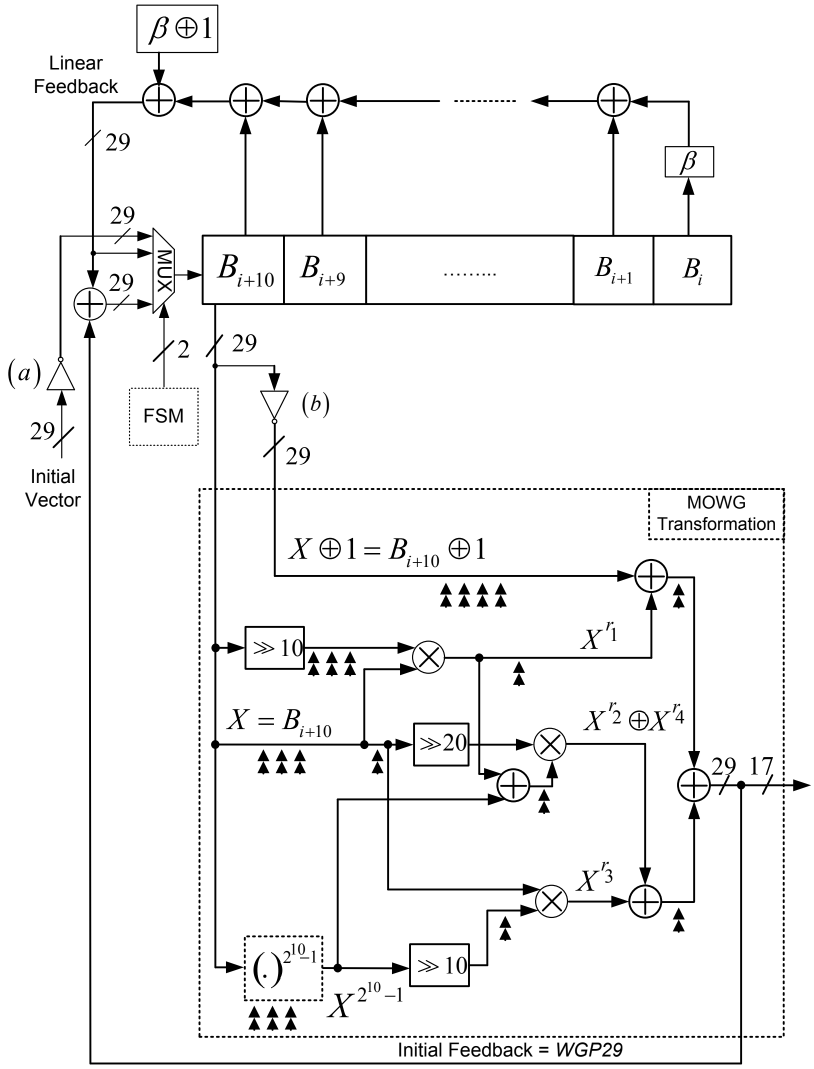


Figure 3.2: Proposed design of the MOWG(29, 11, 17) cipher.

to $B_i = A_i \oplus 1$, $11 \leq i < 33$ (complement of corresponding one in Figure 2.2). However, the Initial Feedback signal in Figure 3.2 has the same value as the one generated in Figure 3.1. This means that the input to the LFSR during the key initialization phase in Figure 3.2 is complemented w.r.t the one in Figure 2.2. Throughout the run phase, the only input to the LFSR is the Linear Feedback signal $B_i = A_i \oplus 1$, $33 \leq i < 2^{319} - 1$. This sets the MOWG transform of Figure 3.2 to generate the same key-stream bits of Figure 3.1. It is clear that the maximum delay of the MOWG transformation is reduced by an amount equivalent to the delay of two inverters, as compared to the one in Figure 3.1. The revised LFSR in Figure 3.2 has additional $H(\beta \oplus 1) = 28$ inverters, compared to Figure 2.2. This is due to the new constant term $\beta \oplus 1$ in the feedback polynomial.

The following section presents the finite state machine.

3.1.4 The Finite State Machine

This section exposes the architecture of the FSM and describes how it schedules the input to the LFSR throughout the three phases of operation.

Figure 3.3 shows the components of the FSM. The FSM has two inputs, namely clk and reset, 1-bit each, while there are two outputs denoted as op0 and op1. The reset input is pulled down before each run of the cipher. This forces the 11-bit one-hot counter to initialize to $(1, 0, \ldots, 0)$, i.e. output 0 is the only bit set to a high logic level. Also, when the reset signal is low, the 2-bit binary counter resets its state to $(0, 0)$. Due to the 1-bit Register connected to the AND gate at the reset input of the 11-bit one-hot counter, this counter starts incrementing one clock cycle after the reset signal gets pulled up. This assures that the 11-bit one-hot counter returns to its initial state after 11 clock cycles. Then, it triggers the 2-bit binary counter to increment which starts the initialization phase. The output of the 2-bit binary counter controls the cipher's phase of operation. This is done by generating the op0 and op1 signals according to Table 3.1. The op0 and op1 signals select one of the three inputs of the multiplexer in Figure 3.2 and connect it to the input of the LFSR, during each phase. It is noted that the loading phase takes 11 clock cycles, then starts the key initialization phase which takes 22 clock cycles, followed by the run phase. During the run phase, the clock inputs of the 11-bit

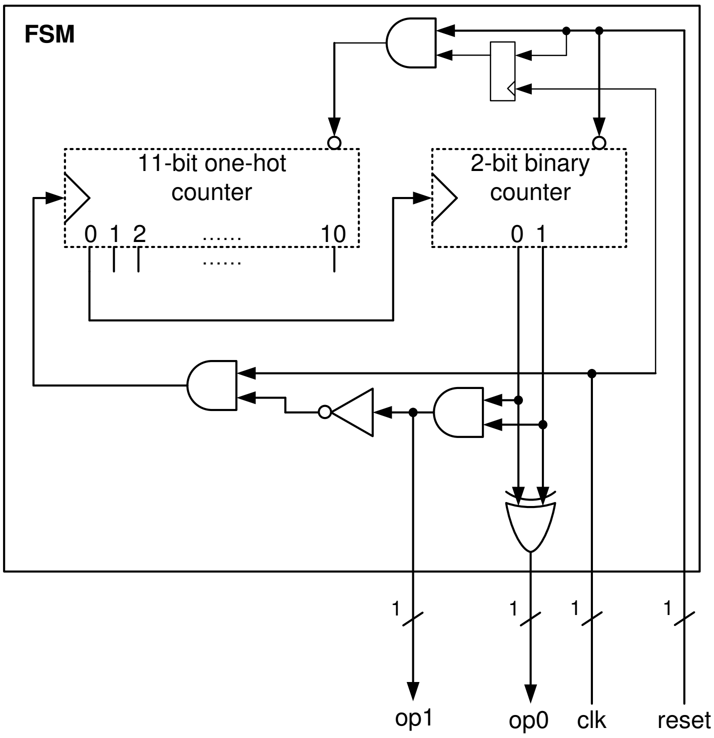


Figure 3.3: FSM of the MOWG.

2-bit counter		op1	op0	phase of operation
bit 1	bit 0			
0	0	0	0	Load Key and IV
1	0	0	1	Key Initialization
0	1	0	1	Key Initialization
1	1	1	0	Running Phase

Table 3.1: Phase of operation in the proposed MOWG as a function of the state of the 2-bit binary counter.

one-hot counter and the 2-bit binary counter become idle.

In what follows, space and time complexities of the proposed MOWG$(29, 11, 17)$ are studied.

3.1.5 Space and Time Complexities

This section provides the space and time complexities of the MOWG design in Figure 3.2.

3.1.5.1 Space Complexity

The space complexity is evaluated in terms of number of gates in each component, in order to obtain the overall hardware cost. Let N_R, N_A, N_X, N_O, and N_I denote the number of 1-bit Registers, AND gates, XOR gates, OR gates, and Inverters, respectively.

MOWG Transform The transform dominates the hardware complexity of the MOWG design, as it consists of 7 field multipliers and 4 $GF\left(2^{29}\right)$ adders. A $GF\left(2^{29}\right)$ adder requires 29 XOR gates. Also, the multiplier in [71] is used for implementation, which has 841 AND gates and 1218 XOR gates. Therefore, the total hardware cost of the transformation is as listed in Table 3.2.

LFSR The LFSR has 11-stages of 29-bit shift registers, and a feedback polynomial. The feedback polynomial is composed of 1 field multiplier (with a constant)[1], 5 $GF\left(2^{29}\right)$ additions, and $H(\beta \oplus 1) = 28$ Inverters. Therefore, the hardware complexity of the LFSR is as summarized in Table 3.2.

---

[1] A multiplication with a constant can be further optimized so that it contains few XOR gates.

Component	N_R	N_A	N_X	N_O	N_I
MOWG Transform	-	5887	8642	-	-
LFSR	319	841	1363	-	28
FSM (Figure 3.3)	14	3	1	-	1
29-bit 4-to-1 MUX	-	174	-	87	2

Table 3.2: Count of 1-bit registers and logic gates in the different components of the proposed MOWG design.

4-to-1 29-bit Multiplexer The 4-to-1 29-bit multiplexer is composed of a binary tree of three 2-to-1 29-bit multiplexers and 2 NOTs (selectors). Each 2-to-1 29-bit multiplexer is built from 29 parallel 2-to-1 1-bit multiplexers. A 2-to-1 one bit multiplexer consists of two AND gates and one OR gate. Therefore, the total cost of the 4-to-1 29-bit multiplexer is as summarized in Table 3.2.

FSM From Figure 3.3, there are 3 AND gates, 1 XOR gate and 1 Inverter in the FSM. The 11-bit one-hot counter is simply an 11-stages circular shift register with set/reset inputs having the output of the last shift register fed to the input of the first one. The 2-bit binary counter is built from two JK Flip Flops. The two inputs of the first FF are pulled to high logic and its output drives the two inputs of the second FF (one can also use D FF instead of the JK FF to design the 2-bit binary counter). Thus, one can find the total number of one-bit registers in the FSM as

$$N_R = 11 + 2 + 1 = 14.$$

Table 3.2 summarizes the number of gates in the FSM.

In addition to the above-mentioned components, the MOWG cipher contains two 29-bit bit-wise complement operators (inverter symbol (a) and inverter symbol (b) in Figure 3.2) and a $GF\left(2^{29}\right)$ adder (computing the bit-wise XOR of Initial Feedback signal and the Linear Feedback signal). Let N_O^{MOWG}, N_I^{MOWG}, N_R^{MOWG}, N_A^{MOWG}, and N_X^{MOWG} denote the number of OR gates, Inverters, 1-bit Registers, AND gates, and XOR gates in the MOWG of Figure 3.2, respectively. Therefore, by adding the corresponding number of gates in this $GF\left(2^{29}\right)$ adder and in inverter symbols (a) and (b) to the number of gates in the FSM, the 4-to-1 29-bit multiplexer, the LFSR, and the MOWG transform (see Table 3.2) one obtains

$$\begin{aligned} &N_O^{MOWG} = 87, \quad N_I^{MOWG} = 89, \quad N_R^{MOWG} = 333, \\ &N_A^{MOWG} = 6905, \quad N_X^{MOWG} = 10035. \end{aligned}$$

3.1.5.2 Time Complexity

Here, the formulation for the critical path delay of the MOWG cipher (Figure 3.2) is derived. There are three critical paths in the MOWG:

- Critical path of the LFSR.
- Critical path along the MOWG transformation during the key initialization phase.
- Critical path along the MOWG transformation during the run phase.

The LFSR's path has one multiplication and five finite field additions. This results in a propagation delay of

$$T_A + (1 + \lceil \log_2 (6) \rceil + \lceil \log_2 (29) \rceil)\, T_X = T_A + 9T_X, \tag{3.7}$$

where T_A and T_X denote the propagation delay of an AND and an XOR, respectively. The delay through a finite field multiplier is $T_A + (1 + \lceil \log_2 (29) \rceil)\, T_X$ [71]. On the other hand, the delays through the two MOWG transform paths have 5 multipliers in series, which corresponds to a delay of

$$5\,(T_A + 6T_X) = 5T_A + 30T_X. \tag{3.8}$$

From (3.7) and (3.8), it is clear that the longest path of the MOWG cipher passes through its transformation.

From Figure 3.2, the critical path of the proposed MOWG during the run phase includes the delays of a 29-bit Register, 5 field multipliers in series, and 3 $GF\left(2^{29}\right)$ adders. This results in the delay stated in (3.9):

$$T_{RunPh} = 5T_A + 33T_X + T_R, \tag{3.9}$$

where T_{RunPh} denotes the maximum time delay through the MOWG during the run phase. In the same figure, the critical path of the MOWG during the key initialization phase includes the delays of 4 $GF\left(2^{29}\right)$ adders, 5 field multipliers, a 29-bit Register, and a 4-to-1 29-bit multiplexer. Notice that the delay through the 4-to-1 29-bit multiplexer is equivalent to the delay through 2 2-to-1 1-bit multiplexers in series. This is equivalent to the sum of the delays through 2 AND gates, 2 OR gates, and 2 Inverters. Therefore, the delay of the MOWG during the key initialization phase is

$$T_{KIPh} = 7T_A + 34T_X + T_R + 2T_O + 2T_I \tag{3.10}$$

Comparing (3.9) and (3.10), it is clear that $T_{KIPh} > T_{RunPh}$.

3.2 Low Complexity WG Cipher

This section proposes a new design of the WG(29, 11). The proposed WG design considers Figure 3.2 with an added trace to the output of the $WGP29$ as the starting point for optimization. Properties of the trace function when the elements of $GF(2^m)$ are represented in ONB of type-II (which exists for $m = 29$ [52]) are first introduced. The proposed WG design utilizes these properties in order to minimize the hardware complexity of its transform. Note that the proposed design eliminates some necessary signals for the generation of the Initial Feedback, which is required to conduct the key initialization phase of the cipher. Missing of the Initial Feedback signal is recovered by introducing a serialized scheme to generate it. At the end of this section, the hardware and the time complexities of the new implementation are provided.

3.2.1 Properties of the Trace Function for Type-II ONB

This section presents a method for computing the trace of a multiplication of two field elements when the representation is in the type-II ONB. Also, two corollaries are deduced from the proposed method.

Fact 3.2.1 *[65] Let* $\{\beta, \beta^2, \beta^{2^2}, \ldots, \beta^{2^{m-1}}\}$ *be a type-II ONB for* $GF(2^m)$*. Then*

$$Tr(\beta^{2^i}) = 1, \;\; i = 0, 1, \cdots, m-1,$$

and

$$Tr(\beta^{2^i}\beta^{2^j}) = 0 \;\; \forall i \neq j; \;\; i, j = 0, 1, \cdots, m-1.$$

In other words, a type-II ONB is a self-dual basis. Thus Proposition 3.2.2 is achieved as follows.

Proposition 3.2.2 *In a type-II ONB, the trace of the field multiplication of any two* $GF(2^m)$ *elements* $A = (a_0, a_1, \ldots, a_{m-1})$ *and* $B = (b_0, b_1, \ldots, b_{m-1})$ *is computed as the inner product of A and B, that is:*

$$Tr(AB) = \sum_{i=0}^{m-1} a_i b_i. \tag{3.11}$$

Proof The proof is completed by considering the following derivation:

$$\begin{aligned} Tr(AB) &= Tr\Big(\sum_{i=0}^{m-1} a_i\beta^{2^i} \sum_{j=0}^{m-1} b_j\beta^{2^j}\Big) \\ &= \sum_{0 \le i,j < m} a_i b_j Tr(\beta^{2^i+2^j}) = \sum_{i=0}^{m-1} a_i b_i, \end{aligned}$$

where the last result is obtained using Fact 3.2.1.

Proposition 3.2.2 implies that the trace of a field multiplication of two elements represented in type-II ONB is easily implemented in hardware using m AND gates and $m-1$ XOR gates.

Corollary 3.2.3 *In type-II optimal normal basis, the two relations below are valid for any two elements A and B in* $GF(2^m)$

$$Tr(AB) = Tr((A \gg n)(B \gg n)) = \sum_{i=0}^{m-1} a_{i-n} b_{i-n}, \tag{3.12}$$

and

$$Tr(AB) = Tr((A \ll n)(B \ll n)) = \sum_{i=0}^{m-1} a_{i+n} b_{i+n}, \tag{3.13}$$

where n is a positive integer and the indices of a and b are computed modulo m.

Proof Let A and B be any two elements in $GF(2^m)$ and n an arbitrary positive integer. It is well known that

$$Tr\left(X^{2^{\pm n}}\right) = Tr(X)^{2^{\pm n}} = Tr(X),$$

for any $X \in GF(2^m)$. Therefore, by replacing X with AB one obtains

$$Tr(AB) = Tr\left(A^{2^{\pm n}} B^{2^{\pm n}}\right). \tag{3.14}$$

Using Proposition 3.2.2, the proof is completed by realizing that the squaring operation X^2, and the square root operation $X^{2^{-1}}$, are simply the right cyclic shift and the left cyclic shift of the coordinates of X (or AB) w.r.t the ONB, respectively.

According to Corollary 3.2.3, the trace of the field multiplication of any two elements A and B, represented in type-II ONB, does not change if an n-bit cyclic shift (left or right) is applied to both elements in the same direction.

Corollary 3.2.4 *Let C be a common factor of two or more* $GF(2^m)$ *elements AC, BC, ..., etc, then, the following relation holds:*

$$Tr(AC) + Tr(BC) + \cdots = \sum_{i=0}^{m-1} (a_i + b_i + \cdots) c_i. \tag{3.15}$$

Proof Let A, B, ..., etc, be any two or more arbitrary elements from the finite field $GF(2^m)$. Then,

$$\begin{aligned} Tr(AC) + Tr(BC) + \cdots &= Tr((A \oplus B \oplus \cdots) C) \\ &= \sum_{i=0}^{m-1} (a_i + b_i + \cdots) c_i, \end{aligned}$$

where the last result follows from Proposition 3.2.2, and $C \in GF(2^m)$.

The following section applies the new trace properties of this section in order to optimize the hardware implementation of the WG transform.

3.2.2 Optimizing the WG Transform's Hardware for the Run Phase

Here, it is shown how Proposition 3.2.2 and Corollaries 3.2.3 and 3.2.4 are used to further reduce the number of field multiplications in the WG transform in Figure 3.2 (with trace). Before proceeding, it is important to mention that by applying (3.11), one can generate the trace of the field multiplication of two elements A and B directly from A and B. However, the result of the multiplication operation, i.e. $C = AB$, will be lost. Therefore, it is important to apply (3.11) to the multiplication terms in (3.4) which are not used anywhere else. From Figure 3.2, the two signals $X^{r_2} \oplus X^{r_4}$ and X^{r_3} are used only as inputs to the trace function (after they are bit-wise XORed), while the signal X^{r_1} is required in generating $X^{r_2} \oplus X^{r_4}$. The first two signals are generated as follows

$$\begin{cases} X^{r_2} \oplus X^{r_4} = X^{2^{2k}}\left(X^{r_1} \oplus X^{2^k-1}\right), \\ X^{r_3} = XX^{2^k(2^k-1)}. \end{cases} \tag{3.16}$$

Therefore, applying the trace function to (3.16) one gets

$$\begin{cases} Tr\left(X^{r_2} \oplus X^{r_4}\right) = Tr\left(X^{2^{2k}}\left(X^{r_1} \oplus X^{2^k-1}\right)\right), \\ Tr\left(X^{r_3}\right) = Tr\left(XX^{2^k(2^k-1)}\right). \end{cases} \tag{3.17}$$

Using (3.17), the WG transformation becomes

$$WGT29 = Tr\left(1 \oplus X \oplus X^{r_1}\right) + Tr\left(XX^{2^k(2^k-1)}\right) + Tr\left(X^{2^{2k}}\left(X^{r_1} \oplus X^{2^k-1}\right)\right). \tag{3.18}$$

Applying a right cyclic shift of $2k$-stages to X and $X^{2^k(2^k-1)}$ in the term $Tr\left(XX^{2^k(2^k-1)}\right)$ of (3.18) does not change the value of the trace, i.e.

$$Tr\left(XX^{2^k(2^k-1)}\right) = Tr\left((X)^{2^{2k}}\left(X^{2^k(2^k-1)}\right)^{2^{2k}}\right). \tag{3.19}$$

Using (3.19) in (3.18) gives

$$WGT29 = Tr\left(1 \oplus X \oplus X^{r_1}\right) + Tr\left(X^{2^{2k}}X^{2^{3k}(2^k-1)}\right) + Tr\left(X^{2^{2k}}\left(X^{r_1} \oplus X^{2^k-1}\right)\right). \tag{3.20}$$

Taking $X^{2^{2k}}$ as a common factor in (3.20) one obtains

$$WGT29 = Tr\left(1 \oplus X \oplus X^{r_1}\right) + Tr\left(X^{2^{2k}}\left(X^{r_1} \oplus X^{2^k-1} \oplus X^{2^{3k}(2^k-1)}\right)\right). \tag{3.21}$$

Notice that by applying Corollary 3.2.4 to (3.21), only one multiplication operation is required to generate $X^{r_1} = X^{2^k+1}$ (excluding the generation of the signal X^{2^k-1}). Figure 3.4 captures the

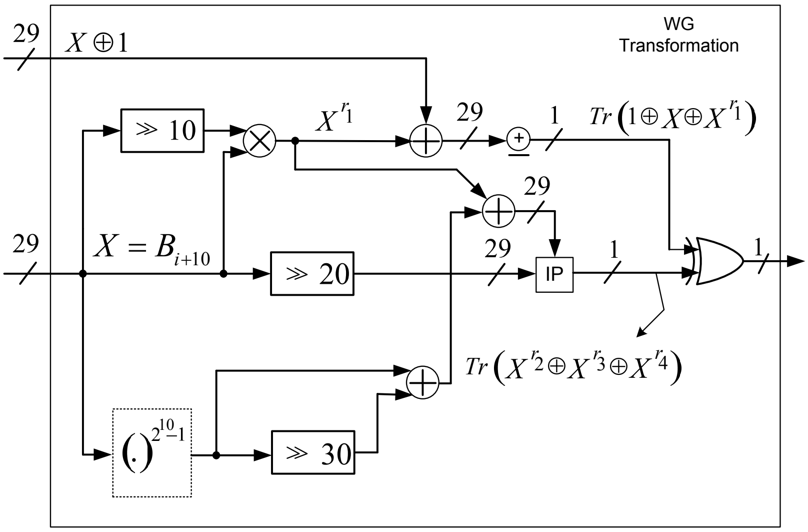


Figure 3.4: The proposed design of the WG transformation.

resulting architecture of the WG transform in (3.21). In this figure, the block denoted by "IP" generates the inner product of the two 29-bit inputs, while $\underline{\oplus}$ adds the 29-bits at its input over $GF(2)$. This architecture uses 5 field multipliers, i.e., 4 multipliers less than the WG transform presented in [68].

In Figure 3.4, the key stream bits are obtained by XORing $Tr(1 \oplus X \oplus X^{r_1})$ and $Tr(X^{r_2} \oplus X^{r_3} \oplus X^{r_4})$. $Tr(1 \oplus X \oplus X^{r_1})$ is the $GF(2)$ addition of the coordinates of $1 \oplus X \oplus X^{r_1}$ w.r.t the ONB. On the other hand, notice that the signals X^{r_3} and $X^{r_2} \oplus X^{r_4}$ do not exist in the WG transform. This is because $Tr(X^{r_2} \oplus X^{r_3} \oplus X^{r_4})$ is generated directly from $X^{2^{2k}}$, X^{r_1}, X^{2^k-1}, and $X^{2^{3k}(2^k-1)}$ using an inner product operation, as it is stated in (3.21). This absence of the two signals X^{r_3} and $X^{r_2} \oplus X^{r_4}$ resulted in the elimination of the Initial Feedback signal. The next section proposes a recovery method for generating the Initial Feedback signal, which is only used in the key initialization phase.

3.2.3 Serializing the Computation of the Initial Feedback Signal

This section presents a method for the recovery of the Initial Feedback signal through serialized computation. To accomplish the multiplication operations during this serial computation, the existing finite field multiplier which is used in generating the signal X^{r_1} in Figure 3.4, is utilized. The proposed scheme generates the Initial Feedback signal by serially computing it over three consecutive clock cycles. Denote this complete round of the serialized Initial Feedback computation (three clock cycles) as an "extended key initialization round. And the single clock cycle version of this computation (as in the MOWG design) as a "simple round". Therefore, with serialization, the entire key initialization phase requires $3 \times 22 = 66$ clock cycles instead of 22 clock cycles (that is, 22 extended rounds instead of 22 simple rounds). It is noted that this only affects the key initialization phase without increasing the number of cycles required for the run phase.

The expansion of the key initialization round from 1 to 3 clock cycles is established through the support of a new FSM's control signal, namely, lfsr_clk (Figure 3.5). This signal controls the clock input of the LFSR and triggers it to shift once every three clock cycles. Also, in order to compute the Initial Feedback signal over three stages, a new hardware module denoted as the Serialized Key Initialization Module (SKIM) will be introduced (Figure 3.6). This module uses the available signals and the field multiplier which is used in the generation of X^{r_1}, in Figure 3.4. This module schedules the proper inputs to the field multiplier in each stage of the serial computation by means of some multiplexers. The output of these multiplexers are controlled by two new signals generated by the FSM, namely, s_0 and s_1 (Figure 3.5). The intermediate results, between two consecutive stages of the computation, are stored in internal 29-bit Registers of the SKIM module. In the following, the FSM changes required for the support of the serialization process are first introduced. Then, the architecture and operation of the SKIM module and its integration to the WG transform in Figure 3.4, are discussed.

3.2.3.1 Architecture and Operation of the Modified FSM

Here, the new architecture and operation of the FSM are described. The architecture, which is shown in Figure 3.5, generates the new set of control signals lfsr_clk, s_0, and s_1. These are required for the serial computation of the Initial Feedback signal. Before each run of the cipher, the FSM resets its 11-bit one-hot counter to $(1, 0, \ldots, 0)$ and its 2-bit binary counter to $(0, 0)$ (where the leftmost bit and the rightmost bit, within the brackets, denote the lowest output bit and the highest output bit of the corresponding counter, respectively). This is done by means of pulling down the reset inputs. When the reset signal is released, the 2-bit binary counter becomes ready. At the same time, the 11-bit one-hot counter's reset input stays pulled down for an extra clock cycle. This is due to the 1-bit Register connected to the input of the AND gate which drives its reset input. This assures that the $(1, 0, \ldots, 0)$ state of the 11-bit one-hot counter consumes a clock cycle, at the beginning of the loading phase. After 11 clock cycles, from the release of the reset signal, the 11-bit one-hot counter returns to the $(1, 0, \ldots, 0)$ state. At this point it triggers the clock input of the 2-bit binary counter. The 2-bit binary counter changes its state to $(1, 0)$, triggering the start of the key initialization phase. Then, the clk signal starts triggering the clock input of the 3-bit one-hot counter. However, the counting will start one clock cycle later, when the output of the 1-bit Register connected to the 3-bit one-hot counter's reset input pulls up. This in turn assures that the 3-bit one-hot counter consumes one

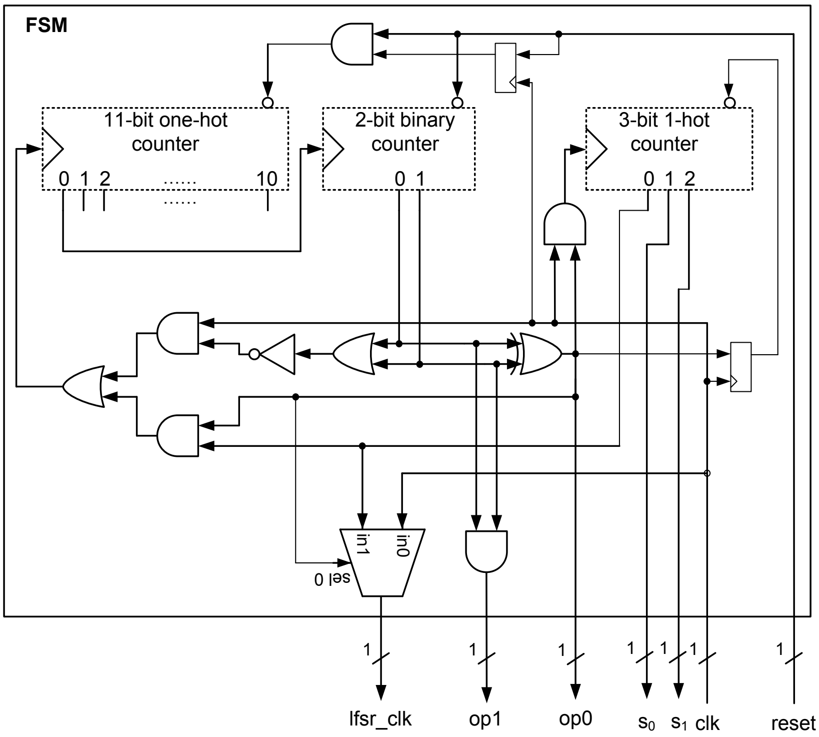


Figure 3.5: Modified FSM after adding the new 3-bit one-hot counter.

clock cycle, before incrementing its initial state of $(1,0,0)$, at the start of the key initialization phase. During this phase, the first output bit of the 3-bit one-hot counter drives the clock input of the 11-bit one-hot counter. Therefore, it takes 33 clock cycles for the 11-bit one-hot counter to complete 11 counts. Hence, it takes 33 clock cycles for the 2-bit binary counter to increment. Therefore, it requires 66 clock cycles for the 2-bit binary counter to increment twice in order to start the running phase. When the running phase starts, with the 2-bit binary counter's state at $(1,1)$, the 11-bit and the 3-bit one-hot counters stop counting, as their clock inputs become idle.

Notice that during the key initialization phase, the lfsr_clk is driven by the first output of the 3-bit one-hot counter. Hence, the LFSR shifts once every three clock cycles. The two signals s_0 and s_1 are derived from the 3-bit one-hot counter's output according to Table 3.3. Notice that this table is realized without any additional hardware by setting s_0 to be the second output, and s_1 to be the third output, of the 3-bit one-hot counter, respectively. Therefore, (s_0, s_1) produces the three patterns of $(0,0)$, $(1,0)$, and $(0,1)$ during the first stage, the second, and the third stage of an extended key initialization round, respectively. During the running phase, (s_0, s_1) will generate $(0,0)$. The following section shows how these patterns are used to accomplish the proper functionality in the key initialization phase as well as in the running phase.

3-bit one-hot counter			s_1	s_0
bit 2	bit 1	bit 0		
0	0	1	0	0
0	1	0	0	1
1	0	0	1	0

Table 3.3: Signals s_0 and s_1 as a function of the output of the 3-bit one-hot counter.

3.2.3.2 Architecture and Operation of the Serialized Key Initialization Module

Here, the SKIM module, which performs the serialized computation of the Initial Feedback signal over an extended key initialization round (three clock cycles), is presented.

Figure 3.6 is a block diagram describing the architecture of this module. The Initial Feedback signal in this figure is connected to the LFSR's input multiplexer as shown in Figure 2.2. Also, X^{r_1} connectivity is shown in more details in Figure 3.7. In this figure, the block denoted by "IP" generates the inner product of the two 29-bit inputs, while $\oplus$ adds the 29-bits at its input over $GF(2)$. The double-headed arrows under a component (correspond to inserted registers) and the dotted arrow output (Initial Feedback), are used for pipelining (see Subsection 3.3.2). The numbers under a register specify the clocking of that register within the pipelined scheme, during initialization phase. During the extended key initialization round, the two signals s_0 and s_1 in Figure 3.6 change values in each stage as mentioned in the previous section. These two

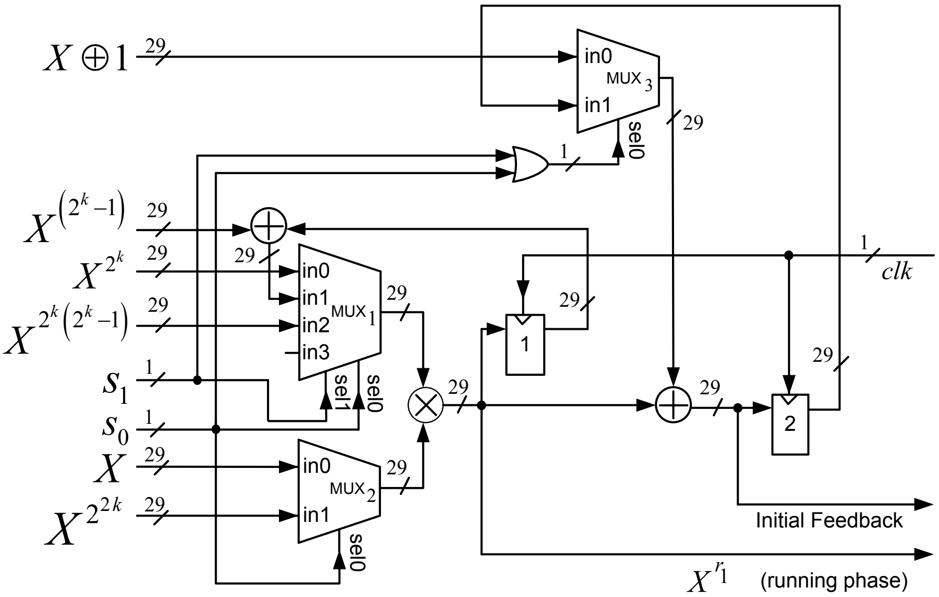


Figure 3.6: Block diagram of the SKIM module.

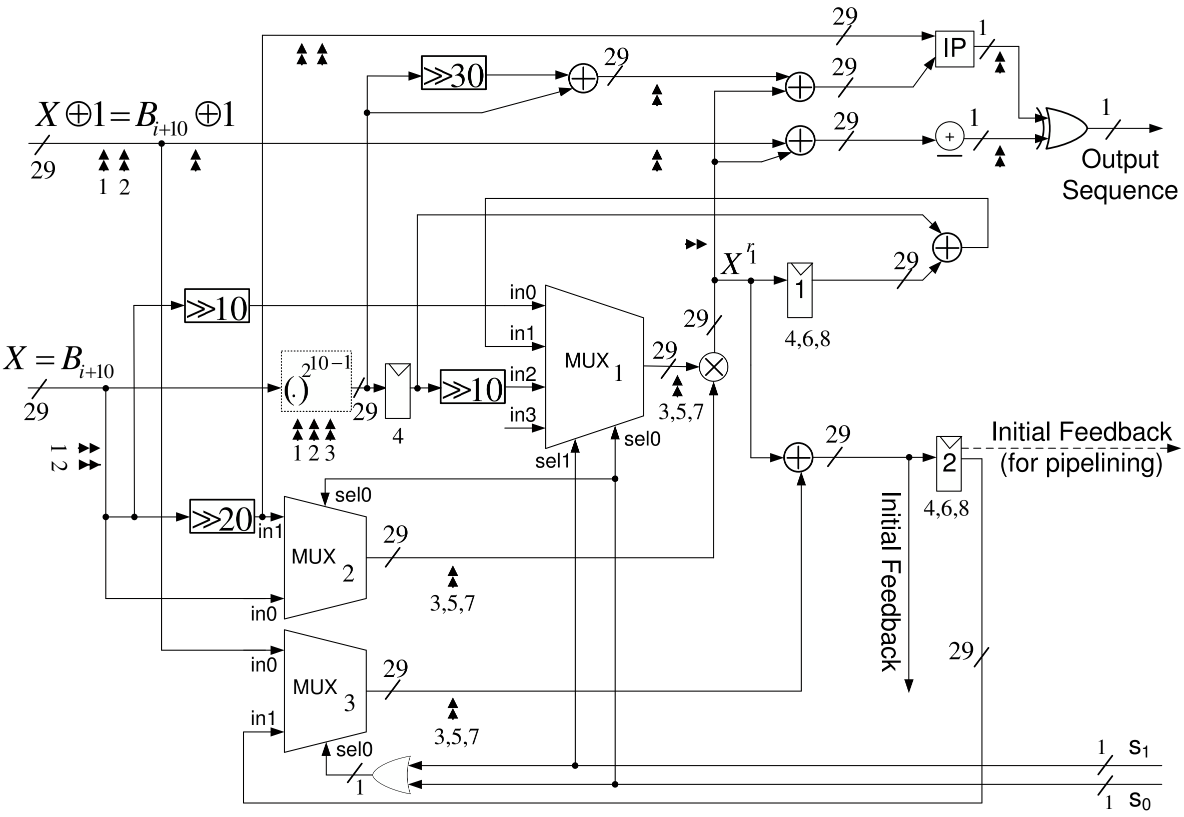


Figure 3.7: The proposed WG transformation after integration with the SKIM module.

signals control the outputs of the three multiplexers MUX_1, MUX_2, and MUX_3 according to Table 3.4. In each stage of the extended key initialization round, the SKIM module computes a partial value of the Initial Feedback signal and stores it in Register 2 (see Figure 3.6).

During the first clock cycle, s_0 and s_1 are both at low logic levels. Hence, MUX_1, MUX_2, and MUX_3 generate the signals X^{2^k}, X, and $X \oplus 1$ at their outputs, respectively. The output of the multiplier becomes $X^{r_1} = X^{2^k+1}$ and that of the $GF\left(2^{29}\right)$ adder is $X^{r_1} \oplus X \oplus 1$. Upon receiving a new clock signal, i.e. at the start of the second clock cycle, Register 1 and Register 2 update their states with the output signal of the multiplier and the output of the $GF\left(2^{29}\right)$ adder, respectively. Also, X^{2^k-1} is stored in a 29-bit register. At the same time s_0 pulls up forcing the outputs of MUX_1, MUX_2, and MUX_3 to become $X^{r_1} \oplus X^{2^k-1}$, $X^{2^{2k}}$, and $X^{r_1} \oplus X \oplus 1$ (the state of Register 2 when the clock signal arrived), respectively. With these settings of the multiplexers and the registers, the multiplier output changes to $X^{r_2} \oplus X^{r_4} = X^{2^{2k}}\left(X^{r_1} \oplus X^{\left(2^k-1\right)}\right)$ and that of the $GF\left(2^{29}\right)$ adder to $X^{r_4} \oplus X^{r_2} \oplus X^{r_1} \oplus X \oplus 1$, denoting Register 1's and Register 2's next states, respectively, when the third clock signal arrives. When the third clock cycle starts, s_0 changes to low logic level while s_1 changes to high logic level, which forces MUX_1, MUX_2, and MUX_3 to generate $X^{2^k\left(2^k-1\right)}$, X, and $X^{r_4} \oplus X^{r_2} \oplus X^{r_1} \oplus X \oplus 1$ at their outputs, respectively. The multiplier

Stage	s_0	s_1	Output			Next State	
			MUX_1	MUX_2	MUX_3	Register 1	Register 2
1	0	0	X^{2^k}	X	$X \oplus 1$	X^{r_1}	$X^{r_1} \oplus X \oplus 1$
2	0	1	$X^{r_1} \oplus X^{(2^k-1)}$	$X^{2^{2k}}$	$X^{r_1} \oplus X \oplus 1$	$X^{r_4} \oplus X^{r_2}$	$X^{r_4} \oplus X^{r_2} \oplus X^{r_1} \oplus X \oplus 1$
3	1	0	$X^{2^k(2^k-1)}$	X	$X^{r_4} \oplus X^{r_2} \oplus X^{r_1} \oplus X \oplus 1$	X^{r_3}	$X^{r_4} \oplus X^{r_3} \oplus X^{r_2} \oplus X^{r_1} \oplus X \oplus 1$

Table 3.4: Multiplexers outputs and next states of Register 1 and Register 2 as a function of s_0 and s_1.

and the $GF\left(2^{29}\right)$ adder outputs become $X^{r_3} = X^{2^k(2^k-1)+1}$ and $X^{r_4} \oplus X^{r_3} \oplus X^{r_2} \oplus X^{r_1} \oplus X \oplus 1$, respectively.

At the arrival of the fourth clock signal (the beginning of a new extended key initialization round) s_0 and s_1 both change back to low logic levels, the LFSR is clocked and latched with the result of the bit-wise XOR of the computed Initial Feedback signal ($X^{r_4} \oplus X^{r_3} \oplus X^{r_2} \oplus X^{r_1} \oplus X \oplus 1$) and the LFSR's Linear Feedback signal. At the arrival of the 67-th clock signal, the LFSR would have been clocked 22 times and the running phase starts.

Throughout the run phase, both s_0 and s_1 stay at logic level 0; therefore MUX_1 generates the signal X^{2^k} and MUX_2 generates the signal X. With these values, the multiplier generates X^{r_1} and the WG transform in Figure 3.7 produces a stream bit, for each cycle.

The following section, studies the space and time complexities of the proposed WG(29, 11) cipher.

3.2.4 Space and Time Complexities

This section begins with presenting the hardware complexity of the proposed WG implementation, followed by its time complexity.

3.2.4.1 Space Complexity

The space complexity of the WG transform is reduced, while that of the WG's FSM is slightly increased, compared to the corresponding ones in the proposed MOWG. Please refer to Tables 3.2 and 3.5 for a comparison of the number of gates in the transform and FSM of the MOWG and WG, respectively. In what follows, the hardware complexities of the WG transform and its FSM are first summarized. Then, the overall hardware cost of the WG design is obtained.

WG Transformation The space complexity of the WG transform has been improved compared to the MOWG transform. This is mainly because the number of field multipliers in the WG transform is reduced by 2 w.r.t that in the MOWG transform. On the other hand, compared to the MOWG transformation in Figure 3.2, the design in Figure 3.7 has the following additional components: a $GF\left(2^{29}\right)$ adder, a 29-bit $GF(2)$ addition, three 29-bit Registers, an XOR gate, an OR gate, one 4-to-1 29-bit multiplexer, two 2-to-1 29-bit multiplexers with 2 selector NOTs, and an inner product. A 29-bit $GF(2)$ adder consists of 28 XOR gates. A 2-to-1 29-bit multiplexer consists of 29 parallel 2-to-1 1-bit multiplexers. The inner product has 29 AND gates and 28 XORs. Refer to Subsection 3.1.5.1 for details about the hardware of the other components listed above. By adding the hardware of the additional components to the gate count in the MOWG transform (Table 3.2), and then subtracting the hardware cost of two field multipliers, the total hardware cost of the proposed WG transform is obtained as listed in Table 3.5.

Component	N_R	N_A	N_X	N_O	N_I
WG Transform	87	4524	6292	146	4
FSM (Figure 3.5)	18	7	1	3	2

Table 3.5: Count of 1-bit registers and logic gates in the components of the proposed WG(29, 11).

FSM The FSM depicted in Figure 3.5 has additional two AND gates, two OR gates, a 2-to-1 1-bit multiplexer (with 1 selector NOT), 1-bit Register, and a 3-bit one-hot counter, as compared to Figure 3.3. Similar to the 11-bit one-hot counter, the 3-bit one-hot counter is simply composed of a three stages circular shift register with set/reset inputs having the output of the last register fed to the input of the first register. By adding the gates in the mentioned components to the number of gates of the FSM in Figure 3.3 (Table 3.2), the total hardware cost of the FSM in Figure 3.5 is as shown in Table 3.5.

The LFSR and the 4-to-1 MUX of the WG have same complexities as the ones in the MOWG (Table 3.2). Moreover, the WG design contains two 29-bit bit-wise complement operations (inverter symbol (a) and inverter symbol (b) in Figure 3.2) and a $GF\left(2^{29}\right)$ adder (computing the bit-wise XOR of Initial Feedback signal and the Linear Feedback signal). Let N_O^{WG}, N_I^{WG}, N_R^{WG}, N_A^{WG}, and N_X^{WG} denote the number of OR gates, Inverters, 1-bit Registers, AND gates, and XOR gates in the proposed WG cipher, respectively. Therefore, by adding the corresponding number of gates in the $GF\left(2^{29}\right)$ adder and in inverter symbols (a) and (b) to the number of gates in the 4-to-1 multiplexer, and the LFSR (see Table 3.2), and as well, to the number of gates in the FSM and the WG transform (see Table 3.5) one obtains

$$N_O^{WG} = 236, \quad N_I^{WG} = 94, \quad N_R^{WG} = 424,$$
$$N_A^{WG} = 5546, \quad N_X^{WG} = 7685.$$

3.2.4.2 Time Complexity

Here, the propagation delay along the critical path of the proposed WG design is derived. Notice that the LFSR is not a candidate for the critical path, since it still has less multipliers contributing to its propagation delay, compared to the WG transform. In what follows, the formulation of the longest path during the key initialization phase is presented. After this, the longest path during the running phase is proved to be the critical path of the cipher.

From Figure 3.7, one can see that the critical path during the key initialization phase extends between the LFSR (not shown in the figure) and the output of the module generating X^{2^k-1}. Hence, the propagation delay through longest path during key initialization phase of the WG is

$$T_{KIPh} = 24T_X + 4T_A + T_R. \tag{3.22}$$

The longest path of the WG cipher during the run phase can also be seen in Figure 3.7 extending between the LFSR and the cipher's output, passing through the X^{2^k-1} module. Therefore, the propagation delay of this run phase longest path is easily obtained by adding the delays of its components as follows

$$T_{RunPh} = 32T_X + 5T_A + T_R. \tag{3.23}$$

From (3.22) and (3.23), the critical path of the cipher is (3.23).

3.3 Results and Comparisons

The following sections compare the proposed designs of the MOWG$(29, 11, 17)$ and the WG$(29, 11)$ ciphers with the corresponding previous implementations in [58], [68], and [56]. Also, further optimizations and general applicability of the proposed algorithms are discussed.

3.3.1 Results from FPGA and ASIC Implementations

The proposed WG and MOWG designs, together with the WG in [68], have been realized using ASIC and FPGA implementations. The ASIC speed and area results are for the 65nm CMOS technology based on Synopsys Design Compiler's estimate of area and clock speed prior to place-and-route, with medium effort for optimizations. The power consumption readings have

Cipher	Transform Architecture Type	Technology	Primary Optimization Target	# Clocks in Init. Phase	Bits/Cycle (Run Phase)	Area (KGate)	Latency (nsec)	Speed (MHz)	Throughput (Mbps)	Throughput Per Area (Kbps/Gate)	Dynamic Power (mW)	Energy (mJ/Gbit)
WG-7 @2MHz [60] (software)	Look-up Table	4-bit microcontroller MARC4 ATAM893 - D	-	10084	-	Code Lines = 1097, Exp/Ret = 7/4	-	-	0.098	-	-	-
WG-7 @8MHz [60] (software)	Look-up Table	8-bit microcontroller ATmega family	-	10074	-	SRAM = 0, Flash = 1100	-	-	0.28	-	-	-
WG [68]	Multiplier-based	CMOS 65nm	Area	22	1	33.2	6.94	144	144	4.34	7.28	50.6
WG [56]	Look-up Table (ROM)	-	-	-	1	319 Registers + 9000 XORs + 2^{29} ROM bits	-	-	-	-	-	-
MOWG [58]	Multiplier-based (Pipelined with Reuse)	CMOS 90nm	-	22	-	187 (Kμm^2)	-	1000	8500	45 (Kbps/μm^2)	-	-
WG (Figure 3.7)	Multiplier-based	CMOS 65nm	Area	66	1	19.9	4.45	224	224	11.2	4.45	19.8
MOWG (Figure 3.2)	Multiplier-based	CMOS 65nm	Area	22	17	26	6.62	151	2567	98.73	5.89	2.3

Table 3.6: Results obtained from ASIC implementations.

Cipher	Transform Architecture Type	Family	Synthesis Tool	Primary Optimization Target	# Clocks in Init. Phase	Bits/Cycle (Run Phase)	LUTs	Latency (nsec)	Speed (MHz)	Throughput (Mbps)	Throughput Per Area (Kbps/LUT)	Total Power (mW)	Energy (J/Gbit)
WG [68]	Multiplier-based	Virtex 4 (xc4vfx12sf363-10)	Xilinx XST	Area	22	1	6449	33.3	30	30	4.65	380	12.67
MOWG [58]	Multiplier-based (Pipelined with Reuse)	Stratix II (EP2S15F484C)	Mentor Graphics PrecisionRTL	-	22	-	4184	-	218	1853	443	-	-
WG (Figure 3.7)	Multiplier-based	Virtex 4 (xc4vfx12sf363-10)	Xilinx XST	Area	66	1	4044	29.4	34	34	8.41	187	5.5
MOWG (Figure 3.2)	Multiplier-based	Virtex 4 (xc4vfx12sf363-10)	Xilinx XST	Area	22	17	5512	28.6	35	595	108	342	0.57

Table 3.7: Results obtained from FPGA implementations.

been conducted under 140 MHz frequency for all the designs. The FPGA designs have been synthesized using Xilinx Synthesis Tool (XST) [2]. The FPGA area and speed results are for Xilinx Virtex4 series FPGA device xc4vfx12sf363-10. All FPGA results are for post place-and-route and the power consumption results have been recorded for a frequency of 29 MHz for all the designs.

The reported ASIC and FPGA results are listed in Tables 3.6 and 3.7, respectively. In Table 3.6, the WG-7 results (another member of the WG family based on an LFSR over $GF\left(2^7\right)$) are from software implementations presented in [60]. KGate is the area equivalence in terms of number of NAND gates $\times 10^3$ (estimated area of one NAND gate is 2.08 $(\mu\text{m})^2$). The results for the WG$(29, 11)$ hardware implementation proposed by [56] are based on theoretical analysis. "Exp" and "Ret" denote the depth of the expression and return stacks, respectively. In Tables 3.6 and 3.7, Throughput is the # bits per cycle × speed (Mbps = 10^6bit/second). Gbit = 10^9bit. Also, the readings shown from the MOWG design in [58] were reported for the pipelined-with-reuse version of the transform. The following paragraphs analyze the reported results and compare the proposed WG and MOWG designs to other listed ones.

The reported results show that the proposed WG takes longer to finish its initialization phase compared to the one in [68] (293 nsec (ASIC)/1.94 msec (FPGA) in the proposed scheme compared to 152 nsec (ASIC)/0.73 msec (FPGA) in [68]). This is not significant because initialization is executed only once per a run. The reported results also show that the proposed WG is superior to the one in [68] in terms of throughput, area, and power consumption. The proposed WG has lower latency, by 36% (ASIC) and 12% (FPGA), w.r.t the one in [68]. And accordingly, the speed/throughput of the proposed WG is increased by 55% (ASIC) and 13% (FPGA), compared to [68]. Also, notice that the normalized throughput (proposed) is twice the one in [68]. This is due to the higher throughput and the significant reduction in area (area reduced by 40% for ASIC and by 37% for FPGA) of the proposed WG compared to the one in [68]. Moreover, one can see that the proposed WG consumes less power (39% ASIC, 51% FPGA) and uses less than half the energy reported for [68].

The WG design in [56] requires 2^m ROM bits for a general WG over $GF(2^m)$. On the other hand, the area of the proposed WG is dominated by its field multipliers, which have space complexity quadratic in m. Specifically, for the WG$(29, 11)$, 2^{29}-bits of ROM are required in [56] (in addition to 9000 XORs and 319 registers). There are no results in [56] about the running speed of the presented WG. According to a similar study on ROM-based and multiplier-based MOWG designs by [58], ROM based ASIC implementations are always larger and slower than using field multipliers, for $m > 11$.

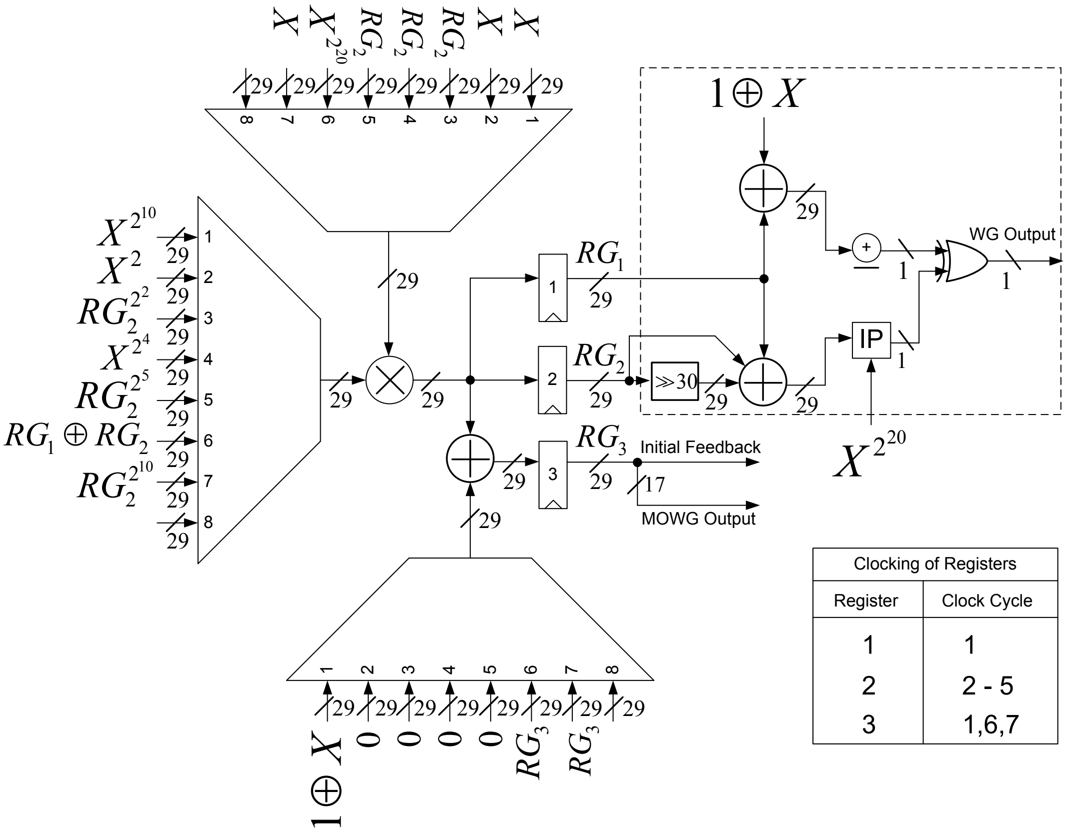
The proposed MOWG design is expected to offer better area and speed compared to the one presented in [58]. The proposed MOWG has 8 multipliers compared to 9 in [58]. Therefore, its area is expected to be scaled down by a ratio close to $8/9$ w.r.t the one in [58]. It is noted that the results from [58] are reported for the pipelined-with-reuse version of the transform. Applying pipeline-with-reuse techniques to the proposed MOWG would result in speed and area readings similar to the ones reported in [58]. For the non-pipelined and the pipelined (without reuse) versions, however, the proposed MOWG is expected to show lower area and a slightly higher speed/throughput, and lower latency, compared to the corresponding versions from [58]. This is due to the removed multiplier and the removed inverters from its critical path (see Figure 3.2). Notice that a 6-stage pipeline of the proposed MOWG offers 6-times the throughput which is reported for its non-pipelined version in Tables 3.6 and 3.7 (see Section 3.3.2). That is, almost double the throughput provided by the pipeline-with-reuse MOWG in [58].

The proposed WG offers higher clock speed, and better area and power consumption, compared to the proposed MOWG. However, the proposed MOWG has higher throughput and better energy per bit. Most important, the WG has more good randomness properties than the MOWG cipher [68, 58]. Therefore, when security and randomness are critical for the application, the proposed WG design is preferred. If instead, throughput and area are the critical criteria for the application, then, in this case, the proposed WG design is superior for low area applications, while the proposed MOWG serves better for high throughput applications. It is noted that one can apply serialization or pipelining to the WG/MOWG transforms for achieving lower area or higher throughput, if it is demanded by the application. This is discussed in the next section.

3.3.2 Discussion

This section discusses the serialization and pipeline techniques as further optimizations to the proposed WG and MOWG. Also, the applicability of the proposed techniques to general MOWG/WG ciphers, when field elements are represented in the NB, is considered.

For low throughput applications, smaller area can be achieved by serial computation of the MOWG/WG transforms. Figure 3.8 presents how this is done using one multiplier. In



Clocking of Registers	
Register	Clock Cycle
1	1
2	2 - 5
3	1,6,7

Figure 3.8: Serial Implementation of MOWG/WG Stream Ciphers.

this figure, the dotted square is used, only, for generating the WG stream bits. The rest of the diagram is common for MOWG and WG. The initialization round takes 8 cycles for both transforms. During run phase of the MOWG, 17 output bits are generated every 7 cycles. For the WG, a stream bit is produced every 6 cycles. The maximum propagation delay is equivalent to 17 levels of gate delays. Compared to 38 levels in (3.23) (WG) and 46 levels in (3.10) (MOWG), the clocks of the serial WG and MOWG are 2.2 and 2.7 times faster, respectively. Therefore, the throughput of the serial versions of the WG and the MOWG ciphers are almost $2/6$ and $3/7$ of the corresponding original ones in Figures 3.2 and 3.7, respectively. The total gate counts for the serial versions of the transforms are 4155 (WG) and 4011 (MOWG). Compared to 11053 gates in the WG transform (Section 3.2.4.1) and 14529 gates in the MOWG transform (Section 3.1.5.1), then, the area of the serial versions of the WG/MOWG transforms are almost $2/5$ and $2/7$ of their original architectures, respectively. If even lower area is demanded, a digit-level field multiplier [76, 74] can be deployed, adding more cycles for each multiplication.

The proposed schemes can achieve higher throughput through pipelined transforms. The LFSR should be reconstructed using the Galois-style feedback, or simply by placing the multiplication with β in between cells B_{i+1} and B_i. Otherwise, the LFSR's speed will constrain the pipelining. Figure 3.2 shows how to achieve a 6-stage pipeline of the MOWG transform using 19 29-bit registers. The pipelined MOWG critical path has 7 levels of logic gate delays. The corresponding throughput and run phase latency are $17/(T_A+6T_X)$ and $6\,(T_A + 6T_X)$, respectively. Since (3.10) has 46 levels of logic gate delays, thus, the throughput of the pipelined MOWG is almost 6 times higher. Similarly, Figure 3.7 shows a 6-stage pipeline of the WG transform. From this figure, one can find the pipelined WG's latency and throughput as $6\,(T_A + 6T_X)$ and $1/(T_A+6T_X)$, respectively (the latency during initialization is higher, i.e., $8\,(T_A + 6T_X)$). Compared to the throughput which results from (3.23), this is almost 5 times higher. For even higher throughput, the unfolding technique presented in [26] can be deployed. Simply, the MOWG/WG LFSR is unfolded to generate n outputs ($2 \le n \le 11$) per a cycle. Hence, by implementing the same number of transforms, the throughput will be n-times higher at the expense of a proportional area increase.

Notice that Equation (3.4) is a general form of the WG permutation (for any MOWG(m, l, d)). Since squarings are cyclic shifts in the NB, then, only the architecture of the power $2^k - 1$ will vary for different values of $k = \left\lceil \frac{m}{3} \right\rceil$. By having the *WGPm*, the MOWG transform is just a proper selection of d bits from the *WGPm* [58]. Also, notice that the compliment LFSR in (3.5) is general for any $GF\left(2^m\right)$. Similarly, except for the power $2^k - 1$, Equation (3.21) is general for any WG(m, l). However, (3.11) is only applicable to $GF\left(2^m\right)$ where self-dual NB exist. Therefore, if there is not self-dual NB [15], the inner product which is used to compute $Tr\left(X^{2^{2k}}\left(X^{r_1} \oplus X^{2^k-1} \oplus X^{2^{3k}\left(2^k-1\right)}\right)\right)$ in Figures 3.4 and 3.7 should be replaced with a field multiplication followed by a trace.

It is interesting to investigate the WG implementation in the PB. It is known that the PB offers area efficient multipliers, compared to the NB representation. However, there is a penalty due to the additional space and propagation delay introduce by the squaring operations. This is considered in the next chapter.

3.4 Conclusion

Two new designs for the MOWG$(29, 11, 17)$ and the WG$(29, 11)$ ciphers have been proposed. As compared to the MOWG presented in [58], the proposed MOWG reduces the number of field multipliers in the transform by one through signal reuse. Also, it increases the speed by eliminating two inverters delay from the critical path. This is accomplished by reconstructing the key/IV loading algorithm and the feedback polynomial of the LFSR. The proposed WG is an optimization of the proposed MOWG with trace (WG version). It is obtained through using the new properties of the trace function for type-II ONB, accompanied with serialized computation of the Initial Feedback signal during key initialization phase.

The proposed designs have been implemented on ASIC and FPGA. The ASIC implementations show that the proposed WG implementation achieves better results compared to [68] for area, speed, and power consumption. The WG improves the power consumption by a 39% reduction, area by a 40% reduction, and speed by an increase of 55%. Similarly, the FPGA implementations show that the proposed WG achieves better results for area, speed, and power consumption compared to [68]. The power consumption is reduced by 51%, the area is reduced by 37%, and the speed is increased by 13%.

Based on these results, the proposed implementations of the MOWG$(29, 11, 17)$ cipher and the WG$(29, 11)$ cipher are promising candidates for high speed and limited resources platforms, respectively, where throughput, area, and power consumption are of critical importance and the guaranteed randomness properties are required.

Chapter 4

Implementations of the WG Stream Ciphers Using PB

Previous chapter presented an optimized WG$(29, 11)$ design based on the Type-II Optimal Normal Basis (ONB-II). Using the novel trace property presented in the previous chapter, the design requires only 6 field multipliers. In this chapter, PB representation is considered for the fist time in the WG stream ciphers. A novel method for computing the trace of the multiplication of two field elements represented in the PB is proposed. It is noted that the proposed trace method is applicable to any $GF(2^m)$, while the one presented in the previous chapter only applies to fields where self-dual bases exist. Based on the trace method proposed here, a PB-based hardware design of the WG$(29, 11)$, which uses 6 multipliers, is presented. Also, pipelined and serialized instances of this standard design are presented (see Figure 4.1). The reported results for the 65nm CMOS ASIC realization of the proposed standard WG$(29, 11)$ design shows smaller area and, slightly improved normalized throughput, compared to the best result presented in the previous chapter.

The only WG-16 hardware design, which uses NB, is presented in [35]. This design is based on composite field arithmetic and properties of the trace function in the tower field representation. In this chapter, a new formulation of the WG-16 permutation which requires 8 multiplications compared to 10 in the formulation of [35] is proposed. Furthermore, a new formulation for the trace function of the multiplication of two field elements is derived, based on which a PB-based WG-16 design is proposed using only 6 multipliers for its transform. Also, pipelined and serialized versions of this standard design, are presented and for each design both the traditional PB and Karatsuba multipliers are considered (see Figure 4.1). According to the conducted ASIC (CMOS 65 nm) implementations, the proposed pipelined instance of the WG-16 offers double the throughput, while it slightly reduces the area, compared to the results reported in [35].

The goal of this chapter is to show hardware implementations for WG ciphers, which in return, provides trade-offs between randomness properties and performance for a selection of ciphers for a particular application. In particular, it is shown that the proposed WG-16 implementations comply with the throughput requirements of the 4G domain. The contributions of this chapter which include a novel trace method and nine new designs of the WG stream ciphers are summarized in Figure 4.1. In this figure, the standard WG(29, 11) implementation shows lower space and slightly improved normalized throughput, compared to the one in previous chapter. Also, the pipelined instance of the proposed WG-16 reports higher throughput and lower area compared to the corresponding ones in [35].

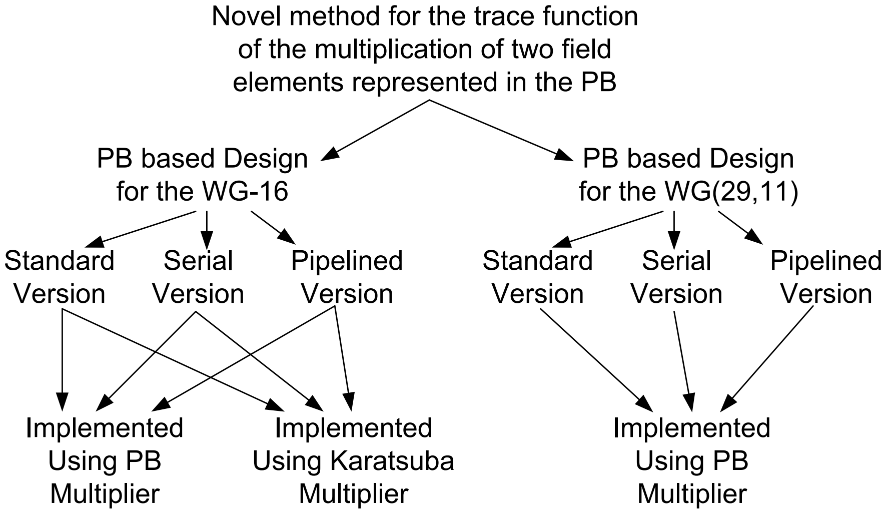

Novel method for the trace function of the multiplication of two field elements represented in the PB
PB based Design for the WG-16
PB based Design for the WG(29,11)
Standard Version
Serial Version
Pipelined Version
Standard Version
Serial Version
Pipelined Version
Implemented Using PB Multiplier
Implemented Using Karatsuba Multiplier
Implemented Using PB Multiplier

Figure 4.1: Contributions of this work.

It is noted that, throughout this chapter, $\oplus$ represents the addition operator in $GF(2^m)$. Also, $C(Z) = Z^l \oplus \sum_{i=0}^{l-1} C_i Z^i$, $C_i \in GF(2^m)$ is the characteristic polynomial of an l-stages LFSR over $GF(2^m)$, from which the feedback recurrence relation can be derived as $A_{j+l} = \sum_{i=0}^{l-1} C_i A_{i+j}$, where $j \geq 0$, $A_i \in GF(2^m)$, and $(A_0, A_1, \ldots, A_{l-1})$ is the initial state of the LFSR.

It is noted that a version of this chapter appears in [32]. The chapter is organized as follows. Section 4.1 presents the proposed WG(29, 11) hardware designs based on the PB. Section 4.2 presents the proposed WG-16 hardware designs based on the PB. Results based on ASIC implementations are discussed in Section 4.3. Section 4.4 concludes the chapter.

4.1 Architectures of the WG(29, 11) Stream Cipher

The WG(29, 11) uses exponentiation over $GF\left(2^{29}\right)$, and therefore, an ONB was assumed to be more efficient for hardware design, compared to other representations, due to the free cost of squaring operations [39, 68]. As it is shown, the previous chapter uses new properties of the trace function for type-II ONB in order to build the cipher using only 6 field multiplications, which is the most optimal WG(29, 11) design so far.

In this section, three PB-based designs for the WG(29, 11) are proposed. These designs include a standard architecture, its serial version, and its pipelined version. The serial version is suitable for low-area applications whereas the pipelined one is proposed for high-speed applications. To the best of the author knowledge, this is the first implementation of the WG cipher based on the PB representation. The parameters of the cipher are chosen carefully for a low area design. Also, for further area reduction, the proposed implementation uses properties of the trace function for PB in order to optimize the WG transform. The proposed scheme offers smaller area and a slightly higher normalized throughput, compared to the best results presented in the previous chapter, at the expense of a small decrease in the speed. In this section, first, the WG transform formulations are derived. This is followed by finding the design parameters. After that, the proposed architecture of the WG(29, 11) is introduced.

4.1.1 Formulation of $WGT29$

Since replacing $\left(Y^{2^{20}-2^{10}+1}\right)$ with $\left(Y^{2^{20}-2^{10}+1}\right)^{2^{20}}$ in (2.9) does not affect $Tr\,(WGP29)$, therefore $WGT29 =$

$$\begin{aligned} &Tr\left(1 \oplus Y \oplus Y\left(Y^{2^5}\right)^{2^5}\right) + \\ &Tr\left(\left(\left(Y^{2^5}\right)^{2^5}\right)^{2^{10}} \left(Y\left(Y^{2^5}\right)^{2^5} \oplus Y^{2^{10}-1} \oplus \left(Y^{2^{10}-1}\right)^{2^{30}}\right)\right). \end{aligned} \tag{4.1}$$

It is noted that (4.1) shows the order of computing the squarings in the transform. To reduce propagation delay due to squarings in the PB, $Y^{2^{10}-1}$ is computed as follows:

$$Y^{2^{10}-1} = \left(\left(\left(Y^{2^5+1}\right)^{2+1}\right)\left(Y^{2^5+1}\right)^{2^4}\right)\left(\left(Y^{2^5+1}\right)^{2+1}\right)^{2^2}. \tag{4.2}$$

The following section introduces the WG(29, 11)'s design parameters.

4.1.2 Design Parameters

This section presents the design parameters for the proposed PB implementation of the WG(29, 11). In what follows, the field polynomial, the squaring matrices, the LFSR's characteristic polynomial, the trace vector, and the formulation for directly computing the trace of the multiplication of two field elements are presented.

4.1.2.1 Field Polynomial and Squaring Matrices

To compute (4.1) and (4.2), field multiplications and squarings are used. In the original design of the WG$(29, 11)$ [39] and all reported schemes to date [68, 31], NB representation is used. The squaring is obtained by cyclic shift in NB and hence it is free in hardware implementation. However, such an operation in PB is not free. On the other hand, field multiplication using PB requires lower complexity than the one using NB. In PB, the complexities of these operations depend on the irreducible polynomial that constructs the finite field. It is known that irreducible trinomials define PBs offering field multiplications with low space and time complexities [72, 63, 5]. For $GF\left(2^{29}\right)$, the following two trinomials are irreducible over $GF(2)$

$$t_1(x) = x^{29} + x^2 + 1, \tag{4.3}$$

and its reciprocal function $t_2(x) = x^{29}\left(t_1\left(x^{-1}\right)\right) = x^{29} + x^{27} + 1$. Between t_1 and t_2, t_1 offers operations with lower space complexities. Specifically, the t_1-based PB multiplier requires $29^2 = 841$ ANDs and $29^2 - 1 = 840$ XORs with a propagation delay of $T_A + 7T_X$ [72], where T_A and T_X are the delays in an AND and an XOR, respectively. In the following, the complexities of the squarings using the PB defined by (4.3) are obtained.

Let A be an arbitrary element of $GF(2^m)$ represented in the PB, and let $V = A^2$. Denote by $\mathbf{a} = (a_0, \ldots, a_{m-1})$ and $\mathbf{v} = (v_0, \ldots, v_{m-1})$, the row vectors holding the bits which represent A and V w.r.t the PB, respectively. Then, $\mathbf{v} = \mathbf{aS}$, where $\mathbf{S}$ is the binary $m \times m$ squaring matrix whose entries are either 0 or 1 [5]. In general, $W = A^{2^e}$ is obtained as $\mathbf{w} = \mathbf{aS}^e$. This formulation involves m inner products $\mathbf{aS}^e_j$, where $\mathbf{S}^e_j$ denotes the j-th column vector of $\mathbf{S}^e$, $0 \leq j < m$. Let N_X denote the number of XOR gates. Then, the hardware realization of $\mathbf{aS}^e$ requires $N_X = \sum_{H\left(\mathbf{S}^e_j\right)>1, 0\leq j<m}\left(H\left(\mathbf{S}^e_j\right) - 1\right)$ and $T_{\mathbf{S}^e} = \lceil \log_2(\theta) \rceil T_X$, where $T_{\mathbf{S}^e}$ is the propagation delay for computing $\mathbf{aS}^e$, $H(\Omega)$ is the Hamming weight of a vector Ω, and $\theta = \max_{H\left(\mathbf{S}^e_j\right)>1}\left\{H\left(\mathbf{S}^e_j\right) \mid 0 \leq j < m\right\}$.

For the PB defined by (4.3), the squaring matrix $\mathbf{S}$ is shown in Figure 4.2. Table 4.1 lists the space and time complexities, before and after signal reuse, for the different squaring matrices used in the WG$(29, 11)$'s implementations. In this table, PD denotes propagation delay.

4.1.2.2 Characteristic Polynomial of the LFSR

A primitive characteristic polynomial of degree 11 over $GF\left(2^{29}\right)$ is required in order for the WG$(29, 11)$ to produce key-streams with maximal period of $2^{319} - 1$ [39, 68]. For space effi-

$$\begin{pmatrix}
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\
0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{pmatrix}$$

Figure 4.2: The matrix $\mathbf{S}$ for WG(29, 11).

ciency, the following primitive pentanomial is selected

$$Z^{11} \oplus Z^6 \oplus Z^2 \oplus Z \oplus \alpha, \tag{4.4}$$

where $\alpha \in GF\left(2^{29}\right)$ is a root of the defining polynomial (4.3). The primitive property of the polynomial has been verified using the "is_primitive()" method provided by the Sage Notebook online tool [3]. Let $\left\{A_i,\ 0 \leq i < 2^{319} - 1\right\}$ denote the sequence generated by (4.4). According to previous chapter, the following recurrence relation generates the sequence $\left\{B_i = A_i \oplus 1,\ 0 \leq i < 2^{319} - 1\right\}$

$$B_{j+11} = \left(B_{j+6} \oplus B_{j+2} \oplus B_{j+1} \oplus \alpha B_j\right) \oplus \alpha, \quad j \geq 0, \tag{4.5}$$

where $\{B_i = A_i \oplus 1,\ 0 \leq i \leq 10\}$ is the initial state of the LFSR. By constructing the LFSR based on (4.5) instead of (4.4), then, one obtains $Y = 1 \oplus A_{i+10} = B_{i+10}$ in (4.1) and (4.2). In addition, notice that (4.5) requires only three field additions, one field multiplication with α (a constant[1]), and one NOT gate (for addition of α).

---

[1] For the field polynomial (4.3), one can easily find that the multiplication with the constant α requires only one XOR gate with a propagation delay T_X.

	No Sig. Reuse		Sig. Reuse	
	XOR	PD	XOR	PD
$\mathbf{S}, \mathbf{S}^{30}$	15	T_X	15	T_X
$\mathbf{S}^2$	37	$2T_X$	30	$2T_X$
$\mathbf{S}^4$	118	$3T_X$	65	$3T_X$
$\mathbf{S}^5$	182	$4T_X$	97	$4T_X$
$\mathbf{S}^{10}$	374	$5T_X$	214	$5T_X$
$\mathbf{S}^{20}$	338	$5T_X$	200	$5T_X$

Table 4.1: The space and time complexities of the different squaring matrices used in the WG(29, 11).

4.1.2.3 Trace Vector

Let the elements in $GF(2^m)$ be represented in the PB which is defined by an irreducible polynomial $f(x)$ of degree m over $GF(2)$. Then, the trace of an element $A \in GF(2^m)$ is obtained as $Tr(A) = \mathbf{a}\tau^T$, where $\mathbf{a} = (a_0, a_1, \ldots, a_{m-1})$ (a_i's are coordinates of A w.r.t PB), $\tau = (\tau_0, \tau_1, \ldots, \tau_{m-1})$ is a unique and constant m-bit vector such that $\tau_i = Tr\left(\alpha^i\right) \in GF(2)$, $0 \le i < m$ and $f(\alpha) = 0$ [5]. Therefore, for the PB $\left\{\alpha^{28}, \ldots, \alpha, 1\right\}$ defined by (4.3), one obtains $\tau_i = 1$ for $i \in \{0, 27\}$ and $\tau_i = 0$ otherwise. Thus,

$$Tr(A) = a_0 + a_{27}. \tag{4.6}$$

4.1.2.4 Trace of Multiplication of Two Field Elements

Previous chapter presented a method for the direct computation of the trace of the multiplication of two elements represented in the type-II ONB. In the following, a formulation for the direct computation of the trace of the multiplication of two field elements represented in PB is constructed. This method is then used to optimize the space complexity of the PB based implementations of the WG(29, 11) and the WG-16 (see Sections 4.1.3 and 4.2.4).

Proposition 4.1.1 *Consider the m-bit trace vector $\tau = (\tau_0, \ldots, \tau_{m-1})$, $\tau_i = Tr\left(\alpha^i\right)$, where α is the root of the defining polynomial of $GF(2^m)$ over $GF(2)$ [5]. For any two field elements $A = (a_{m-1}, \ldots, a_0)$ and $B = (b_{m-1}, \ldots, b_0)$, let $C = AB \in GF(2^m)$. Then:*

$$Tr(C) = \sum_{i=0}^{m-1} \tau_i \sum_{j=0}^{i} a_{i-j} b_j + \sum_{i=0}^{m-1} \tau_i \sum_{k=0}^{m-2} q_{k,i} \sum_{j=k+1}^{m-1} a_{m-j+k} b_j, \tag{4.7}$$

where $\mathbf{Q}_{(m-1)\times m} = [q_{k,i}]$ *is the reduction matrix and,* $\mathbf{U}_{(m-1)\times m} = \left[u_{k,j}\right]$ *and* $\mathbf{L}_{m\times m} = \left[l_{i,j}\right]$ *are as follows [72]*

$$\mathbf{U} = \begin{bmatrix} 0 & a_{m-1} & a_{m-2} & \cdots & a_2 & a_1 \\ 0 & 0 & a_{m-1} & \cdots & a_3 & a_2 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & a_{m-1} & a_{m-2} \\ 0 & 0 & 0 & \cdots & 0 & a_{m-1} \end{bmatrix},$$

and

$$\mathbf{L} = \begin{bmatrix} a_0 & 0 & 0 & \cdots & 0 & 0 \\ a_1 & a_0 & 0 & \cdots & 0 & 0 \\ a_2 & a_1 & a_0 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{m-2} & a_{m-3} & a_{m-4} & \cdots & a_0 & 0 \\ a_{m-1} & a_{m-2} & a_{m-3} & \cdots & a_1 & a_0 \end{bmatrix}.$$

Proof Let $\mathbf{b} = (b_0, \ldots, b_{m-1})$ and $\mathbf{c} = (c_0, \ldots, c_{m-1})$ be row vectors holding the bits of B and C, respectively, then, from [72] one has

$$\mathbf{c}^T = \mathbf{L}\mathbf{b}^T + \mathbf{Q}^T\mathbf{U}\mathbf{b}^T, \tag{4.8}$$

where $\mathbf{Q}^T$ is the transpose of $\mathbf{Q}$. Therefore:

$$\begin{aligned} Tr(C) = \mathbf{c}\tau^T &= \left(\mathbf{L}\mathbf{b}^T\right)^T \tau^T + \left(\mathbf{Q}^T\mathbf{U}\mathbf{b}^T\right)^T \tau^T \\ &= \sum_{i=0}^{m-1}\sum_{j=0}^{m-1} l_{i,j} b_j \tau_i + \sum_{i=0}^{m-1}\sum_{j=0}^{m-1}\sum_{k=0}^{m-2} q_{k,i} u_{k,j} b_j \tau_i \\ &= \sum_{i=0}^{m-1} \tau_i \sum_{j=0}^{i} l_{i,j} b_j + \sum_{i=0}^{m-1} \tau_i \sum_{k=0}^{m-2} q_{k,i} \sum_{j=k+1}^{m-1} u_{k,j} b_j, \end{aligned}$$

where the last result is obtained by noticing that $l_{i,j} = 0$ for $j > i$ and $u_{k,j} = 0$ for $j \le k$ [72], and by replacing $l_{i,j}$ and $u_{k,j}$ with the corresponding entries from $\mathbf{L}$ and $\mathbf{U}$, respectively, one obtains (4.7).

The hardware realization of (4.7) requires n ANDs, $n - 1$ XORs, and a propagation delay of $T_A + \lceil \log_2(n) \rceil T_X$, where $n = \sum_{\tau_i \neq 0}(i+1) + \sum_{\tau_i \neq 0, q_{k,i} \neq 0}(m - k - 1)$ is the upper bound of the number of terms $\left(a_{i-j} b_j\right)$ and $\left(a_{m-j+k} b_j\right)$ in (4.7). It is noted that if τ and $\mathbf{Q}$ have low Hamming weights, then, the computation of $Tr(AB)$ using (4.7) becomes more efficient (in terms of space) than the straight forward method. In what follows, the realization of (4.7) for the WG(29, 11) is derived.

Corollary 4.1.2 *Let $\left\{\alpha^{28}, \ldots, \alpha, 1\right\}$ be the PB of $GF\left(2^{29}\right)$ over $GF(2)$ which is defined by* (4.3). *Then, the trace of the multiplication of two field elements $A = \sum_{i=0}^{28} a_i\alpha^i$ and $B = \sum_{i=0}^{28} b_i\alpha^i$ is computed as follows:*

$$\begin{aligned} Tr(AB) = (a_0 + a_{27}) b_0 + \sum_{j=1}^{25} \left(a_{27-j} + a_{29-j}\right) b_j + \\ (a_1 + a_{26}) b_{28} + \sum_{j=26}^{27} \left(a_{27-j} + a_{29-j} + a_{54-j}\right) b_j. \end{aligned} \tag{4.9}$$

Proof It is noted that τ has only two nonzero components, τ_0 and τ_{27} (see Section 4.1.2.3). The $\mathbf{Q}$ (reduction) matrix for the field polynomial (4.3) have been computed and it has been found that the only nonzero entries in the 1-st and the 28-th columns of this matrix are $q_{0,0}$, $q_{27,0}$, $q_{25,27}$, and $q_{27,27}$. Hence, (4.9) results from substituting these values in (4.7).

It is noted that the realization of (4.9) requires 29 AND and 59 XOR gates with a time delay of $T_A + 6T_X$.

4.1.3 Architecture and FSM

4.1.3.1 Architecture of the WG$(29, 11)$ Cipher

The PB (defined by (4.3)) based architecture of the WG$(29, 11)$, according to the $WGT29$ formulations in (4.1) and (4.2), and the linear recurrence (4.5), is shown in Figures 4.3 and 4.4. In these two figures, $Tr(\bullet)$ generates the trace of a $GF\left(2^{29}\right)$ element (see Section 4.1.2.3). $Tr(\star)$ generates the trace of the multiplication of two $GF\left(2^{29}\right)$ elements using (4.9). Y is the output of the LFSR represented by (4.5). $Y^{2^{10}-1}$ is generated based on (4.2). An arrow represents a register which is inserted for pipelining (see Section 4.1.5). A number n under a register means it is clocked at end of the n-th clock cycle during each computation of the initial feedback in the initialization phase. A zero under a register indicates that the register's clock input is always enabled during the run phase. $r_1 = 2^{10}+1$, $r_2 = 2^{20}+2^{10}+1$, $r_3 = 2^{20}-2^{10}+1$, and $r_4 = 2^{20} + 2^{10} - 1$. The squaring matrices are implemented using the signal reuse constructions, the complexities of which are presented in Table 4.1. The complement operator, i.e. ~, invert the first bit of the input, which requires only one NOT gate. Notice that αB_i, which is required for generating the LFSR feedback signal, is stored in the right most cell of the LFSR (i.e. $B_i^{'}$) as shown in Figure 4.3. This is done to reduce the propagation delay through the LFSR feedback by one multiplier. This construction avoids having the LFSR's critical path constraining the speed of the cipher when pipelining is applied to the transform.

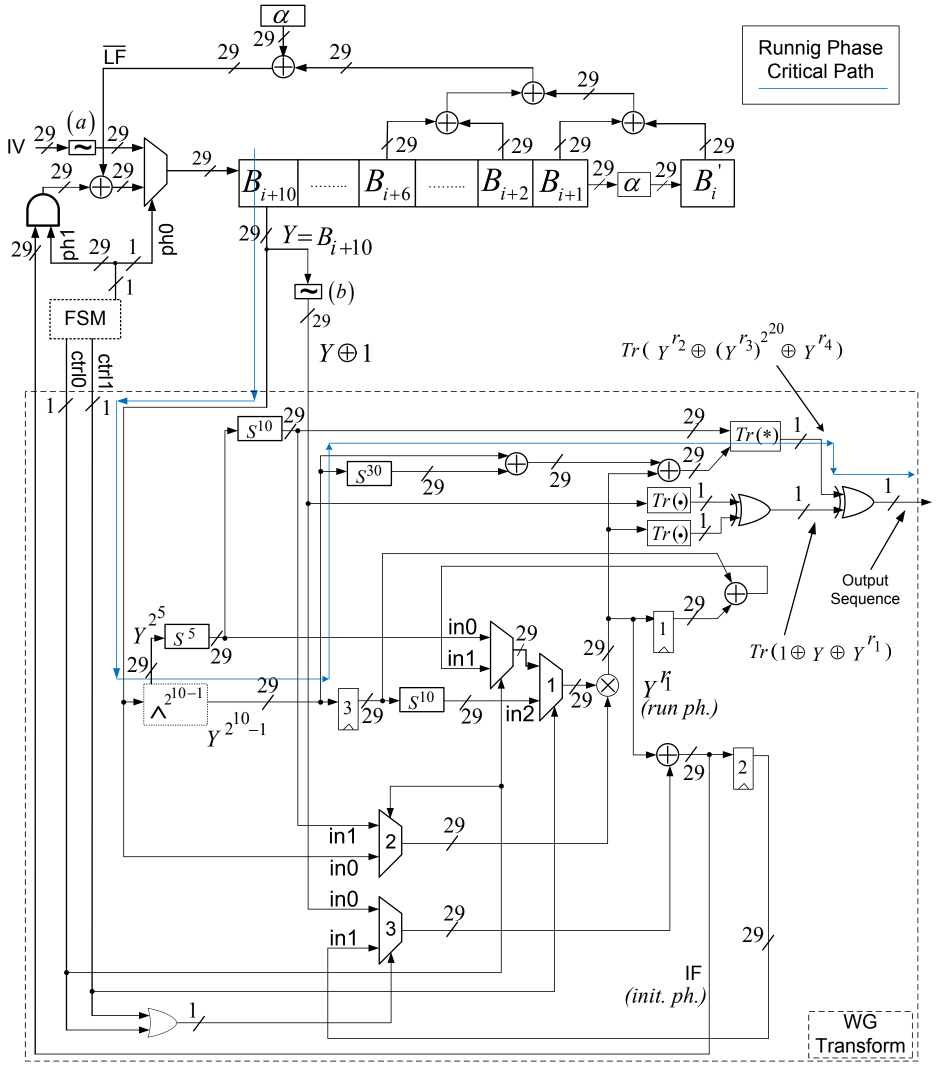


Figure 4.3: Architecture of the WG(29, 11) stream cipher.

The finite state machine (FSM) controls the cipher during three different phases of operation (see Section (4.1.3.2)). During the load phase, the LFSR shifts at each clock cycle, where its leftmost cell is loaded with $1 \oplus IV$ (IV is the initial vector).

It is noted that the initial feedback signal $IF = WGP29$, which is needed for initialization phase, is missing in Figure 4.3. This is a result of computing $WGT29$ according to (4.1) using (4.9). Let $q = 2^{10} - 1$, $r_1 = 2^{10} + 1$, $r_2 = 2^{20} + r_1$, $r_3 = 2^{20} - q$, and $r_4 = 2^{20} + q$. Therefore, the $WGP29$ in (2.9) can be written as $1 \oplus Y \oplus Y^{r_1} \oplus Y^{r_2} \oplus Y^{r_3} \oplus Y^{r_4}$, and is recovered using serial computation over 3 clock cycles as described in Table 4.2. In this table, ctrl0 and ctrl1 are generated by the FSM. $WGP29$ is the next state of Register 2 in stage 3. Rows of the table are listed in order of computation stages (first to last). It is noted that, next state of Register 3 is always Y^q (Figure 4.3). During the initialization phase, the LFSR shifts once every 3 clock

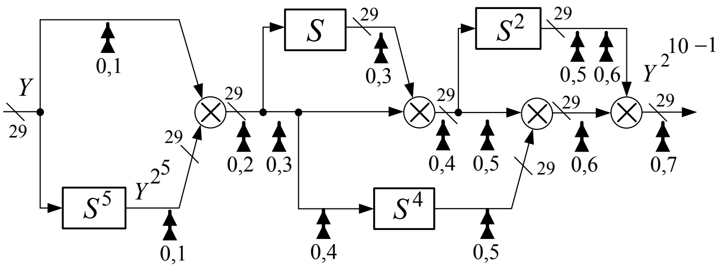


Figure 4.4: Architecture of the $2^{10}-1$ module.

cycles and loads its leftmost cell with $IF \oplus \overline{LF}$, where $\overline{LF} = LF \oplus 1$ and LF is the original linear feedback given by (4.4).

ctrl0	ctrl1	Output			Next State	
		MUX # 1	MUX # 2	MUX # 3	Register 1	Register 2
0	0	$Y^{2^{10}}$	Y	$Y \oplus 1$	Y^{r_1}	$Y^{r_1} \oplus Y \oplus 1$
1	0	$Y^{r_1} \oplus Y^q$	$Y^{2^{20}}$	$Y^{r_1} \oplus Y \oplus 1$	$Y^{r_4} \oplus Y^{r_2}$	$Y^{r_4} \oplus Y^{r_2} \oplus$ $Y^{r_1} \oplus Y \oplus 1$
0	1	$Y^{2^{10}q}$	Y	$Y^{r_4} \oplus Y^{r_2} \oplus$ $Y^{r_1} \oplus Y \oplus 1$	Y^{r_3}	$Y^{r_4} \oplus Y^{r_3} \oplus$ $Y^{r_2} \oplus Y^{r_1} \oplus$ $Y \oplus 1$

Table 4.2: Computation of the $IF = WGP29$ signal over 3 clock cycles during the initialization phase.

In the running phase, the LFSR updates its state in each clock cycle, where B_{i+10} is loaded with $\overline{LF}$. In Figure 4.3, the keystream bits are obtained from XORing $Tr\,(1 \oplus Y \oplus Y^{r_1})$ with $Tr\left(Y^{r_2} \oplus (Y^{r_3})^{2^{20}} \oplus Y^{r_4}\right)$. $Tr\,(1 \oplus Y \oplus Y^{r_1})$ is the result of XORing $Tr\,(1 \oplus Y)$ and $Tr\,(Y^{r_1})$. $Tr\,(1 \oplus Y)$ and $Tr\,(Y^{r_1})$ are produced by applying operator $Tr\,(\bullet)$ to $1 \oplus Y$ and Y^{r_1}, respectively. The operator $Tr\,(\bullet)$ generates its output according to (4.6). Y^{r_1} is generated by multiplying Y with $Y^{2^{10}}$ in $GF\left(2^{29}\right)$ (by setting ctrl0 = ctrl1 = 0 for the running phase in Figure 4.3). Y is the output B_{i+10} of the LFSR and $Y^{2^{10}}$ is obtained from the squarer $\mathbf{S}^5$ operating on Y^{2^5}, which in turn, is available from the generator of $Y^{2^{10}-1}$ (see Figure 4.4). $1 \oplus Y$ is the addition of $Y \in GF\left(2^{29}\right)$ with the unity element $1 = (0, \ldots, 0, 1)$ represented w.r.t. PB. Thus, $1 \oplus Y$ results from inverting the least significant bit of B_{i+10} by the complement operator $\sim$. $Tr\left(Y^{r_2} \oplus (Y^{r_3})^{2^{20}} \oplus Y^{r_4}\right)$ is generated by applying (4.9) to $Y^{2^{20}}$ and $\left(Y^{r_1} \oplus Y^q \oplus Y^{2^{30}q}\right)$. The signal $Y^{2^{20}}$ is the result of $\mathbf{S}^{10}$ operating on $Y^{2^{10}}$. Signal $Y^{r_1} \oplus Y^q \oplus Y^{2^{30}q}$ is the bitwise XOR of Y^{r_1}, Y^q, and $Y^{2^{30}q}$, where $Y^{2^{30}q}$ is obtained from $\mathbf{S}^{30}$ operating on Y^q and Y^q is generated as presented in Figure 4.4.

4.1.3.2 The Finite State Machine (FSM)

The architecture of the FSM is shown in Figure 4.5. The FSM controls the inputs to the LFSR during the three phases of operation through signals ph0 and ph1. As presented in Table 4.3 for the column of Figure 4.3 the loading phase takes 11 clock cycles followed by the initialization phase which stays for 33 + 33 = 66 clock cycles, then starts the run phase. The FSM is built from a 2-bit binary counter, an 11-bit 1-hot counter, and a 3-bit 1-hot counter. The first counter generates ph0 and ph1. The 11-bit counter triggers the clock of the 2-bit counter, every 11 counts, during loading and initialization. The 3-bit counter, generates ctrl0 and ctrl1, and triggers the clock of the 11-bit counter as well as the clock of the LFSR, every 3 counts, during initialization.

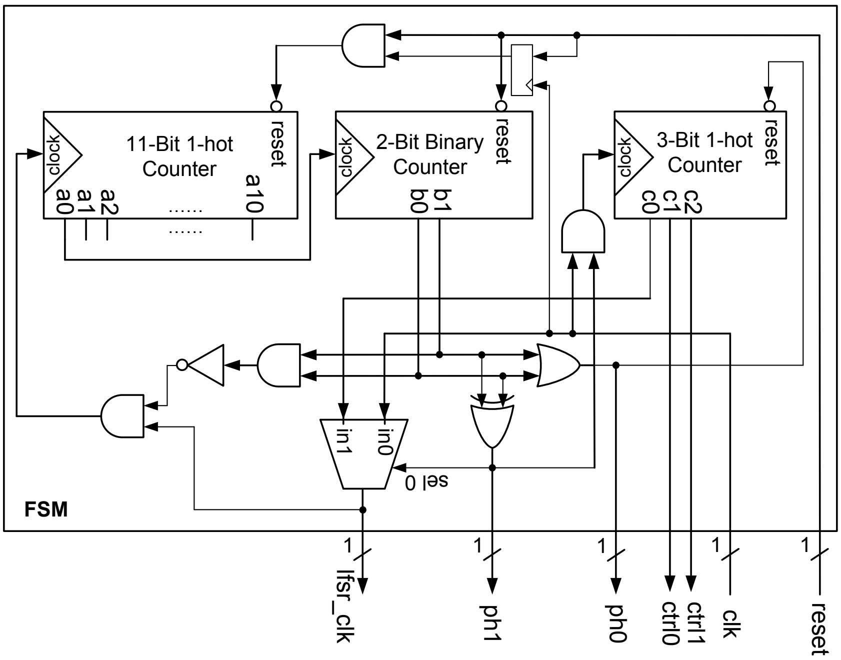


Figure 4.5: FSM for the PB based implementation of the WG(29, 11) stream cipher.

4.1.4 Serialized Implementation of the PB Based WG(29, 11)

4.1.4.1 Architecture of the Serialized WG(29, 11)

Here, a serialized $WGP29/WGT29$ design is presented for area constrained applications. The serial WG(29, 11) which is proposed in this section has the same LFSR, compared to the standard design in Figure 4.3; however, the WG transform and the FSM are modified. Figure 4.6 presents the proposed serial $WGP29/WGT29$ architecture. In this figure, $Y = B_{i+10}$ is the LFSR's output (see Figure 4.3), $r_1 = 2^{10} + 1$, $r_2 = 2^{20} + 2^{10} + 1$, $r_3 = 2^{20} - 2^{10} + 1$, and

2-bit counter		ph1/ph0	phase of operation	Number of Clock Cycles for the Proposed Designs					
				Figure 4.3	Figure 4.6	Figure 4.8	Figure 4.12a	Figure 4.13	Figure 4.15
a_1	a_0								
0	0	0/0	Load	11	11	11	32	32	32
0	1	1/1	Init.	33	88	132	96	288	416
1	0	1/1	Init.	33	88	132	96	288	416
1	1	0/1	Run	-	-	-	-	-	-

Table 4.3: Phase of operation in the proposed PB based WG designs as a function of the state of the 2-bit binary counter.

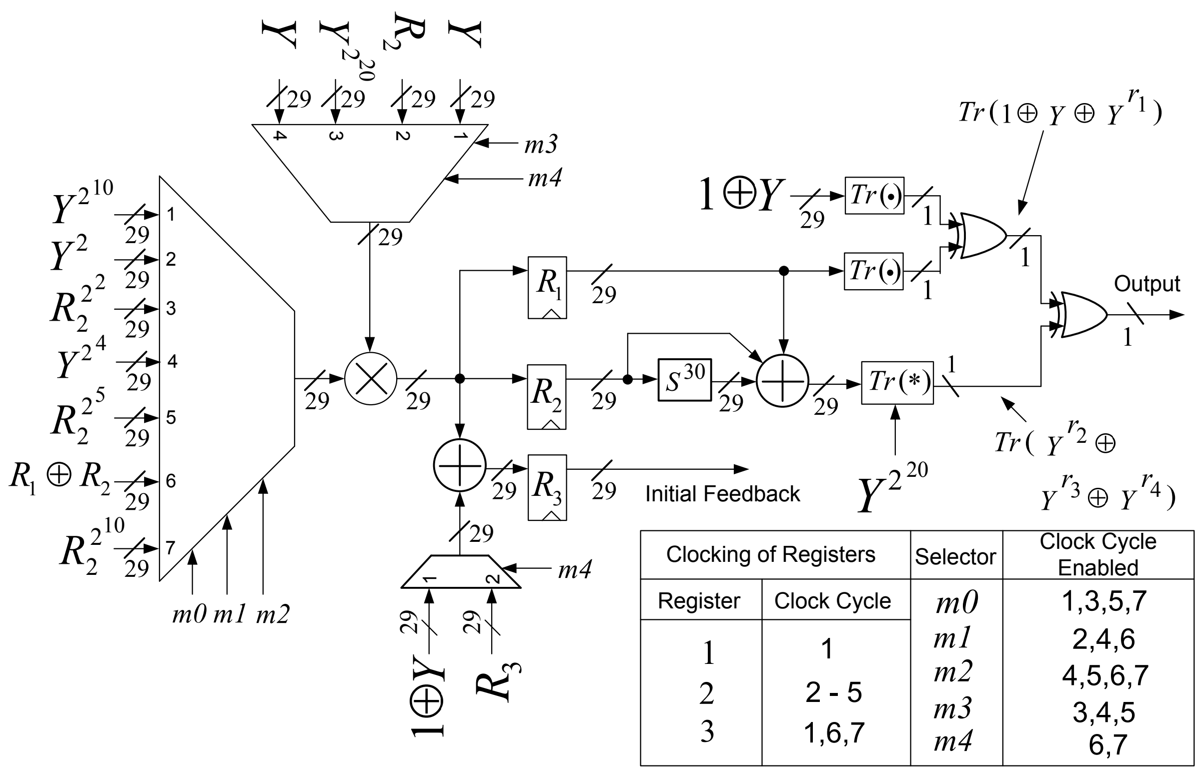
$r_4 = 2^{20} + 2^{10} - 1$. In this architecture, only one multiplier is used. The computations of the different variables used in (4.1) and (4.2) are accomplished sequentially according to Table 4.4.

		Clock Cycle			
		1	2	3	4
Next State	Register 1	Y^{r_1}	Y^{r_1}	Y^{r_1}	Y^{r_1}
	Register 2	-	Y^{2+1}	$Y^{\sum_{i=0}^{3} 2^i}$	$Y^{\sum_{i=0}^{4} 2^i}$
	Register 3	$1 \oplus Y \oplus Y^{r_1}$	$1 \oplus Y \oplus Y^{r_1}$	$1 \oplus Y \oplus Y^{r_1}$	$1 \oplus Y \oplus Y^{r_1}$

		Clock Cycle		
		5	6	7
Next State	Register 1	Y^{r_1}	Y^{r_1}	Y^{r_1}
	Register 2	Y^q	Y^q	Y^q
	Register 3	$1 \oplus Y \oplus Y^{r_1}$	$1 \oplus Y \oplus$ $Y^{r_1} \oplus Y^{r_2} \oplus Y^{r_4}$	$1 \oplus Y \oplus Y^{r_1} \oplus$ $Y^{r_2} \oplus Y^{r_3} \oplus Y^{r_4}$

Table 4.4: Steps for computing the $WGP29$ and $WGT29$ in the serial implementation of the WG$(29, 11)$ design.

It is noted that no changes are required for the loading phase of the serial WG$(29, 11)$. However, in the architecture of Figure 4.6, an initialization round takes 7 clock cycles to generate the $WGP29$ signal. The LFSR is updated at the 8-th clock cycle. During the run phase, a stream bit is produced every 6 cycles. During these two phases, the multiplexers provide the inputs to the multiplier and the adder. The multiplexers' inputs are multiplexed by selectors



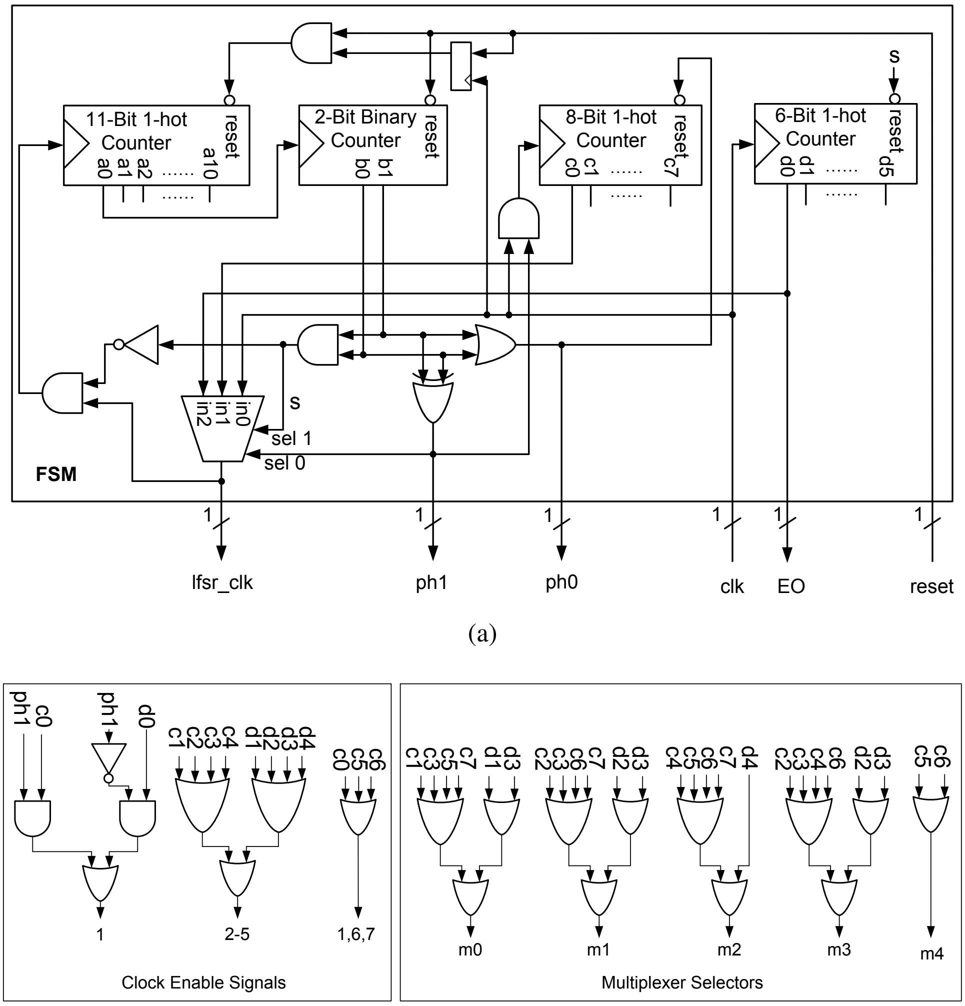
Clocking of Registers		Selector	Clock Cycle Enabled
Register	Clock Cycle		
1	1	m0	1,3,5,7
2	2 - 5	m1	2,4,6
3	1,6,7	m2	4,5,6,7
		m3	3,4,5
		m4	6,7

Figure 4.6: Architecture of the serial $WGP29/WGT29$ implementation.

m_0 - m_4. The 3 registers are clocked as it is specified by the clocking table in Figure 4.6. The clocking of the different registers is enabled by means of clock enable signals (see Section 4.1.4.2). In this design, the lfsr_clk signal in Figure 4.7a is required in order to clock the LFSR once every 1 clock cycle, 8 clock cycles, and 6 clock cycles, during loading, initialization, and run phases, respectively. This means that the initialization phase takes a total of $8 \times 22 = 176$ clock cycles. The number of clock cycles needed for different phases of Figure 4.6 are presented in the corresponding column of Table 4.3. Moreover, the signal EO in Figure 4.7a is used to enable the keystream output every 6 clock cycles during the run phase. These selectors, clock enables, lfsr_clk, and EO signals are generated through the FSM, as it is presented next.

4.1.4.2 FSM for the Serialized PB based WG(29, 11)

Figure 4.7a is a block diagram for the FSM which is used for the serialized PB based WG(29, 11). Also, Figure 4.7b shows the details of generating the Clock Enable Control Signals and the Multiplexers' Selectors. For the clock enable signals, the number at the output of an OR gate indicates the number of the enabled clock cycle during the initialization phase. m_0, m_1, m_2, m_3, and m_4 are the selectors for the multiplexers. The FSM controls the inputs to the LFSR during the three phases of operations. As shown in Table 4.3 for Figure 4.6, the



(a)

(b)

Figure 4.7: a) Architecture of the FSM for the serialized implementation of the WG(29, 11). b) Generating the Clock Enable Control Signals and the Multiplexers' Selectors.

loading phase takes 11 clock cycles followed by the initialization phase which stays for 176 clock cycles, then starts the run phase. The FSM is built from a 2-bit binary counter, an 11-bit 1-hot counter, an 8-bit 1-hot counter, and a 6-bit 1-hot counter. The 2-bit counter generates ph0 and ph1 according to Table 4.3. The 11-bit counter triggers the clock of the 2-bit counter, every 11 counts, during the loading and initialization. The 8-bit counter, generates the clock enable signals and the multiplexers' selectors (see Figure 4.6), and triggers the clock of the 11-bit counter as well as the clock of the LFSR, every 8 counts, during initialization. In the run phase, the 6-bit counter, generates the clock enable signals and the multiplexers' selectors, and triggers the clock of the LFSR, every 6 counts. From the starting of the run phase, the 6-bit counter enables the output of the cipher every 6 counts.

4.1.5 Pipelined Implementation of the PB Based WG$(29, 11)$

4.1.5.1 Architecture of the Pipelined PB Based WG$(29, 11)$

Figures 4.8 and 4.4 present the pipelined version of the PB based implementation of the $WGT29$.

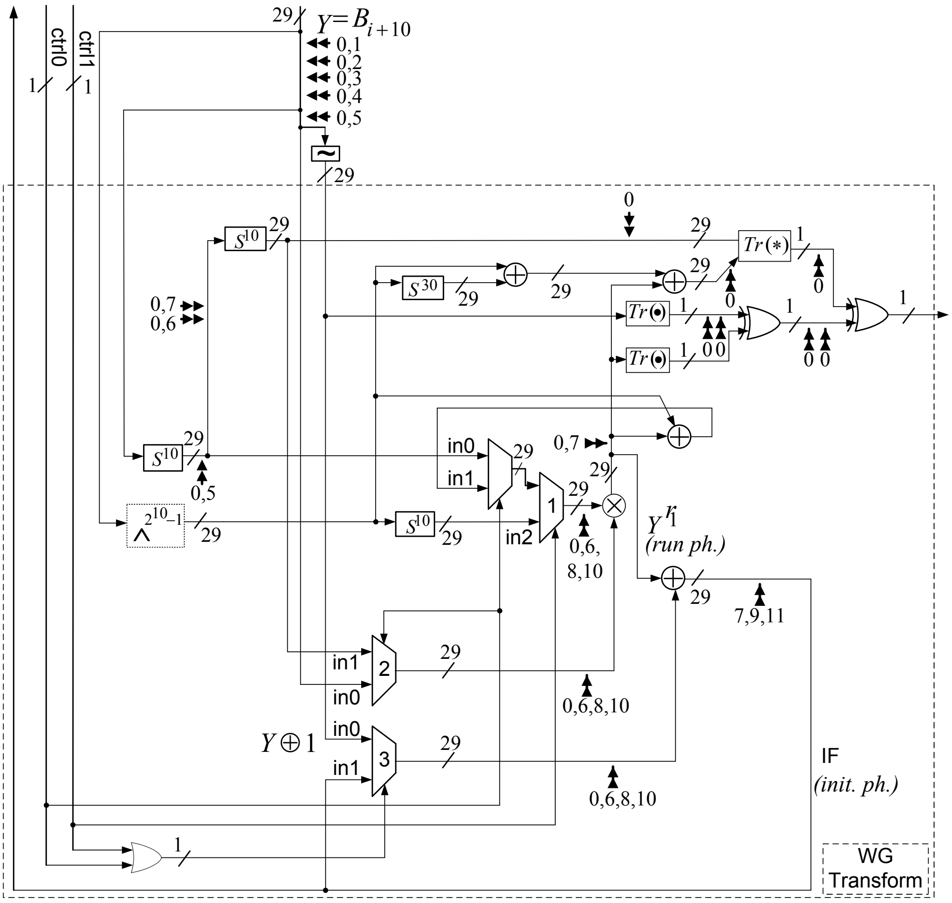


Figure 4.8: Pipelined version of the $WGT29$.

The pipeline has been constructed with 10-stages during the run phase and 12-stages during the initialization phase, in order to achieve a critical path with only one multiplier. In these figures, the double headed arrows point to the locations where the registers are inserted, for the pipeline. The numbers under these arrows indicate the clock cycles, during each initial feedback computation throughout initialization, during which the registers will be clock-enabled. A zero below a register means that its clock input will always be enabled during the run phase. The clocking of the different registers in the transform is controlled by means of clock enable signals (see Section 4.1.5.2).

It is noted that no changes are required for the loading phase. However, during the initialization and the run phases, an input signal now requires 12 and 10 clock cycles, respectively, to propagate to the output of the transform/permutation. Therefore, for the initialization phase, the lfsr_clk signal in Figure 4.9 triggers the LFSR once every 12 cycles. This means that the initialization phase takes a total of $12 \times 22 = 264$ clock cycles as presented in Table 4.3 for Figure 4.8. Also, the multiplexers' outputs in Figure 4.8 are controlled through the signals ctrl0 and ctrl1 (Figure 4.10) during the initialization and the run phases. For the run phase, an output enable signal, EO in Figure 4.9, is used to enable the keystream output after the first 10 clock cycles. The following section presents the FSM and show how the different control signals are derived.

4.1.5.2 FSM for the Pipelined PB Based WG(29, 11)

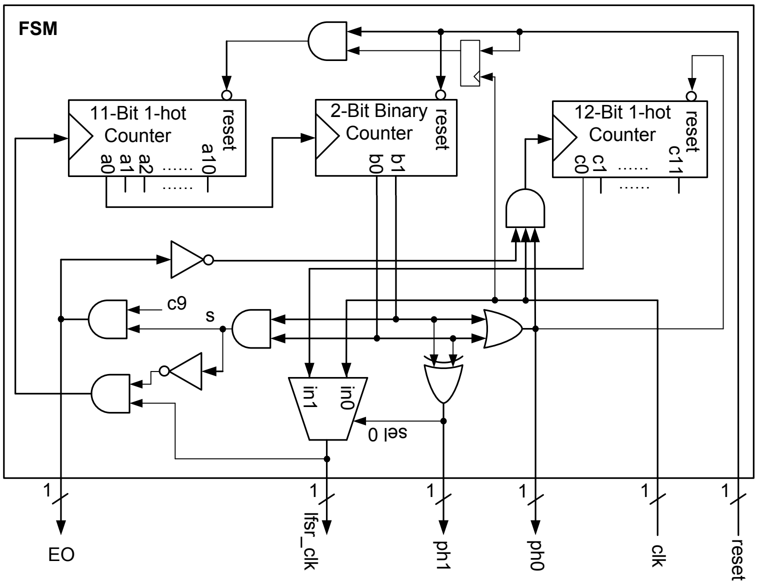


Figure 4.9: Architecture of the FSM for the pipelined version of the WG(29, 11).

Figure 4.9 presents the architecture of the FSM which is used for the pipelined version of the PB based implementation of the WG(29, 11). Figure 4.10 shows the details of generating the clock enable signals and, ctrl0 and ctrl1 signals. The numbers at the output indicate the clock cycles, during each initial feedback computation throughout initialization, during which the register will be clock-enabled. A 0 at the output means the clock input will be always enabled during the run phase. Signals s and c_i, $0 \leq i \leq 11$, are shown in Figure 4.9. Similar to the previously introduced FSMs in this chapter, the FSM controls the inputs to the LFSR during the three phases of operations through generating the signals ph0 and ph1. According to column of Figure 4.8 in Table 4.3, the loading phase takes 11 clock cycles, followed by

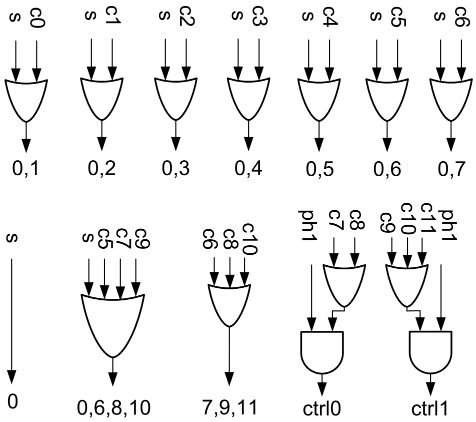


Figure 4.10: Clock enable control signals for the pipelined version of the WG(29, 11).

the initialization phase which stays for 264 clock cycles, followed by the run phase. The FSM is built from a 2-bit binary counter, an 11-bit 1-hot counter, and 12-bit 1-hot counter. The 2-bit counter generates ph0 and ph1 according to Table 4.3. The 11-bit counter triggers the clock of the 2-bit counter, every 11 counts, during the loading and initialization. The 12-bit counter, generates the clock enable signals and the multiplexers' selectors (ctrl0 and ctrl1, see Figure 4.8), and triggers the clock of the 11-bit counter as well as the clock of the LFSR, every 12 counts, during initialization. In the run phase, signal s in Figure 4.9 and the 12-bit counter, generate the clock enable signals and the multiplexers' selectors (fixed at ctrl0=ctrl1=0), respectively. The 12-bit counter enables the output of the cipher after 10 counts from the start of the run phase. The LFSR is triggered with each clock cycle in the run phase.

4.2 Architectures of the WG-16 Stream Cipher

The WG-16 cipher has been proposed by the authors of [34] for securing the 4G's confidentiality and integrity protection schemes against the attack in [90]. The only WG-16 hardware design, which uses NB, is presented in [35]. This design is based on composite field arithmetic and properties of the trace function in the tower field representation.

Here, a new formulation of the WG-16 permutation is proposed. This formulation requires 8 multiplications compared to 10 in the formulation of [35]. Based on this formulation, and using the trace property in (4.7), this section presents six hardware architectures of the WG-16, based on the PB representation for the first time. The six designs include a standard architecture, its serial version, and its pipelined version using two different types of multipliers for each version. The serial version can be used for low-area applications whereas the pipelined one is suitable for high-speed applications. The pipelined instance of the proposed scheme offers almost twice the throughput which is reported by the implementations in [35], at a slightly smaller area. In what follows, the formulation of the WG-16 transform followed by the formulations used for squaring and trace function are derived. In addition, the formulation for direct computation of the trace of the multiplication of two field elements, in the PB, is obtained. Then, the proposed standard architecture of the WG-16 is shown. The section ends by presenting serialized and pipelined versions of the standard design.

4.2.1 Formulations of $WGP16$ and $WGT16$

The $WGP16$'s formulation in (2.14) requires 10 multiplications when the field elements are represented in the PB. In the following, a new formulation is derived which requires 8 multiplications.

Proposition 4.2.1 *The WG permutation of the WG-*16 *stream cipher is computed as follows*

$$\begin{aligned} WGP16 = & 1 \oplus Y \oplus Y^{2^{11}+1} \oplus Y^{2^{11}(2^5-1)+2^6} \\ & \oplus Y^{2^{11}+1}\left(Y^{2^6} \oplus Y^{2(2^5-1)}\right), \end{aligned} \tag{4.10}$$

where $Y = (A_{i+31})^{1057} \oplus 1$*,* A_{i+31} *is the output of the LFSR described by* (2.15)*, and* Y^{2^5-1} *is computed as follows*

$$Y^{2^5-1} = \left(Y^{2^2+1}\right)^{2+1} Y^{2^4}. \tag{4.11}$$

Proof Let $e_1 = 2^{11} + 1$, $e_2 = 2^{11} + 2^6 + 1$, $e_3 = -2^{11} + 2^6 + 1$, and $e_4 = 2^{11} + 2^6 - 1$ in (2.13). By noticing that $e_3 + 2^{16} - 1 \equiv e_3 \pmod{2^{16} - 1}$, then, one obtains

$$e_2 = e_1 + 2^6, \quad e_3 = 2^{11}s + 2^6, \quad e_4 = e_1 + 2s,$$

where $s = 2^5 - 1$, and the proof is completed by taking Y^{e_1} as a common factor between Y^{e_2} and Y^{e_4}.

The WG transform is obtained by taking the trace of (4.10). Equation (4.10) requires 8 $GF\left(2^{16}\right)$ multiplications: 1 for computing $Y^{2^{11}+1}$, 3 for computing Y^{2^5-1}, 1 for computing $Y^{2^{11}(2^5-1)+2^6}$, 1 for computing $Y^{2^{11}+1}\left(Y^{2^6} \oplus Y^{2(2^5-1)}\right)$, and 2 for computing $1 \oplus Y = (A_{i+31})^{1057}$. In addition to this, (4.10) requires 7 squarings and 5 $GF\left(2^{16}\right)$ additions. For the transform, the computation of the trace of $WGP16$ is required. Section 4.2.3 presents a method which reduces the number of multiplications in the $WGT16$ to only 6 through computing $Tr\left(Y^{2^{11}(2^5-1)}Y^{2^6}\right)$ directly from $Y^{2^{11}(2^5-1)}$ and Y^{2^6}, and $Tr\left(Y^{(2^{11}+1)}\left(Y^{2^6} \oplus Y^{2(2^5-1)}\right)\right)$ directly from $Y^{(2^{11}+1)}$ and $Y^{2^6} \oplus Y^{2(2^5-1)}$, without performing the multiplications.

4.2.2 Squaring Matrices and Trace Vector

Similar to the WG$(29, 11)$, in what follows, the squaring matrices and the trace vector for the field polynomial (2.16) are presented.

4.2.2.1 Squaring Matrices

Figure 4.11 shows the squaring matrix $\mathbf{S}$ for the field polynomial (2.16). One can find the required squaring operations for the WG-16 permutation from (4.10) and (4.11). Table 4.5 lists the space and propagation delay complexities of the different squaring matrices used in the WG-16 implementation (before and after signal reuse). In this table, PD denotes propagation delay.

$$\begin{pmatrix}
0&1&0&0&0&0&0&1&0&0&0&1&1&1&1&1\\
1&1&0&1&0&0&0&0&0&1&0&1&1&0&1&0\\
1&0&1&1&0&1&0&0&0&0&0&0&0&0&0&0\\
0&0&1&0&1&1&0&1&0&0&0&0&0&0&0&0\\
0&0&0&0&1&0&1&1&0&1&0&0&0&0&0&0\\
0&0&0&0&0&0&1&0&1&1&0&1&0&0&0&0\\
0&0&0&0&0&0&0&0&1&0&1&1&0&1&0&0\\
0&0&0&0&0&0&0&0&0&0&1&0&1&1&0&1\\
0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1
\end{pmatrix}$$

Figure 4.11: The matrix $\mathbf{S}$ for WG-16.

	No Sig. Reuse		Sig. Reuse			No Sig. Reuse		Sig. Reuse	
	XOR	PD	XOR	PD		XOR	PD	XOR	PD
$\mathbf{S}$	30	$3T_X$	21	$3T_X$	$\mathbf{S}^6$	99	$4T_X$	63	$4T_X$
$\mathbf{S}^2$	82	$3T_X$	45	$3T_X$	$\mathbf{S}^9$	89	$4T_X$	58	$4T_X$
$\mathbf{S}^4$	103	$4T_X$	64	$4T_X$	$\mathbf{S}^{10}$	102	$4T_X$	60	$4T_X$
$\mathbf{S}^5$	89	$4T_X$	58	$4T_X$	$\mathbf{S}^{11}$	115	$4T_X$	62	$4T_X$

Table 4.5: Space and propagation delay complexities of the different squaring matrices used in the WG-16.

4.2.2.2 Trace Vector

The trace vector for the PB $\left\{\alpha^{15}, \ldots, \alpha, 1\right\}$ defined by (2.16) is $\tau = (\tau_0, \ldots, \tau_{15})$ where $\tau_i = 1$ for $i \in \{11, 13\}$ and $\tau_i = 0$ otherwise (see Section 4.1.2.3). Thus, for $A \in GF\left(2^{16}\right)$

$$Tr(A) = a_{11} + a_{13}. \tag{4.12}$$

4.2.3 Trace of the Multiplication of Two Field Elements for the PB Based WG-16

The following is the realization of (4.7) when applied to WG-16.

Corollary 4.2.2 *Consider the $GF\left(2^{16}\right)$ defined by* (2.16) *where $\left\{\alpha^{15}, \ldots, \alpha, 1\right\}$ is its PB. Then, the trace of the multiplication of two field elements $A = \sum_{i=0}^{15} a_i\alpha^i$ and $B = \sum_{i=0}^{15} b_i\alpha^i$ is computed as follows:*

$$\begin{aligned}Tr(AB) = &\sum_{j=0}^{11}\left(a_{11-j} + a_{13-j}\right)b_j + \sum_{j=12}^{13} a_{13-j}b_j + \\ &\sum_{j=7}^{9} a_{22-j}b_j + \left(a_{12} + a_{15}\right)b_{10} + \\ &\sum_{j=11}^{13}\left(a_{22-j} + a_{25-j} + a_{26-j}\right)b_j + \\ &\sum_{j=14}^{15}\left(a_{22-j} + a_{25-j} + a_{26-j} + a_{29-j}\right)b_j.\end{aligned} \tag{4.13}$$

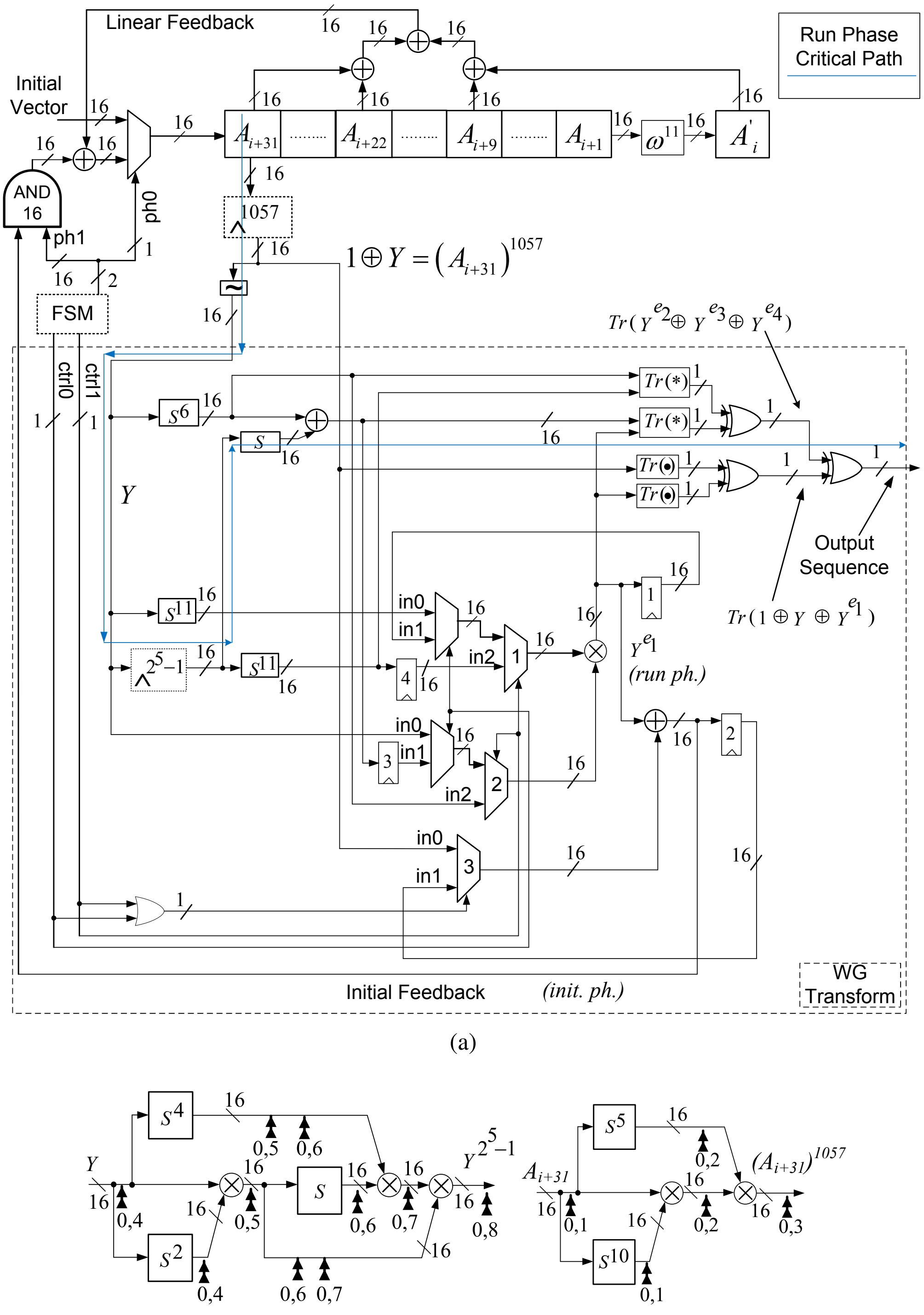
Proof τ has only two nonzero components, τ_{11} and τ_{13} (see Subsection 4.2.2.2). By computing the **Q** (reduction) matrix for the field polynomial (2.16), one finds that the only nonzero entries for the 12-th and the 14-th columns of this matrix are $q_{6,11}$, $q_{8,11}$, $q_{9,11}$, $q_{11,11}$, $q_{8,13}$, $q_{10,13}$, $q_{11,13}$, and $q_{13,13}$. Hence, by replacing these values of τ_i and $q_{k,i}$ in (4.7), one gets (4.13).

It is noted that the realization of (4.13) requires 23 AND and 47 XOR gates and introduces a propagation delay of $T_A + 7T_X$.

4.2.4 Architecture and FSM

4.2.4.1 Architecture of the WG-16 Cipher

Let $e_1 = 2^{11} + 1$, $e_2 = 2^{11} + 2^6 + 1$, $e_3 = -2^{11} + 2^6 + 1$, $e_4 = 2^{11} + 2^6 - 1$, and $s = 2^5 - 1$. Figures 4.12a , 4.12b, and 4.12c present the proposed architecture of the WG-16 according to the *WGP*16 formulations in (4.10) and (4.11), and the linear recurrence (2.15), based on the PB defined by (2.16).



(a)

(b)

(c)

Figure 4.12: a) Architecture of the WG-16. b) Generation of the signal Y^s ($s = 2^5 - 1$). c) Generation of the signal $(A_{i+31})^{1057}$.

In Figure 4.12a, $Tr(\bullet)$ generates the trace of a $GF\left(2^{16}\right)$ element. $Tr(\star)$ generates the trace of the multiplication of two $GF\left(2^{16}\right)$ elements. Figure 4.12b shows the used architecture for generation of the signal Y^s ($s = 2^5 - 1$). Figure 4.12c shows the architecture for the generation of the signal $(A_{i+31})^{1057}$. The squaring matrices in the three figures are implemented using the signal reuse constructions of Table 4.5. In figures 4.12b, and 4.12c, a double-headed arrow points to the location where a register is inserted for pipelining purposes (see Section 4.2.6.1).

In Figure 4.12a, the FSM controls the components of the cipher during the different phases of operation. This is accomplished through signals lfsr_clk, ph0, ph1, ctrl0 and ctrl1 (see Section 4.2.4.2 for details).

During the load phase, the LFSR shifts at each clock cycle while its leftmost cell is loaded through the Initial Vector input.

It is noted that the signal $Y^{e_2} \oplus Y^{e_4}$ is missing in Figure 4.12a. This is due to the generation of $Tr(Y^{e_2} \oplus Y^{e_4})$ directly from $Y^{2^{11}+1}$ and $\left(Y^{2^6} \oplus Y^{2s}\right)$ using (4.13). As a result, the Initial Feedback ($WGP16$) signal, which is needed for the initialization phase, does not exist. This is recovered by generating $WGP16$ over 3 clock cycles, during initialization, as presented in Table 4.6. In this table, the control signals ctrl0 and ctrl1 are generated by the FSM. $WGP16$ is the next state of Register 2 in stage 3. Rows are listed in order of computation stages (first to last). It is noted that, next state of Register 4 in Figure 4.12a is always Y^s. During the initialization phase, the lfsr_clk signal triggers the LFSR every 3 clock cycles. The leftmost cell is loaded with the result from the field addition of the LFSR feedback and $WGP16$ (Initial Feedback).

$^{ctrl0}/_{ctrl1}$	Output			Next State		
	MUX # 1	MUX # 2	MUX # 3	Register 1	Register 2	Register 3
0/0	Y	$Y^{2^{11}}$	$Y \oplus 1$	Y^{e_1}	$Y^{e_1} \oplus Y \oplus 1$	$Y^{2^6} \oplus Y^{2s}$
1/0	Y^{e_1}	$Y^{2^6} \oplus Y^{2s}$	$Y^{e_1} \oplus Y \oplus 1$	$Y^{e_2} \oplus Y^{e_4}$	$Y^{e_4} \oplus Y^{e_2} \oplus Y^{e_1} \oplus Y \oplus 1$	$Y^{2^6} \oplus Y^{2s}$
0/1	Y^{2^6}	$Y^{2^{11}s}$	$Y^{e_4} \oplus Y^{e_2} \oplus Y^{e_1} \oplus Y \oplus 1$	Y^{e_3}	$Y^{e_4} \oplus Y^{e_3} \oplus Y^{e_2} \oplus Y^{e_1} \oplus Y \oplus 1$	$Y^{2^6} \oplus Y^{2s}$

Table 4.6: Computation of the $WGP16$ signal over 3 clock cycles.

In the running phase, the LFSR updates its state at each clock cycle. The only feedback is the LFSR feedback. The keystream bits are obtained by XORing the signals $Tr(1 \oplus Y)$, $Tr(Y^{e_1})$, $Tr(Y^{e_3})$, and $Tr(Y^{e_2} \oplus Y^{e_4})$. $Tr(1 \oplus Y)$ and $Tr(Y^{e_1})$ are produced from $1 \oplus Y$ and Y^{e_1} using (4.12). Y^{e_1} is generated by multiplying Y with $Y^{2^{11}}$ in $GF\left(2^{16}\right)$. Y is generated by complementing the least significant bit of $(A_{i+31})^{1057}$, and $Y^{2^{11}}$ is obtained from the squarer $\mathbf{S}^{11}$ operating on Y. $1 \oplus Y$ is simply $(A_{i+31})^{1057}$. $Tr(Y^{e_3})$ is generated by applying (4.13) to $Y^{2^{11}(2^5-1)}$

and Y^{2^6}. The signal Y^{2^6} is the result of $\mathbf{S}^6$ operating on Y. The signal $Y^{2^{11}(2^5-1)}$ is the result of $\mathbf{S}^{11}$ operating on Y^{2^5-1}. $Tr\left(Y^{e_2} \oplus Y^{e_4}\right)$ is generated by applying (4.13) to $Y^{2^{11}+1}$ and $\left(Y^{2(2^5-1)} \oplus Y^{2^6}\right)$. The signal $Y^{2(2^5-1)}$ is the result of $\mathbf{S}$ operating on Y^{2^5-1}, while signal $Y^{2(2^5-1)} \oplus Y^{2^6}$ is the bitwise XOR of $Y^{2(2^5-1)}$ and Y^{2^6}.

4.2.4.2 The Finite State Machine

The FSM for the PB based WG-16 is similar to the one used for the PB based implementation of the WG(29, 11) (see Section 4.1.3.2). However, the WG-16's FSM replaces the 11-bit 1-hot counter with a 5-bit binary counter and, the clocking of the 2-bit binary counter occurs after a complete 32 counts for the 5-bit counter. As can be seen from column of Figure 4.12a in Table 4.3, the loading phase takes 32 clock cycles. This is followed by the initialization phase which stays for 192 clock cycles, where each initialization round is extended to 3 clock cycles (for computing $WGP16$) by means of the 3-bit 1-hot counter. During this phase, the LFSR is clocked 64 times, once every 3 clocks, by means of the 3-bit 1-hot counter. After this starts the run phase. Also, the 3-bit counter controls the multiplexers' selectors, ctrl0 and ctrl1, during initialization and run phases.

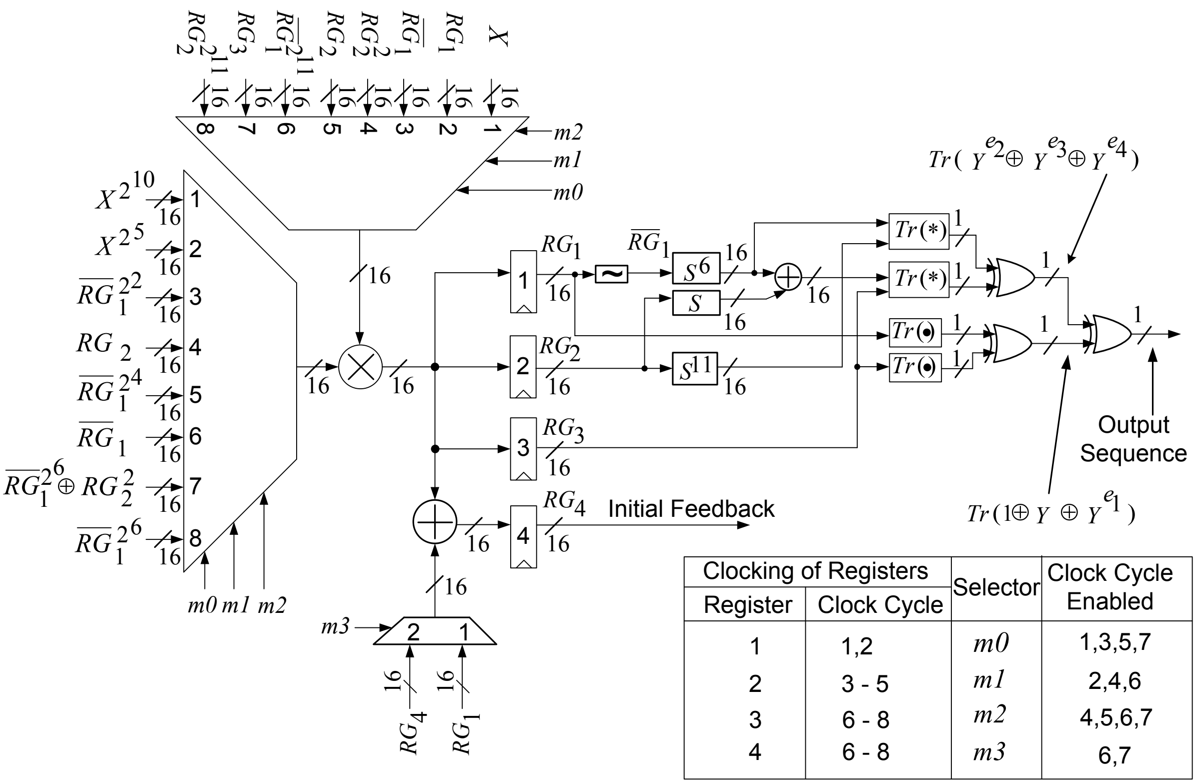
4.2.5 Serialized Implementation of the PB Based WG-16

4.2.5.1 Architecture of the Serialized WG-16

The serialized computation of the WG-16 transform results in a lower space complexity, compared to the standard design in Figure 4.12a. Figure 4.13 presents the proposed architecture for the serial WG-16.

In this architecture, $X = A_{i+31}$ and $Y = 1 \oplus X^{1057}$. The $WGP16$ is computed over 8 cycles (initialization phase) while the $WGT16$ is computed over 6 cycles (run phase). The design uses only one field multiplier. The computations are accomplished according to Table 4.7.

It is noted that no changes are required for the loading phase, as a result of applying the serial computation. In this architecture, an initialization round takes 8 clock cycles to generate the $WGP16$ signal. The LFSR is updated at the 9-th clock cycle. During the run phase, a stream bit is produced every 6 cycles. During these two phases, the multiplexers provide the inputs to the multiplier and adder. The multiplexers' inputs are multiplexed through selectors m_0 - m_3 during computations. The 4 registers are clocked as it is specified by the clocking table in Figure 4.13. The clocking of the different registers is enabled by means of clock enable signals (see Section 4.2.5.2). In this design, the FSM's signal lfsr_clk is required in order to clock the LFSR once every 1 , 9 , and 6 clock cycles, during loading, initialization, and run phases,



Clocking of Registers		Selector	Clock Cycle Enabled
Register	Clock Cycle		
1	1,2	*m0*	1,3,5,7
2	3 - 5	*m1*	2,4,6
3	6 - 8	*m2*	4,5,6,7
4	6 - 8	*m3*	6,7

Figure 4.13: Architecture of the serial implementation for the PB based design of the WG-16.

respectively. This means that the initialization phase takes a total of $9 \times 64 = 576$ cycles. Moreover, an output enable signal EO is used to enable the keystream output every 6 cycles during the run phase. These selectors, clock enables, lfsr_clk, and EO signals are generated through the FSM, as it is presented next.

4.2.5.2 FSM for the Serialized WG-16

The FSM for the serialized WG-16 is a modified version of the one in Section 4.1.4.2. The FSM of the serial WG-16 is obtained by replacing the 11-bit 1-hot counter with a 5-bit binary counter and the 8-bit 1-hot counter with a 9-bit 1-hot counter. The 2-bit binary counter generates ph0 and ph1, and is clocked once every 32 counts from the 5-bit binary counter. As it is shown in column of Figure 4.13 in Table 4.3, the initialization phase takes 576 clock cycles. Each initialization round takes 9 clock cycles. The LFSR is clocked at the arrival of the 9-th clock cycle by means of the 9-bit 1-hot counter. During the run phase, the LFSR is clocked once every 6 clock cycles by means of the 6-bit counter. The cipher's output is enabled once every 6 clock cycles during the run phase, through the 6-bit counter. The clock enable signals which control the clocking of the registers in Figure 4.13, and the multiplexers' selectors m, m_1, m_2, and m_3 are derived from the signal ph1, the outputs of the 9-bit 1-hot counter (initialization),

		Clock Cycle			
		1	2	3	4
Next State	Register 1	$X^{2^{10}+1}$	X^{1057}	X^{1057}	X^{1057}
	Register 2	-	-	Y^{2^2+1}	$Y^{\sum_{i=0}^{3} 2^i}$
	Register 3	-	-	-	-
	Register 4	-	-	-	-
		5	6	7	8
Next State	Register 1	X^{1057}	X^{1057}	X^{1057}	X^{1057}
	Register 2	Y^s	Y^s	Y^s	Y^s
	Register 3	-	Y^{e_1}	$Y^{e_2} \oplus Y^{e_4}$	Y^{e_3}
	Register 4	-	$1 \oplus Y \oplus Y^{e_1}$	$1 \oplus Y \oplus Y^{e_1} \oplus$ $Y^{e_2} \oplus Y^{e_4}$	$1 \oplus Y \oplus Y^{e_1} \oplus$ $Y^{e_2} \oplus Y^{e_3} \oplus Y^{e_4}$

Table 4.7: Computing $WGP16$ and $WGT16$ in the serial implementation of WG-16.

and outputs of the 6-bit 1-hot counter (run phase), as it is shown in Figure 4.14. In the figure,

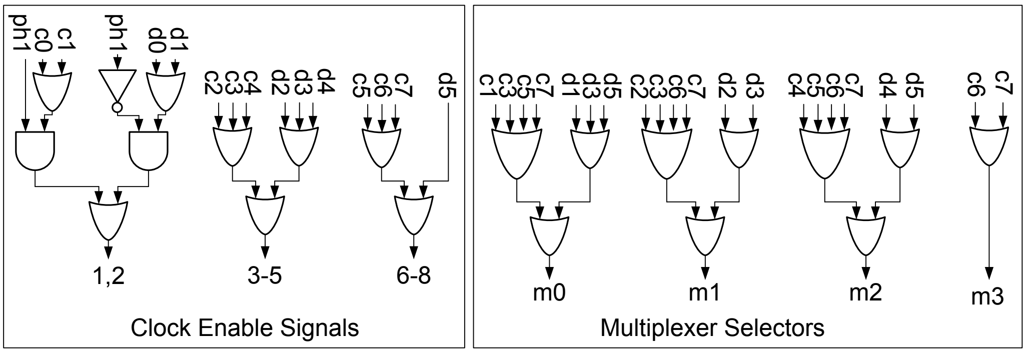


Figure 4.14: Generating the Clock Enable Control Signals and the Multiplexers' Selectors for the serial version of the WG-16.

for the clock enable signals, the number at the output of an OR gate indicates the number of the enabled clock cycle during the initialization phase. Signal s is shown in Figure 4.7a, while signals c_i, $0 \leq i \leq 8$ and d_i, $0 \leq i \leq 5$, are the outputs of the 9-bit and the 6-bit 1-hot counters, respectively.

4.2.6 Pipelined Implementation of the PB Based WG-16

4.2.6.1 Architecture

A pipelined version of the PB based implementation of the WG-16 is presented in Figures 4.15, 4.12b, and 4.12c. The critical path of this architecture has only one multiplier. This is accomplished through a pipeline which has 11-stages during the run phase and 13-stages during the initialization phase. In these figures, the double headed arrows point to the locations where the registers are inserted, for the pipeline. Also, the numbers under an arrow specify the corresponding clock cycles which trigger it during each *WGP*16 computation throughout the initialization phase (13 clock cycles for each computation). A zero under an arrow indicates that the register is enabled during the run phase. The clocking of the different registers in the transform is controlled by means of clock enable signals (see Section 4.2.6.2).

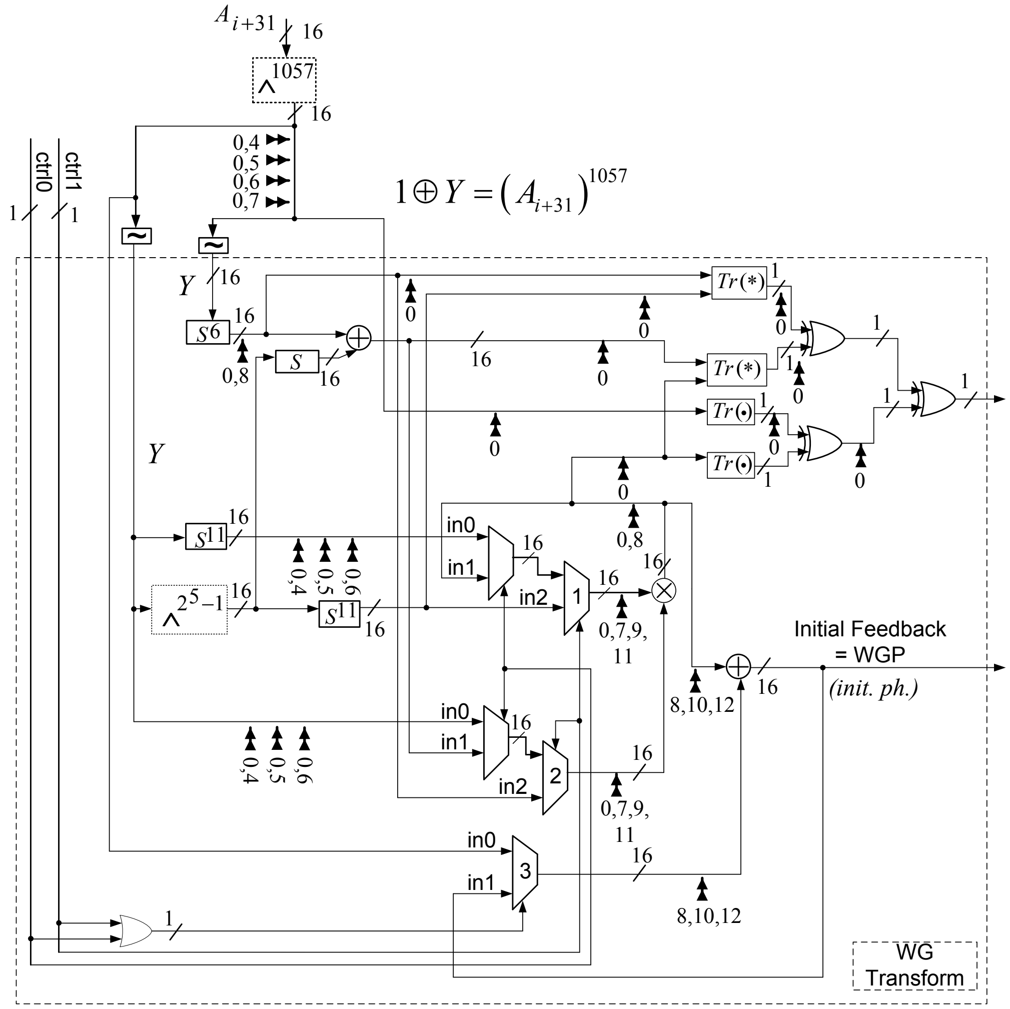


Figure 4.15: Pipelined version of the WG-16 transform.

No changes are required for the loading phase. During the initialization and the run phases, an input signal requires 13 and 11 clock cycles, respectively, to propagate to the output of the transform/permutation. Therefore, for the initialization phase, the lfsr_clk signal triggers the LFSR once every 13 cycles. This means that the initialization phase takes a total of $13 \times 64 = 832$ clock cycles. Also, the multiplexers' outputs in Figure 4.15 are controlled through signals ctrl0 and ctrl1 during the initialization and the run phases. For the run phase, an output enable signal EO is used to enable the keystream output after the first 11 clock cycles. The following section presents the FSM and show how the different control signals are derived.

4.2.6.2 FSM for the Pipelined WG-16

The FSM for the pipelined version of the WG-16 is obtained from the one introduced in Section 4.1.5.2, where a 5-bit binary counter and a 13-bit 1-hot counter replace the 11-bit 1-hot counter and the 12-bit 1-hot counter, respectively. The 2-bit binary counter is clocked once every time the 5-bit binary counter completes 32 counts, during load and initialization. From column of Figure 4.15 in Table 4.3, the loading phase takes 32 clock cycles followed by the initialization phase which stays for 832 clock cycles, then starts the run phase. The 13-bit 1-hot counter expands the initialization phase to a total of 832 clock cycles. At the end of each computation of the $WGP16$ (13 clock cycles), the LFSR is shifted once. The $WGP16$ computations are controlled through the two signals ctrl0 and ctrl1, which are generated by the 13-bit counter (Figure 4.16). Signal ctrl0 is set during clock cycles 8 and 9, while signal ctrl1 is set during clock cycles 10, 11, and 12. These two signals always reset throughout the run phase. Signals ph0 and ph1 select the LFSR's input. These are derived from the output of the 2-bit binary counter according to Table 4.3. After 11 clock cycles from the start of the run phase, the output of the cipher is enabled by means of the 13-bit counter. The clock enable signals are derived from the outputs of the 13-bit counter during initialization and from the outputs of the 2-bit counter (signal s is shown in Figure 4.9) during the run phase, as can be seen from Figure 4.16. In this figure, signals c_i, $0 \le i \le 12$ and d_i, $0 \le i \le 5$, are the outputs of the 13-bit and the 6-bit 1-hot counters, respectively.

4.3 Implementation Results and Comparisons

This section presents speed and area results based on ASIC implementations for the nine different proposed designs. The space and speed trade offs concerning the standard, pipelined, and serial versions of the proposed PB based WG(29, 11) and WG-16 designs, are examined and compared to the counterparts.

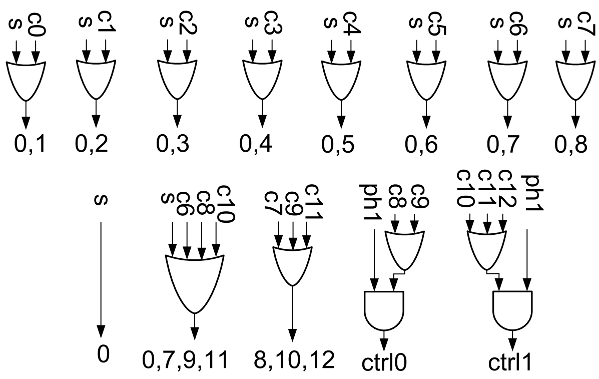


Figure 4.16: Generating the clock enable signals and, ctrl0 and ctrl1 signals for the pipelined version of the WG-16.

4.3.1 ASIC Implementations

Table 4.8 presents the speed and area readings for the nine WG designs which have been proposed, based on the ASIC implementations. In this table, GE denotes Gate Equivalence in terms of number of NAND gates and TP denotes the throughput. The ASIC implementations provide speed and area results for the 65nm CMOS technology with medium effort for optimizations using Synopsys Design Vision [4]. The results are based on Design Vision's estimate of area and clock speed prior to place-and-route. The PB realizations are accomplished using the multiplier presented in [72] for both the WG(29, 11) and the WG-16. The WG-16 has been also realized using the Karatsuba multiplier [53]. We use the VHDL implementations presented in [28] for these two multipliers. Table 4.8 presents the area and speed results for the ASIC implementations of the different designs. The results for the hardware design of the WG(29, 11) which is proposed in [56] are based on theoretical analysis. In addition, the results for the WG(29, 11) design in [68] are reported in previous chapter of this thesis. For the WG-16 which is presented in [35], the results are reported for post place and route.

4.3.2 Results and Comparisons

As shown in Table 4.8, the space complexity of the proposed standard WG(29, 11) is reduced, w.r.t the ones previously presented in [68] and the previous chapter, and the normalized throughput is improved. While the proposed standard WG(29, 11) design shows higher throughput compared to the one in [68], it reports a slightly lower throughput compared to the type-II ONB based design presented in the previous chapter. The WG design presented in [56] requires a number of ROM bits which is exponential in m (the dimension of the binary extension field). For the WG(29, 11), this realization requires 2^{29}-bits of ROM in addition to

9000 XORs and 319 registers, as can be seen from Table 4.8. On the other hand, the space complexities of the proposed designs are based on the area of the multiplier, which is quadratic in m. For high speed applications, the throughput which is reported in Table 4.8 for the proposed pipelined version of the PB based WG$(29, 11)$ design is almost 4.5 times compared to the proposed standard one. This comes at an expense of almost 23% increase in the space complexity. On the other hand, for area constrained applications, the serial version shows up to 59% decrease in the space complexity compared to the standard design, according to the results in Table 4.8. This comes at the expense of reducing the throughput to the half.

In Table 4.8, the Karatsuba based PB implementations of the standard, pipelined, and serial WG-16 show optimal readings for throughput, space, and normalized throughput, compared to the same realizations using the multiplier in [72]. In the same table, in comparison with the pipelined WG-16 implementations presented in [35], the proposed pipelined PB based WG-16 demonstrates almost 2.5 times the throughput with even less space complexity. In addition, for low area applications, the serial version shows up to 42% decrease in the space complexity compared to the standard design. This comes at an expense of around 40% decrease in throughput. On the other hand, for high speed requirements, the pipelined version of the PB based WG-16 design increases the throughput by almost 7 times compared to the standard one. This comes at an expense of almost 33% increase in space complexity.

Moreover, for WG-16, which is proposed by the authors of [34] to overcome the security flaws in the LTE integrity protocols [90], the reported results of the proposed design in Table 4.8 clearly show that the different realizations offer bit rates greater than 100 Mbps and, hence, satisfy the LTE's peak bit rate requirements [49]. Although SNOW 3G [54] and ZUC [10] show better normalized throughput readings compared to our WG-16 designs in Table 4.8, the reported space complexities for the proposed WG-16 (specially, serial instances) are competitive to SNOW 3G and ZUC. Hence, WG-16 is an interesting, low area, candidate for the 4G domain. Table 4.8 also lists the 1-bit output versions of Grain and Trivium which show better performances compared to the proposed designs of the WG-16. On the other hand, our pipelined version of the WG-16 has higher throughput, and normalized throughput, while our serial WG-16 instance shows a very close area complexity, compared to Mickey128.

If even higher throughput is demanded, one can apply the unfolding technique which is presented in [26] to the proposed pipelined WG$(29, 11)$ and WG-16. In this technique, by implementing multiple transforms, the throughput will increase proportionally. Digit-level field multipliers [66] can be considered if lower area is demanded; however, at the expense of adding more cycles for each multiplication.

4.4 Conclusion

This chapter proposed for the first time new architectures for efficient computations of the WG stream ciphers using polynomial basis. The proposed architectures require fewer multiplication operations as compared to the WG counterparts. Moreover, an area efficient method for the direct computation of the trace of the multiplication of two $GF(2^m)$ elements have been derived. Unlike the trace method presented in previous chapter which applies only to type-II ONB, the trace method proposed in this chapter applies to any PB. Based on the proposed trace properties, two classes of PB based designs (standard architecture) have been proposed, one for the WG(29, 11) stream cipher and the other one for the WG-16 stream cipher. In addition, a serialized version and a pipelined version, has been proposed for each of the proposed standard designs.

Nine different proposed designs have been realized through ASIC implementations using the 65nm CMOS technology. The ASIC implementations show that the proposed PB based WG(29, 11) design achieves better area and normalized throughput results compared to all WG(29, 11) counterparts which use NB. Also, it has been shown that the proposed pipelined PB based WG-16 provides almost double the throughput which is offered by the implementations presented in [35], at even smaller area. In addition, the throughput readings reported for the different designs of the WG-16 stream cipher meet the requirements for the peak bit rate specifications of the 4G mobile technology.

Based on these results, the proposed WG(29, 11) and WG-16 designs using PB are competitive candidates, compared to the previously proposed implementations, for securing mobile and communication systems [23, 11, 7]. Specifically, the proposed WG-16 designs are promising for the 4G communications where the guaranteed randomness properties and security aspects are of significant importance.

Implementation	Basis	WG Transform Architecture	Multiplier	Technology	GE	Speed (MHz)	TP $(Mbps)$	Normalized Throughput $(^{Kbps}/_{Gate})$
SNOW 3G [9]	-	-	-	90nm	34000	-	1900	55.88
SNOW 3G [54]	-	-	-	130nm	25016	249	7900	315.97
ZUC [10]	-	-	-	65nm	10000	-	1500	150
Grain128 (1-bit output version) [41]	-	-	-	130nm	1857	926	926	499
Trivium (1-bit output version) [41]	-	-	-	130nm	2599	358	358	138
Mickey128 [41]	-	-	-	130nm	5039	413	413	82
WG(29,11) [68]	ONB	Standard	[71]	65nm	33200	144	144	4.34
WG(29,11) [56]	-	Look-up Table (ROM)	-	-	319 Registers + 9000 XORs + 2^{29} ROM bits	-	-	-
WG(29,11) [31]	ONB	Standard	[71]	65nm	19900	224	224	11.26
WG(29,11) (This work, Figure 4.3)	PB	Standard	[72]	65nm	17165	202	202	11.77
WG(29,11) (This work, Figure 4.6)	PB	Serialized	[72]	65nm	7050	610	101	14.32
WG(29,11) (This work, Figure 4.8)	PB	Pipelined	[72]	65nm	21190	917	917	43.28
WG-16 [35]	NB	Pipelined $(^{M_{16}}/_{I_8})$	-	65nm	12031	552	552	45.88
WG-16 [35]	NB	Pipelined $(^{M_8}/_{I_8})$	-	65nm	12352	558	558	45.17
WG-16 (This work, Figure 4.12a)	PB	Standard	[72]	65nm	9103	189	189	20.76
WG-16 (This work, Figure 4.12a)	PB	Standard	[53]	65nm	8060	193	193	23.94
WG-16 (This work, Figure 4.15)	PB	Pipelined	[72]	65nm	11795	1149	1149	97.41
WG-16 (This work, Figure 4.15)	PB	Pipelined	[53]	65nm	10681	1370	1370	128.26
WG-16 (This work, Figure 4.13)	PB	Serialized	[72]	65nm	5267	680	113	21.45
WG-16 (This work, Figure 4.13)	PB	Serialized	[53]	65nm	5026	714	119	23.67

Table 4.8: Results obtained for area and speed from the ASIC implementations.

Chapter 5

Digit-Level Architectures for $GF\left(2^m\right)$ Multiplication in the GNB

This chapter, focuses on field multiplication based on the GNB representation for binary extension fields of odd values of m. This includes the five fields recommended by NIST for Elliptic curve digital signature algorithm (ECDSA) [12]. For clarity of reference, in what follows, the multiplication of two field elements is referred to as single multiplication, while the multiplication of two or more elements is denoted by hybrid multiplication.

This chapter, proposes three new digit-level architectures for the single GNB multiplication, which follow different input/output order schemes. Two new digit-level architectures for the FSIPO single GNB multiplication are proposed, one follows an MSD order of its inputs while the other follows an LSD order. It is worth mentioning that a FSIPO multiplier does not require any preloading of the operands, which is not the case for the other input schemes (see Chapter 2). This makes the FSIPO multipliers advantageous for achieving high throughput in applications where the data path capacity, for inputs preloading, is small and m is large. Also, an area efficient version of the MSD DL-PISO single GNB multiplier, which was originally presented in [70], is proposed.

In addition to above three single multipliers, a new DL-SIPO hybrid-double GNB multiplier, and for the first time in literature, a DL-PIPO hybrid-triple GNB multiplier, are proposed by combining the proposed DL-PISO and DL-FSIPO single multipliers. The proposed digit-level hybrid-triple multiplication scheme accomplishes three field multiplications using the latency required for a single digit-level multiplication, at the expense of more area.

Furthermore, and based on the new hybrid-triple GNB multiplier, a digit-level eight-ary field exponentiation architecture is proposed. Compared to the existing digit-level eight-ary schemes [83, 42], the proposed architecture offers almost the same latency while it does not require any precomputation or storage of the field element's odd powers which are less than 8.

The following, summarizes the contributions of this chapter.

Contributions

The contributions of this chapter are summarized in Figure 5.1. In this chapter, seven new digit-level architectures are proposed for the $GF(2^m)$ single, hybrid-double, and hybrid-triple multiplication, in addition to a new digit-level architecture for the $GF(2^m)$ eight-ary field exponentiation (see Figure 5.1), based on the GNB representation when m is odd. The contributions of this chapter are explained as follows:

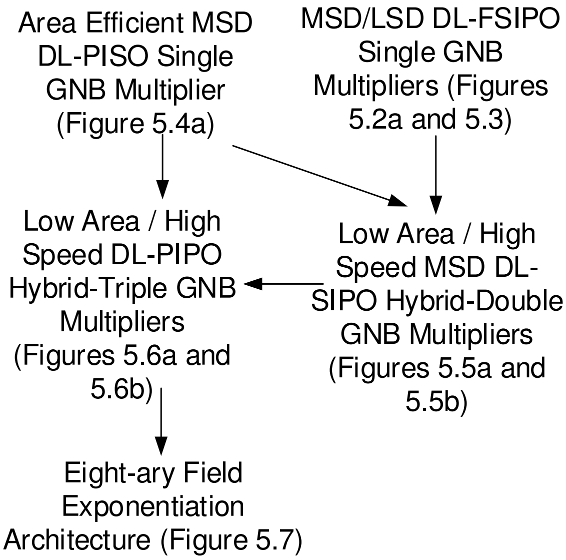

Area Efficient MSD DL-PISO Single GNB Multiplier (Figure 5.4a)
MSD/LSD DL-FSIPO Single GNB Multipliers (Figures 5.2a and 5.3)
Low Area / High Speed DL-PIPO Hybrid-Triple GNB Multipliers (Figures 5.6a and 5.6b)
Low Area / High Speed MSD DL-SIPO Hybrid-Double GNB Multipliers (Figures 5.5a and 5.5b)
Eight-ary Field Exponentiation Architecture (Figure 5.7)

Figure 5.1: Summary of contributions.

- Two new architectures are proposed for MSD/LSD DL-FSIPO single GNB multipliers (Figures 5.2a and 5.3). It is noted that these multipliers do not require preloading of inputs. Therefore, they are advantageous to achieve high throughput in applications where the parallel preloading of the inputs is not possible due to limited sizes of the data path, especially when m is large. For the single bit digit size case, one obtains a bit-level versions of the proposed MSD/LSD DL-FSIPO single GNB multipliers. It is noted that Feng [36] proposed the original most significant bit (MSB), bit-level (BL), FSIPO NB multiplication scheme. However, the MSB version which is obtained from the proposed MSD DL-FSIPO single GNB multiplier is based on a slightly modified formulation compared to the one in [36]. Also, while there are no space and/or time complexities formulations presented in [36], the formulations for the space and time complexities of the proposed MSD/LSD DL-FSIPO single GNB multiplication architectures are derived. For $GF\left(2^5\right)$, the bit-level versions of the proposed FSIPO multipliers (based on the type-2 GNB) require smaller space and time complexities compared to the $GF\left(2^5\right)$ multiplier which is presented in [36]. Moreover, this work proposes reduction of the number of XOR gates through applying sub-expression sharing techniques to the multiplication by β.

- An area efficient MSD DL-PISO single GNB multiplier is proposed (Figure 5.4a), where the number of XOR gates of the original MSD DL-PISO GNB multiplier in [70] is reduced based on applying the sub-expression sharing presented in [17].

- Low area/high speed designs for an MSD DL-SIPO hybrid-double GNB multiplier are proposed (Figure 5.5), constructed by combining the proposed MSD DL-FSIPO and DL-PISO single GNB multipliers (see Figure 5.1). It is noted that the proposed hybrid-double GNB multiplier is the first DL-SIPO scheme proposed for the hybrid-double GNB-based multiplication, while the one presented in [16] follows a DL-PIPO scheme. This in turn allows for proposing a digit-level hybrid-triple multiplier, as it is stated next.

- Low area/high speed designs for a DL-PIPO hybrid-triple GNB multiplier are proposed (Figure 5.6). As shown in Figure 5.1, the proposed DL-PIPO hybrid-triple GNB multipliers are constructed by combining the proposed MSD DL-PISO single and the MSD DL-SIPO hybrid-double GNB multipliers. It is noted that, as far as the author know, the proposed digit-level PIPO hybrid-triple GNB multipliers are the first such multipliers, in the open literature, which perform three digit-level field multiplications using the latency of only one multiplication, at the expense of more area.

- Finally, a digit-level architecture which accomplishes field exponentiation based on radix-8 representation of the exponent is proposed (Figure 5.7). The proposed scheme has almost the same latency which is offered by the exiting digit-level eight-ary exponentiation schemes [83, 42], however, it does not require any precomputations or storage of the field element's odd powers which are less than 8.

The chapter is organized as follows. Section 5.1, presents the proposed MSD/LSD DL-FSIPO single GNB multiplication schemes. Section 5.2 explains the proposed MSD DL-PISO single GNB multiplier. Section 5.3 presents the proposed MSD DL-SIPO hybrid-double and the DL-PIPO hybrid-triple GNB multiplication schemes. Section 5.4 introduces the new digit-level eight-ary field exponentiation architecture. Section 5.5 concludes the chapter.

5.1 Proposed DL-FSIPO Single GNB Multipliers

The following, starts by presenting the proposed MSD DL-FSIPO single GNB multiplier, followed by the LSD one. In addition to their proofs, the proposed digit-level formulations presented in this section, i.e. (5.1), (5.2), (5.3), and (5.4), have been verified through simulations using the Sage tool [3]. It is noted that the proposed multipliers in this section do not require preloading of inputs, and perform the multiplication operation as the input digits enter the multiplier. This is advantageous, especially for large m (> 160 in [12]), to achieve high throughput in applications where the parallel preloading of the inputs is not possible due to limited sizes of the data path.

5.1.1 Proposed MSD DL-FSIPO Single GNB Multiplier

In this section, a digit-level MSD architecture is proposed for the FSIPO single GNB multiplication. In what follows, the formulations for the MSD DL-FSIPO single multiplication in the GNB is first derived, followed by presenting the proposed architecture of the MSD DL-FSIPO single GNB multiplier and, the section ends by analyzing the space and time complexities.

5.1.1.1 Formulations

This section, derives formulations for digit-level multiplication of two $GF(2^m)$ elements represented in the GNB, where the two inputs of the multiplier are entered serially, digit-by-digit, in an MSD first order. In what follows, the proposed MSD first recursive construction of field elements when represented in the GNB is shown.

Lemma 5.1.1 *Given a digit size* $0 < d < m$, *a field element* $A = (a_0, \ldots, a_{m-1}) \in GF(2^m)$ *represented in the GNB, is constructed recursively, starting from the most significant digit* A_{k-1} *(total of* $k = \left\lceil \frac{m}{d} \right\rceil$ *digits* A_0 *through* A_{k-1}*), as follows:*

$$A^{(i)} = A_{k-1-i} + \left(A^{(i-1)}\right)^{2^d} \tag{5.1}$$

where i takes values from 0 *upto* $k-1$, $A^{(-1)} = 0$, $A = A^{(k-1)}$, *and* $A_{k-1-i} = \sum_{j=0}^{d-1} a_{d(k-1-i)+j}\beta^{2^j}$ *is the* $(k-1-i)$*-th digit of* $A = (A_0, \ldots, A_{k-1})$ *with* $a_{d(k-1-i)+j} = 0$ *for* $d(k-1-i) + j \geq m$.

Proof By substituting for $i = 0, \ldots, k-1$ in (5.1), one gets

$$\begin{aligned} A^{(k-1)} =& A_0 + \left(A_1 + \cdots \left(A_{k-2} + (A_{k-1})^{2^d}\right)^{2^d} \cdots\right)^{2^d} \\ =& \sum_{i=k-1}^{0} A_{k-1-i}^{2^{d(k-1-i)}}, \end{aligned}$$

and by noticing that $A_{k-1-i} = \sum_{j=0}^{d-1} a_{d(k-1-i)+j}\beta^{2^j}$ one obtains

$$\begin{aligned}A^{(k-1)} &= \sum_{i=k-1}^{0}\sum_{j=0}^{d-1} a_{j+d(k-1-i)}\beta^{2^{j+d(k-1-i)}} \\ &= \sum_{j=0}^{d-1} a_j\beta^{2^j} + \sum_{j=0}^{d-1} a_{j+d}\beta^{2^{j+d}} + \cdots + \\ &\sum_{j=0}^{d-1} a_{j+d(k-1)}\beta^{2^{j+d(k-1)}} = \sum_{j=0}^{m-1} a_j\beta^{2^j},\end{aligned}$$

where the last result is achieved since $a_{j+d(k-1)} = 0$ for $j + d(k-1) \geq m$.

Then, the multiplication of the $GF(2^m)$ elements A and B is obtained as follows.

Proposition 5.1.2 *Let $E = AB$ be the multiplication of the two elements $A, B \in GF(2^m)$ represented in the GNB. By using construction* (5.1), *one obtains $E = A^{(k-1)}B^{(k-1)}$, where $k = \left\lceil \frac{m}{d} \right\rceil$ and d is the digit size, by the following recurrence starting at $i = 0$ upto $k-1$*

$$A^{(i)}B^{(i)} = \sum_{j=0}^{d-1}\left(\left(a_{d(k-1-i)+j}\left(B_{k-1-i} + \left(B^{(i-1)}\right)^{2^d}\right) + b_{d(k-1-i)+j}\left(A^{(i-1)}\right)^{2^d}\right)^{2^{-j}}\beta\right)^{2^j} + \left(A^{(i-1)}B^{(i-1)}\right)^{2^d}. \tag{5.2}$$

Proof $A^{(i)}B^{(i)}$ is obtained by substituting for $A^{(i)}$ and $B^{(i)}$ in $A^{(i)}B^{(i)}$, using (5.1), as

$$\begin{aligned}A^{(i)}B^{(i)} &= \left(A_{k-1-i} + \left(A^{(i-1)}\right)^{2^d}\right)\left(B_{k-1-i} + \left(B^{(i-1)}\right)^{2^d}\right) \\ &= A_{k-1-i}\left(B_{k-1-i} + \left(B^{(i-1)}\right)^{2^d}\right) + \\ &B_{k-1-i}\left(A^{(i-1)}\right)^{2^d} + \left(A^{(i-1)}B^{(i-1)}\right)^{2^d},\end{aligned}$$

and by substituting for $A_{k-1-i} = \sum_{j=0}^{d-1} a_{d(k-1-i)+j}\beta^{2^j}$ in $A_{k-1-i}\left(B_{k-1-i} + \left(B^{(i-1)}\right)^{2^d}\right)$, and for $B_{k-1-i} = \sum_{j=0}^{d-1} b_{d(k-1-i)+j}\beta^{2^j}$ in $B_{k-1-i}\left(A^{(i-1)}\right)^{2^d}$ the following is obtained

$$A^{(i)}B^{(i)} = \sum_{j=0}^{d-1} a_{d(k-1-i)+j}\beta^{2^j}\left(B_{k-1-i} + \left(B^{(i-1)}\right)^{2^d}\right) + \sum_{j=0}^{d-1} b_{d(k-1-i)+j}\beta^{2^j}\left(A^{(i-1)}\right)^{2^d} + \left(A^{(i-1)}B^{(i-1)}\right)^{2^d}$$

which yields

$$A^{(i)}B^{(i)} = \sum_{j=0}^{d-1}\left(\left(a_{d(k-1-i)+j}\left(B_{k-1-i} + \left(B^{(i-1)}\right)^{2^d}\right) + b_{d(k-1-i)+j}\left(A^{(i-1)}\right)^{2^d}\right)^{2^{-j}}\beta\right)^{2^j} + \left(A^{(i-1)}B^{(i-1)}\right)^{2^d}.$$

It is noted that the correctness of (5.2) has also been verified using simulations with the Sage tool [3].

In (5.2), the multiplication of A by B (elements of $GF\left(2^m\right)$) represented in the GNB, is reduced recursively to a number of bit-wise AND operations, field additions, multiplications with the normal element β, and cyclic shifts for computing the powers 2^{-j}, 2^j, and 2^d. Notice that the addition of the digit B_{k-1-i} to $\left(B^{(i-1)}\right)^{2^d}$ in (5.2) is a free of cost concatenation. This is because the most significant digit of $B^{(i-1)}$ is 0^d for $0 \leq i < k$, where 0^d denotes a string of zeros of length d.

Since it is already given that $A^{(-1)} = B^{(-1)} = 0$, therefore by using (5.2), starting at $i = 0$, and proceeding up to $i = k-1$ (k clock cycles), the final result of the multiplication $E = A^{(k-1)}B^{(k-1)}$ is obtained. At each step, the $(k-1-i)$-th digit of A and B, i.e. A_{k-1-i} and B_{k-1-i}, in addition to $A^{(i-1)}$, $B^{(i-1)}$, and $A^{(i-1)}B^{(i-1)}$, are used for computing $A^{(i)}$ and $B^{(i)}$, and $A^{(i)}B^{(i)}$ according to (5.1) and (5.2), respectively. The following example illustrates the proposed multiplication scheme.

Example 5.1.3 *Table 5.1 shows the steps of multiplying the* $GF\left(2^3\right)$ *elements* $A = B = \beta^{2^2} = (0, 0, 1)$*, which are represented in the type-*2 *GNB* $\left\{\beta, \beta^2, \beta^{2^2}\right\}$ *(that is, Optimal normal basis type-*2*), according to* (5.1) *and* (5.2)*, for the case of* $d = 2$ *(i.e.,* $k = 2$*). Note that* $k = 2$ *and* $a_3 = b_3 = 0$*. Also,* (5.2) *is rewritten as* $A^{(i)}B^{(i)} = Z_{1-i}\beta + \left(W_{1-i}^{2^{-1}}\beta\right)^2 + \left(A^{(i-1)}B^{(i-1)}\right)^{2^2}$*, where* $Z_{1-i} = a_{2-2i}X_{1-i} + b_{2-2i}Y_{1-i}$*,* $W_{1-i} = a_{3-2i}X_{1-i} + b_{3-2i}Y_{1-i}$*,* $X_{1-i} = B_{1-i} + \left(B^{(i-1)}\right)^{2^2}$*, and* $Y_{1-i} = \left(A^{(i-1)}\right)^{2^2}$*, for* $0 \leq i < 2$*.*

i	$A_{1-i} = a_{2-2i}\beta + a_{3-2i}\beta^2$	$B_{1-i} = b_{2-2i}\beta + b_{3-2i}\beta^2$	$A^{(i-1)}$	$B^{(i-1)}$
0	$A_1 = a_2\beta + a_3\beta^2 = \beta$	$B_1 = b_2\beta + b_3\beta^2 = \beta$	$A^{(-1)} = 0$	$B^{(-1)} = 0$
1	$A_0 = a_0\beta + a_1\beta^2 = 0$	$B_0 = b_0\beta + b_1\beta^2 = 0$	$A^{(0)} = A_1 + \left(A^{(-1)}\right)^{2^2} = \beta$	$B^{(0)} = B_1 + \left(B^{(-1)}\right)^{2^2} = \beta$

	$X_{1-i} = B_{1-i} + \left(B^{(i-1)}\right)^{2^2}$	$Y_{1-i} = \left(A^{(i-1)}\right)^{2^2}$	$Z_{1-i} = a_{2-2i}X_{1-i} + b_{2-2i}Y_{1-i}$	$W_{1-i} = a_{3-2i}X_{1-i} + b_{3-2i}Y_{1-i}$
0	$X_1 = B_1 + \left(B^{(-1)}\right)^{2^2} = \beta$	$Y_1 = \left(A^{(-1)}\right)^{2^2} = 0$	$Z_1 = a_2X_1 + b_2Y_1 = \beta$	$W_1 = a_3X_1 + b_3Y_1 = 0$
1	$X_0 = B_0 + \left(B^{(0)}\right)^{2^2} = \beta^{2^2}$	$Y_0 = \left(A^{(0)}\right)^{2^2} = \beta^{2^2}$	$Z_0 = a_0X_0 + b_0Y_0 = 0$	$W_0 = a_1X_0 + b_1Y_0 = 0$

	$A^{(i-1)}B^{(i-1)}$	$A^{(i)}B^{(i)} = Z_{1-i}\beta + \left(W_{1-i}^{2^{-1}}\beta\right)^2 + \left(A^{(i-1)}B^{(i-1)}\right)^{2^2}$
0	$A^{(-1)}B^{(-1)} = 0$	$A^{(0)}B^{(0)} = Z_1\beta + \left(W_1^{2^{-1}}\beta\right)^2 + \left(A^{(-1)}B^{(-1)}\right)^{2^2} = \beta^2$
1	$A^{(0)}B^{(0)} = \beta^2$	$A^{(1)}B^{(1)} = Z_0\beta + \left(W_0^{2^{-1}}\beta\right)^2 + \left(A^{(0)}B^{(0)}\right)^{2^2} = \beta^{2^3} = \beta$

Table 5.1: Steps for multiplication of the two $GF\left(2^3\right)$ elements $A = B = \beta^{2^2} = (0, 0, 1)$.

Next, the proposed architecture of the MSD DL-FSIPO single GNB multiplier is presented.

5.1.1.2 Architecture

Figure 5.2a presents the architecture of the proposed MSD DL-FSIPO single GNB multiplier.

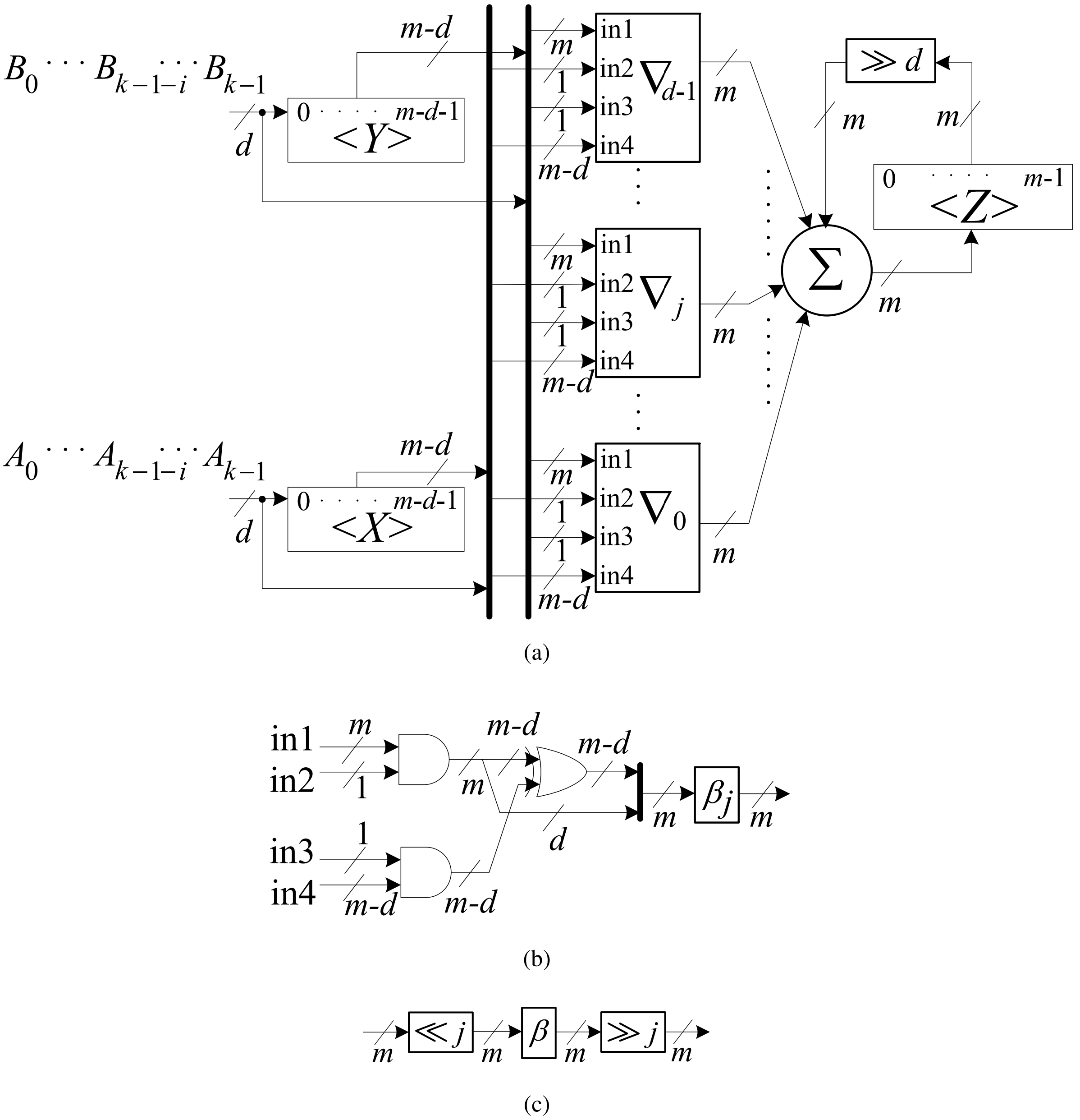


Figure 5.2: (a) Architecture of the proposed MSD DL-FSIPO single GNB multiplier. (b) Architecture of ∇_j. (c) Architecture of β_j.

This architecture is constructed based on (5.1) and (5.2). In Figure 5.2a, d denotes the digit size, $k = \left\lceil \frac{m}{d} \right\rceil$ denotes the total number of cycles of computations, and $0 \leq i < k$ refers to the i-th clock cycle. Part (b) of the same figure presents the architecture of ∇_j which is used in Figure 5.2a, $0 \leq j < d$, where for $0 \leq i < k$: in1 $= B_{k-1-i} + \left(B^{(i-1)}\right)^{2^d}$, in2 $= a_{d(k-1-i)+j}$, in3 $= b_{d(k-1-i)+j}$, and in4 $= \left(B^{(i-1)}\right)^{2^d}$. Also, part (c) shows the architecture of β_j which is used in Figure 5.2b, $0 \leq j < d$.

Initially, the $(m-d)$-bits shift registers $\langle X\rangle$ and $\langle Y\rangle$, and the m-bits register $\langle Z\rangle$, are cleared (i.e., initialized by $A^{(-1)}$, $B^{(-1)}$, and $A^{(-1)}B^{(-1)}$, respectively). Then, at each i-th iteration of the following k iterations, $\langle X\rangle$, $\langle Y\rangle$, and $\langle Z\rangle$ update their states from $A^{(i-1)}$, $B^{(i-1)}$, and $A^{(i-1)}B^{(i-1)}$ to $A^{(i)}$, $B^{(i)}$, and $A^{(i)}B^{(i)}$, respectively, as follows. The two $GF(2^m)$ input elements A and B are entered to registers $\langle X\rangle$ and $\langle Y\rangle$, respectively, one digit per a clock cycle, following a most significant digit first order starting with the $(k-1)$-th digits (according to (5.1)). At the i-th iteration, $\langle X\rangle$ and $\langle Y\rangle$ perform a d-fold right shift (not cyclic) and, the $(k-1-i)$-th digits of A and B are written to the least significant d-bits of $\langle X\rangle$ and $\langle Y\rangle$, respectively. At the same time, register $\langle Z\rangle$ accumulates the result of the field addition $\sum_{j=0}^{d-1} \nabla_j + \left(A^{(i-1)}B^{(i-1)}\right)^{2^d}$, where

$$\nabla_j = \left(\left(a_{d(k-1-i)+j}\left(B_{k-1-i} + \left(B^{(i-1)}\right)^{2^d}\right) + b_{d(k-1-i)+j}\left(A^{(i-1)}\right)^{2^d}\right)^{2^{-j}} \beta\right)^{2^j}$$

is generated as shown in Figures 5.2b and 5.2c. According to (5.2), this results in writing $A^{(i)}B^{(i)}$ to $\langle Z\rangle$. Then, after k clock cycles, i.e. $i = k-1$, one obtains $\langle Z\rangle = A^{(k-1)}B^{(k-1)} = AB$. It is noted that the proposed architecture implements $B_{k-1-i} + \left(B^{(i-1)}\right)^{2^d}$ in (5.2) by concatenating the d-bits of B_{k-1-i} to the least significant digit of $\left(B^{(i-1)}\right)^{2^d}$ (the concatenations are shown by thick vertical lines in Figure 5.2, two in Figure 5.2a and one in Figure 5.2b). This concatenation is possible since the least significant digit of $\left(B^{(i-1)}\right)^{2^d}$ is simply 0^d for all $0 \leq i < k$ (notice from (5.2) that $A^{(k-1)}$ and $B^{(k-1)}$ are not used in generating $A^{(k-1)}B^{(k-1)}$).

In the following, the space and time complexities of the proposed MSD DL-FSIPO single GNB multiplier are studied.

5.1.1.3 Space and Time Complexities

The space complexity of the proposed MSD DL-FSIPO single GNB multiplier is listed in Table 5.2. This includes the count of logic gates, Flip Flop (FF), and preloading multiplexers (for parallel inputs preloading). In this table, d is the digit size. T is the GNB type. P and S denote either the corresponding input/output is loaded/generated in parallel or in serial, respectively. It is noted that this table shows the space complexity of the proposed MSD DL-FSIPO single GNB multiplier before applying sub-expression sharing. From Figure 5.2b, one can see that each ∇_j module, $0 \leq j < d$, consists of $m+m-d = 2m-d$ two input AND gates, and therefore, the total number of two input AND gates in the d ∇_j modules of Figure 5.2a is $d(2m-d)$. The total number of two input XOR gates in the $\sum$ module of Figure 5.2a (a $GF(2^m)$ adder which

Multiplier	FF	AND	XOR	2 : 1 1-bit MUX	Input 1	Input 2	Output
DL-PISO [61]	$2m$	$d\,[T\,(m-1)+1]$	$d\,[T\,(m-1)]$	$2m$	P	P	S
DL-PISO [37]	$2m$	dm	$d\,[T\,(m-1)]$	$2m$	P	P	S
DL-PISO[1] [16]	$2m$	dm	$\leq d\left[(T-1)\left((m-1)-\frac{d-1}{2}\right)\right]+d\,(m-1)$	$2m$	P	P	S
DL-PIPO [17]	$3m$	dm	$d\left[\frac{(m-1)(T-1)}{2}+m\right]$	$2m$	P	P	P
DL-SIPO[1] [16]	$2m$	dm	$\leq d\,(T-1)\left[(m-1)-\frac{d-1}{2}\right]+dm$	m	S	P	P
M/LSD DL-FSIPO[2] (Figures 5.2 & 5.3)	$3m-2d$	$d\,(2m-d)$	$\leq d\,[(2m-d)+(T-1)\,(m-1)]$	0	S	S	P
MSD DL-PISO[1] (Figure 5.4a)	$2m$	dm	$\leq d\left[(T-1)\left((m-1)-\frac{d-1}{2}\right)\right]+d\,(m-1)$	$2m$	P	P	S

[1] without applying group sub-expression elimination. [2] without applying sub-expression elimination.

Table 5.2: Space complexity of digit-level single GNB multipliers.

adds $d+1$ field elements) is dm. In addition, each ∇_j module, $0 \leq j < d$, has $\leq (m-d) + (T-1)\,(m-1)$ XORs out of which are $\leq (T-1)\,(m-1)$ contributed by the multiplications by β (before sub-expression elimination, see Section 2.9.2.2). Therefore, the total number of XORs in the MSD DL-FSIPO single GNB multiplier is $\leq d\,[(2m-d)+(T-1)\,(m-1)]$. In addition, while register $\langle Z \rangle$ has m FFs, only $m-d$ FFs are required for each one of registers $\langle X \rangle$ and $\langle Y \rangle$, since the $(k-1)$-th digit in these two registers is always zero throughout the computations. Hence, the total number of FFs is $2\,(m-d)+m = 3m-2d$. One can also see that there are no preloading multiplexers required for the proposed MSD DL-FSIPO single GNB multiplier.

On the other hand, Table 5.3 reports the time complexity of the proposed digit-level MSD FSIPO single GNB multiplier, in terms of the propagation delay of the corresponding levels of two input XOR and AND gates through the critical path. As seen from Figure 5.2a, the critical path of the proposed architecture passes through one ∇_j module and the $\sum$ module. The

Multiplier	Propagation Delay	Serial Preloading Latency	Computation Latency
DL-PISO [61]	$T_A + (\lceil \log_2 (T\,(m-1)+1) \rceil)\, T_X$	k	k
DL-PISO [37]	$T_A + (\lceil \log_2 m \rceil + \lceil \log_2 T \rceil)\, T_X$	k	k
DL-PISO [16]	$T_A + (\lceil \log_2 m \rceil + \lceil \log_2 T \rceil)\, T_X$	k	k
DL-PIPO [17]	$T_A + (\lceil \log_2 (d+1) \rceil + \lceil \log_2 T \rceil)\, T_X$	k	k
DL-SIPO [16]	$T_A + (\lceil \log_2 (d+1) \rceil + \lceil \log_2 T \rceil)\, T_X$	k	k
M/LSD DL-FSIPO (Figures 5.2 & 5.3)	$T_A + [1 + \lceil \log_2 (d+1) \rceil + \lceil \log_2 T \rceil]\, T_X$	0	k
MSD DL-PISO (Figure 5.4a)	$T_A + (\lceil \log_2 m \rceil + \lceil \log_2 T \rceil)\, T_X$	k	k

Table 5.3: Time complexity of digit-level single GNB multipliers.

propagation delay of a ∇_j module, $0 \leq j < d$, is $T_A + (1 + \lceil \log_2 (T) \rceil) T_X$, where $\lceil \log_2 (T) \rceil T_X$ is the propagation delay through β_j (due to the multiplication with β, see Section 2.9.2.2). Therefore, by adding the delay of the $\sum$ module (a $GF(2^m)$ adder which adds $d + 1$ field elements), which is $\lceil \log_2 (d + 1) \rceil T_X$, the total propagation delay of the proposed multiplier becomes $T_A + [1 + \lceil \log_2 (d + 1) \rceil + \lceil \log_2 T \rceil] T_X$.

5.1.1.4 Bit-Level Case

It is noted that the original bit-level FSIPO multiplication scheme was presented by Feng [36] for an MSB order of the inputs. By considering a single-bit digit size, one obtains a bit-level MSB FSIPO single GNB multiplier from the proposed digit-level architecture. The obtained proposed MSB BL-FSIPO single GNB multiplier offers a maximum propagation delay of $T_A + 3T_X$, while it requires 13 FFs, 9 ANDs, and 13 XORs (for the case of $GF\left(2^5\right)$ and $T = 2$). On the other hand, the $GF\left(2^5\right)$ multiplier presented in [36] has a maximum propagation delay of $T_A + 6T_X$ and requires 13 FFs, 9 ANDs, and 15 XORs. Moreover, in this work, the formulations for the space and time complexities of the proposed MSD DL-FSIPO single GNB multiplier are derived, while there are no such formulations presented in [36]. In addition, this work further reduces the space complexity of the proposed architecture through applying sub-expression sharing techniques to the multiplication by β (see [69, 25] for example).

Table 5.4 estimates the corresponding space and time complexity readings for the case of bit-level ($d = 1$) versions of the different multipliers in Tables 5.2 and 5.3, considering the type-4 GNB of $GF\left(2^{163}\right)$, based on the 65nm CMOS standard library's statistics. It is noted that, the NAND gate equivalence (GE) is obtained for a two input AND, two input XOR, D-type FF, and a 2 : 1 1-bit MUX through synthesis using the Synopsys Design Vision tool [4] to be 1.25, 2, 3.75, and 2, respectively. Similarly, the maximum propagation delay (MPD) is obtained for a two input AND and two input XOR to be 0.03ns and 0.04ns, respectively. In this table, latency denotes the number of clock cycles required for computing the m-bits of output. TP is

Multiplier	MPD *ns*	Serial Input Loading			Parallel Input Loading		
		Total Gates	Latency	TP/G @ 1 GHz	Total Gates[2]	Latency	TP/G @ 1 GHz
BL-PISO [61]	0.43	3329.75	326	150	3981.75	164	250
BL-PISO [37]	0.43	2722.25	326	184	3374.25	164	295
BL-PIPO [17]	0.15	2849.5	326	175	3501.5	164	284
LSB BL-SIPO [16]	0.15	2724.25	326	184	3050.25	164	326
M/LSB BL-FSIPO[1] (Figures 5.2 & 5.3, $d = 1$)	0.19	3854.5	163	259	3854.5	163	259

[1] without elimination. If we apply the elimination in [69], savings is 127 XORs = 254 GE.

[2] with MUXs.

Table 5.4: Space and time complexity readings for the case of type-4 GNB of $GF\left(2^{163}\right)$ digit-level single multipliers.

throughput and TP/G denotes throughput (@ 1 GHz) per total GE measured in Kbps/Gate. As one can see from this table, the bit-level version of the proposed DL-FSIPO single GNB multiplier offers half the latency and provides the best normalized throughput compared to the other multipliers in the case of serial loading of inputs. Moreover, one can further reduce the space complexity of the proposed architecture through applying sub-expression sharing techniques to the multiplication by β. For example, applying the elimination algorithm proposed in [69], saves 127 XORs which is equivalent to 254 GE.

In the following section, new LSD DL-FSIPO single GNB multiplier is introduced.

5.1.2 Proposed LSD DL-FSIPO Single GNB Multiplier

This section, presents the LSD DL-FSIPO single GNB multiplier. The section starts by deriving the formulations for the LSD DL-FSIPO single multiplication scheme. Then, it presents the architecture of the proposed multiplier. The section concludes by studying space and time complexities.

5.1.2.1 Formulations

Here, the formulations are derived for digit-level multiplication of two $GF\left(2^m\right)$ elements based on the GNB representation, where the two inputs of the multiplier are entered in an LSD first order. First, the following shows how one constructs the elements of $GF\left(2^m\right)$ when represented in the GNB, digit by digit, starting from the least significant digit.

Lemma 5.1.4 *Given the digit size* $0 < d < m$*, an arbitrary* $GF\left(2^m\right)$ *element* $A = (a_0, \ldots, a_{m-1})$ *represented in the GNB is constructed, recursively, starting with its least significant digit, as follows:*

$$A^{(i)} = \left(A_i + A^{(i-1)}\right)^{2^{-d}} \tag{5.3}$$

where i takes values starting from 0 *upto* $k-1$, $k = \left\lceil \frac{m}{d} \right\rceil$, $A = A^{(k-1)}$, $A^{(-1)} = 0$*, and* $A_i = \sum_{j=0}^{d-1} a_{di+j-r}\beta^{2^j}$ *is the i-th digit of A such that* $a_{di+j-r} = 0$ *for* $di + j - r < 0$ *given* $r = kd - m$ *which represents the number of left padded zeros.*

Proof By substituting for $i = 0, \ldots, k-1$ in (5.3), one gets

$$\begin{aligned} A^{(k-1)} &= \left(A_{k-1} + \cdots \left(A_1 + (A_0)^{2^{-d}}\right)^{2^{-d}} \cdots \right)^{2^{-d}} \\ &= \sum_{i=0}^{k-1} A_i^{2^{-d(k-i)}}, \end{aligned}$$

and by noticing that $A_i = \sum_{j=0}^{d-1} a_{di+j-r}\beta^{2^j}$ the following is obtained

$$\begin{aligned} A^{(k-1)} &= \sum_{i=0}^{k-1}\sum_{j=0}^{d-1} a_{di+j-r}\beta^{2^{j-d(k-i)}} \\ &= \left(\sum_{i=0}^{k-1}\sum_{j=0}^{d-1} a_{di+j-r}\beta^{2^{j+di}}\right)^{2^{-dk}} . \end{aligned}$$

Now, let $l = di + j$, and notice that $dk = m + r$ (since $r = kd - m$), then

$$\begin{aligned} A^{(k-1)} &= \left(\sum_{l=0}^{m+r-1} a_{l-r}\beta^{2^l}\right)^{2^{-m-r}} \\ &= \left(\sum_{l=-r}^{m-1} a_l\beta^{2^{l+r}}\right)^{2^{-r}} = \sum_{l=0}^{m-1} a_l\beta^{2^l}, \end{aligned}$$

where the last result is achieved since $a_l = 0$ for $l < 0$.

Then, the multiplication of two $GF(2^m)$ elements A and B represented in the GNB and constructed by (5.3), is obtained as follows.

Proposition 5.1.5 *Let $E = AB$ be the multiplication of two elements $A, B \in GF(2^m)$ represented in the GNB. Therefore, using construction (5.3), $E = A^{(k-1)}B^{(k-1)}$ is obtained by the following recurrence*

$$A^{(i)}B^{(i)} = \left[\sum_{j=0}^{d-1}\left(\left(a_{di+j-r}\left(B_i + B^{(i-1)}\right) + b_{di+j-r}A^{(i-1)}\right)^{2^{-j}}\beta\right)^{2^j} + A^{(i-1)}B^{(i-1)}\right]^{2^{-d}}, \qquad (5.4)$$

where i takes values starting from 0 upto $k-1$, $k = \left\lceil \frac{m}{d} \right\rceil$, and $A^{(-1)} = B^{(-1)} = A^{(-1)}B^{(-1)} = 0$.

Proof $A^{(i)}B^{(i)}$ is obtained by substituting for $A^{(i)}$ and $B^{(i)}$ in $A^{(i)}B^{(i)}$, using (5.3), as

$$\begin{aligned} A^{(i)}B^{(i)} &= \left[\left(A_i + A^{(i-1)}\right)\left(B_i + B^{(i-1)}\right)\right]^{2^{-d}} \\ &= \left[A_i\left(B_i + B^{(i-1)}\right) + B_iA^{(i-1)} + A^{(i-1)}B^{(i-1)}\right]^{2^{-d}}, \end{aligned}$$

and by substituting for $A_i = \sum_{j=0}^{d-1} a_{di+j-r}\beta^{2^j}$ in $A_i\left(B_i + B^{(i-1)}\right)$, and for $B_i = \sum_{j=0}^{d-1} b_{di+j-r}\beta^{2^j}$ in $B_iA^{(i-1)}$ one gets

$$\begin{aligned} A^{(i)}B^{(i)} = \Bigg[&\sum_{j=0}^{d-1} a_{di+j-r}\beta^{2^j}\left(B_i + B^{(i-1)}\right) + \\ &\sum_{j=0}^{d-1} b_{di+j-r}\beta^{2^j}A^{(i-1)} + A^{(i-1)}B^{(i-1)}\Bigg]^{2^{-d}} \end{aligned}$$

which yields

$$\begin{aligned} A^{(i)}B^{(i)} = & \Bigg[\sum_{j=0}^{d-1} \Bigg(\Bigg(a_{di+j-r} \left(B_i + B^{(i-1)} \right) + \\ & b_{di+j-r} A^{(i-1)} \Bigg)^{2^{-j}} \beta \Bigg)^{2^j} + A^{(i-1)}B^{(i-1)} \Bigg]^{2^{-d}}. \end{aligned}$$

In (5.4), and similar to (5.2), the multiplication of A by B (elements of $GF(2^m)$) represented in the GNB, is reduced recursively to a number of bit-wise AND operations, field additions, multiplications with the normal element β, and cyclic shifts for computing the powers 2^{-j}, 2^j, and 2^{-d}. It is noted that the field addition of the term B_i in (5.4) is realized through concatenation. The concatenation is possible since the least significant digit of $B^{(i-1)}$ is always 0 for $0 \leq i < k$ (only $B^{(k-1)}$ has a non zero LSD; however, $B^{(k-1)}$ is not used in computing $A^{(k-1)}B^{(k-1)}$).

Since it is already given that $A^{(-1)} = B^{(-1)} = 0$, therefore, starting at $i = 0$, and proceeding up to $i = k - 1$, one obtains the final result of the multiplication $AB = A^{(k-1)}B^{(k-1)}$. As one can see from (5.4), at each step, the i-th digits in A and B, i.e. A_i and B_i, together with $A^{(i-1)}$, $B^{(i-1)}$, and $A^{(i-1)}B^{(i-1)}$, are used for computing $A^{(i)}B^{(i)}$. The following example illustrates the proposed multiplication scheme for the case of $GF\left(2^3\right)$ where $d = 2$ and the elements are represented in the GNB type-2.

Example 5.1.6 *Table 5.5 shows how one multiplies the* $GF\left(2^3\right)$ *elements* $A = B = \beta^{2^2} = (0, 0, 1)$*, which are represented in the type-*2 *GNB* $\left\{\beta, \beta^2, \beta^{2^2}\right\}$ *(that is, Optimal normal basis type-*2*), using* (5.3) *and* (5.4) *for the case of* $d = 2$*. Note that, in this case* $k = 2$*, and* $r = 1$*. For*

i	$A_i = a_{2i-1}\beta + a_{2i}\beta^2$	$B_i = b_{2i-1}\beta + b_{2i}\beta^2$	$A^{(i-1)}$	$B^{(i-1)}$
0	$A_0 = a_{-1}\beta + a_0\beta^2 = 0$	$B_0 = b_{-1}\beta + b_0\beta^2 = 0$	$A^{(-1)} = 0$	$B^{(-1)} = 0$
1	$A_1 = a_1\beta + a_2\beta^2 = \beta^2$	$B_1 = b_1\beta + b_2\beta^2 = \beta^2$	$A^{(0)} = \left(A_0 + A^{(-1)}\right)^{2^{-2}} = 0$	$B^{(0)} = \left(B_0 + B^{(-1)}\right)^{2^{-2}} = 0$

i	$X_i = B_i + B^{(i-1)}$	$Y_i = A^{(i-1)}$	$Z_i = a_{2i-1}X_i + b_{2i-1}Y_i$	$W_i = a_{2i}X_i + b_{2i}Y_i$
0	$X_0 = B_0 + B^{(-1)} = 0$	$Y_0 = A^{(-1)} = 0$	$Z_0 = a_{-1}X_0 + b_{-1}Y_0 = 0$	$W_0 = a_0X_0 + b_0Y_0 = 0$
1	$X_1 = B_1 + B^{(0)} = \beta^2$	$Y_1 = A^{(0)} = 0$	$Z_1 = a_1X_1 + b_1Y_1 = 0$	$W_1 = a_2X_1 + b_2Y_1 = \beta^2$

i	$A^{(i-1)}B^{(i-1)}$	$A^{(i)}B^{(i)} = \left(Z_i\beta + \left(W_i^{2^{-1}}\beta\right)^2 + A^{(i-1)}B^{(i-1)}\right)^{2^{-2}}$
0	$A^{(-1)}B^{(-1)} = 0$	$A^{(0)}B^{(0)} = \left(Z_0\beta + \left(W_0^{2^{-1}}\beta\right)^2 + A^{(-1)}B^{(-1)}\right)^{2^{-2}} = 0$
1	$A^{(0)}B^{(0)} = 0$	$A^{(1)}B^{(1)} = \left(Z_1\beta + \left(W_1^{2^{-1}}\beta\right)^2 + A^{(0)}B^{(0)}\right)^{2^{-2}} = \left(\left(\beta^2\right)^2\right)^{2^{-2}} = \left(\beta^{2^2}\right)^{2^{-2}} = \beta$

Table 5.5: Steps for multiplication of the two $GF\left(2^3\right)$ elements $A = B = \beta^{2^2} = (0, 0, 1)$.

this example, rewrite $A^{(i)}B^{(i)} = \left(Z_i\beta + \left(W_i^{2^{-1}}\beta\right)^2 + A^{(i-1)}B^{(i-1)}\right)^{2^{-2}}$, *where* $Z_i = a_{2i-1}X_i + b_{2i-1}Y_i$, $W_i = a_{2i}X_i + b_{2i}Y_i$, $X_i = B_i + B^{(i-1)}$, *and* $Y_i = A^{(i-1)}$, *for* $0 \leq i < 2$.

Next, the proposed architecture of the LSD DL-FSIPO single GNB multiplier is presented.

5.1.2.2 Architecture

Here, the architecture of the proposed LSD DL-FSIPO single GNB multiplier is introduced. This architecture is shown in Figure 5.3, which is constructed based on (5.3) and (5.4). In this figure, the digit size is d, $0 < d < m$, and i denotes the i-th clock cycle of the computations, $0 \leq i < k$ where $k = \left\lceil \frac{m}{d} \right\rceil$. Architectures of ∇_j and β_j (which is a component of ∇_j) blocks, $0 \leq j < d$, are shown in Figures 5.2b and 5.2c, respectively, where in Figure 5.2b, and at iteration $0 \leq i < k$, one has: in1 $= B_i + B^{(i-1)}$, in2 $= a_{di+j-r}$, in3 $= b_{di+j-r}$, and in4 $= A^{(i-1)}$. First, the $(m-d)$-bits shift registers $\langle X \rangle$ and $\langle Y \rangle$, and the m-bits register $\langle Z \rangle$ are cleared (in other words, $\langle X \rangle$, $\langle Y \rangle$, and $\langle Z \rangle$ are loaded with $A^{(-1)}$, $B^{(-1)}$, and $A^{(-1)}B^{(-1)}$, respectively). After this, at the i-th iteration, $0 \leq i < k$, registers $\langle X \rangle$, $\langle Y \rangle$, and $\langle Z \rangle$ change states from $A^{(i-1)}$, $B^{(i-1)}$, and $A^{(i-1)}B^{(i-1)}$ to $A^{(i)}$, $B^{(i)}$, and $A^{(i)}B^{(i)}$, respectively, as follows. At iteration i, $\langle X \rangle$ and $\langle Y \rangle$ perform a d-fold left shift (not cyclic) and, at the same time, the i-th digits of the field elements

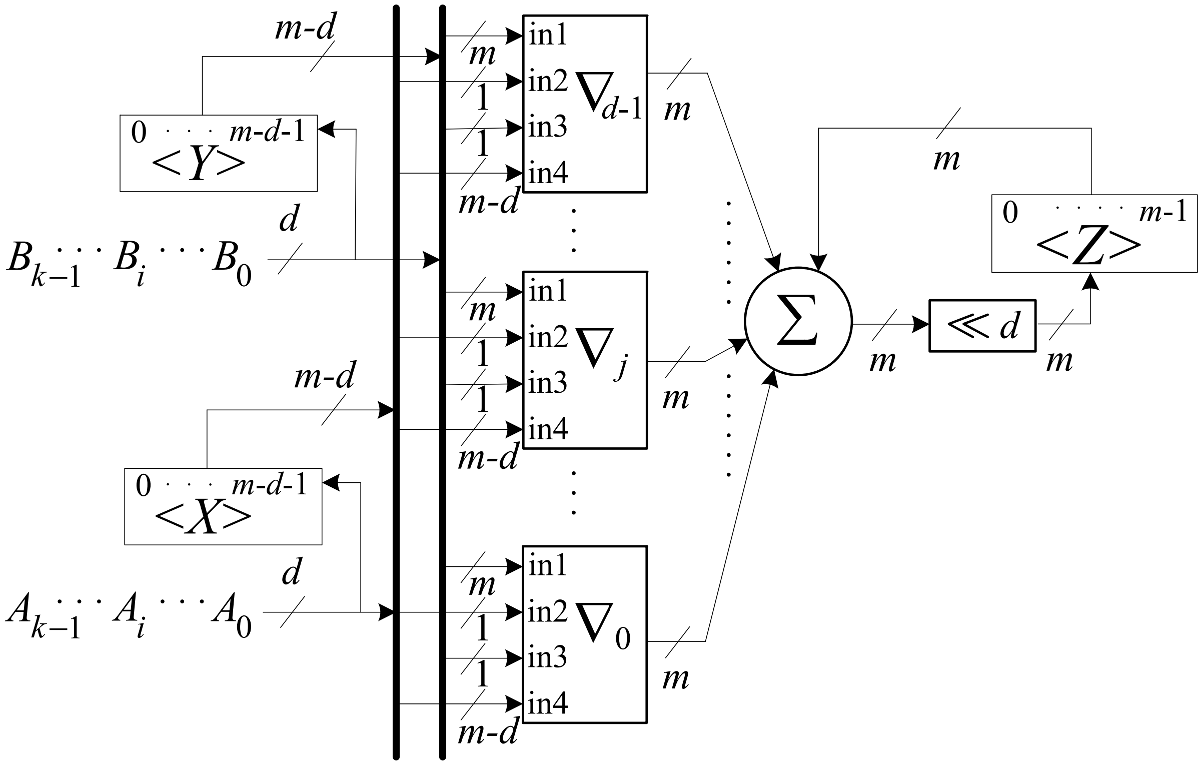


Figure 5.3: Architecture of the proposed LSD DL-FSIPO single GNB multiplier.

A and B are written to the most significant d-bits of $\langle X \rangle$ and $\langle Y \rangle$, respectively, according to (5.3). Moreover, and at the same time, register $\langle Z \rangle$ accumulates the d-fold left cyclic shift of $\sum_{j=0}^{d-1} \nabla_j + A^{(i-1)}B^{(i-1)}$, where the architecture of ∇_j is captured in Figure 5.2b and implements

$$\left(\left(a_{di+j-r}\left(B_i + B^{(i-1)}\right) + b_{di+j-r}A^{(i-1)}\right)^{2^{-j}} \beta\right)^{2^j}$$

for in1 $= B_i + B^{(i-1)}$, in2 $= a_{di+j-r}$, in3 $= b_{di+j-r}$, and in4 $= A^{(i-1)}$. Therefore, after the i-th clock cycle, $\langle Z \rangle = A^{(i)}B^{(i)}$, as one can see from (5.4). After k clock cycles one gets $\langle Z \rangle = A^{(k-1)}B^{(k-1)} = AB$. It is noted that, since the least significant digit of $B^{(i-1)}$ is always 0^d for $0 \leq i < k$, where 0^d denotes a string of zeros of length d, the proposed architecture implements $B_i + B^{(i-1)}$ in (5.4) by concatenating B_i to the least significant digit of $B^{(i-1)}$.

In the following, the space and time complexities of the LSD DL-FSIPO single GNB multiplier are studied.

5.1.2.3 Space and Time Complexities

The space complexity of the proposed LSD DL-FSIPO single GNB multiplier is listed in Table 5.2, in terms of the count of logic gates, FFs, and preloading multiplexers (for the case of parallel inputs preloading). In Section 5.1.1.3, it was found that each ∇_j module, $0 \leq j < d$, has $2m - d$ AND gates and $\leq (m - d) + (T - 1)(m - 1)$ XOR gates. Figure 5.3, on the other hand, shows that the number of two input XOR gates in the $\sum$ module (a $GF(2^m)$ adder which adds $d + 1$ field elements) is dm. And hence, the total number of two input AND gates is $d(2m - d)$, while the total number of XOR gates adds up to $\leq d[(2m - d) + (T - 1)(m - 1)]$. In addition, one can notice that, while register $\langle Z \rangle$ has m FFs, registers $\langle X \rangle$ and $\langle Y \rangle$ have $m - d$ FFs each, since their least significant digits will always be zero throughout the k clock cycles of computations. This adds up to a total of $3m - 2d$ FFs. It is also noted that no preloading is required for the proposed LSD DL-FSIPO single GNB multiplier.

The time complexity of the proposed LSD DL-FSIPO single GNB multiplier is also reported in Table 5.3, in terms of levels of the propagation delay of two input XOR and AND gates through its critical path. Similar to the proposed MSD DL-FSIPO single GNB multiplier (see Section 5.1.1.3), Figure 5.3 shows that the propagation delay of the proposed LSD architecture is equivalent to the sum of the propagation delays through one ∇_j module and the $\sum$ module. Hence, the maximum propagation delay in the proposed LSD DL-FSIPO single GNB multiplier is $T_A + [1 + \lceil \log_2 (d + 1) \rceil + \lceil \log_2 T \rceil] T_X$, as shown in Table 5.3.

It is noted that, the proposed multiplication algorithms in Propositions 5.1.2 and 5.1.5 are different than the one presented in [48]. Propositions 5.1.2 and 5.1.5 build a $GF(2^m)$ element recursively digit-by-digit, starting from the MSD and LSD, respectively. This behaviour results in a DL-FSIPO GNB multiplication scheme. The algorithm presented in [48] takes opposite action by recursively shrinking a $GF(2^m)$ element bit-by-bit, starting from the LSB. The latter behavior constructs a BL-PIPO GNB multiplication scheme, but not a DL-FSIPO GNB one. In addition, the authors of [48] present a bit-parallel GNB multiplier extended from their algorithm. Bit-parallel multipliers do not require any input or output registers for their processing and usually target high throughput applications by generating the output in one clock cycle at the expense of a space complexity which is quadratic in m for the scheme in [48]. On the other hand, this chapter focuses on digit-level multiplications for resource constrained applications which requires input / output registers and trade-off space complexity against larger number of clock cycles.

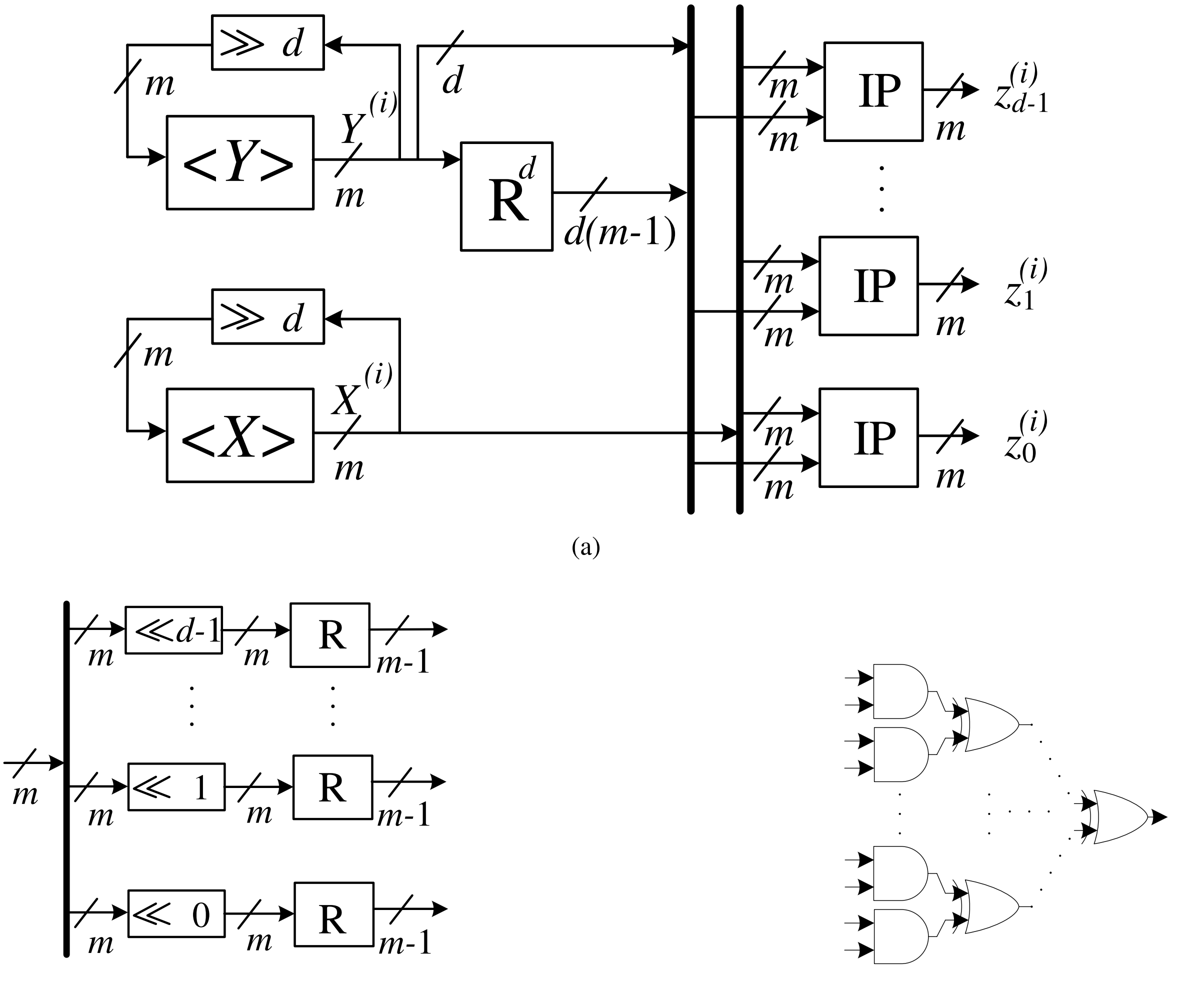
It is worth mentioning that, although the presented DL-FSIPO multiplication algorithms are different from the one in [48], however, they meet at the bit-parallel level. Accordingly, one might construct a multiplexer based DL-FSIPO GNB multipliers through applying partitioning to the bit-parallel architecture presented in [48] (the proposed DL-FSIPO single GNB multipliers are AND / XOR based). In this case, similar efforts to those presented in this chapter need to be taken in order to optimize number of FFs and number of XOR gates within the fixed multiplication by β. Also, notice that, the underlying multiplication algorithm needs to be theoretically aligned / proved according to the proposed formulations. Otherwise, it would be more natural to construct a DL-PIPO GNB multiplier by partitioning of the architecture in [48], which reflects the underlying multiplication algorithm presented in [48]. In fact, the missing of reference to Feng's original work [36] throughout [48] indicates that authors of [48] were determined to use a DL-PIPO algorithm.

5.2 Proposed DL-PISO Single GNB Multiplier

In this section, the proposed architecture of the area efficient MSD DL-PISO single GNB multiplier is presented. This multiplier is an area-optimized instance of the one presented by the authors of [70], which is based on (2.6), and hence, the reader is referred to [70] for more details about the formulations. The area reduction is accomplished through applying the group sub-expression sharing algorithm presented in [17]. In what follows, the architecture of the proposed multiplier is first shown, followed by analyzing its space and time complexities.

5.2.1 Architecture

Here, an area efficient architecture is presented for the MSD DL-PISO single GNB multiplier, as it is shown in Figure 5.4a. Part (b) of this figure depicts the architecture of the R^d block



(a)

(b)

(c)

Figure 5.4: (a) The proposed architecture of the MSD DL-PISO single GNB multiplier.

before applying sub-expression sharing. Part (c) shows the architecture of the IP block. The architecture in this figure differs from the LSD DL-PISO single GNB multiplier, which is presented in [16], in that it generates the multiplication output in the order of most significant digit first. This is accomplished through generating the d output bits $z_{d-1}^{(i)}$ through $z_0^{(i)}$, during iteration i, where $0 \leq i < k$ and $k = \left\lceil \frac{m}{d} \right\rceil$, as follows. In Figure 5.4a, a bit $z_n^{(i)}$ denotes the left most (least significant) coordinate of $P_{X^{(i)}}^{2^{-n}}\left(Y^{(i)}\right)$, where $0 \leq n < d$ and , $X^{(i)} = A^{2^{(i+1)d-t}}$ and $Y^{(i)} = B^{2^{(i+1)d-t}}$ $(t = k \times d - m)$ denote the contents of registers $\langle X \rangle$ and $\langle Y \rangle$ at the i-th iteration of the computations. It is noted that $z_n^{(i)}$, the left most coordinate of $P_{X^{(i)}}^{2^{-n}}\left(Y^{(i)}\right)$, is obtained according to (2.6) as follows

$$z_n^{(i)} = x_n^{(i)} y_{n+1}^{(i)} + \sum_{u=1}^{m-1} x_{((n+u))}^{(i)} \left(\sum_{v=1}^{T} y_{((n+R[u,v]))}^{(i)} \right). \tag{5.5}$$

In (5.5), $x_j^{(i)}$ and $y_j^{(i)}$, respectively, denote the j-th coordinates (cells) of registers $\langle X \rangle$ and $\langle Y \rangle$ during the i-th iteration. In the same formulation, $(())$ denotes the reduction modulo-m. Therefore, by initializing registers $\langle X \rangle$ and $\langle Y \rangle$ such that $X^{(0)} = A^{2^{d-t}}$ and $Y^{(0)} = B^{2^{d-t}}$, one obtains the most significant digit of the output. It is noted that the t-fold left cyclic shift in $A^{2^{d-t}}$ and $B^{2^{d-t}}$ is implemented in order to allow for appending zeros to the t most significant bits of the first output digit (most significant digit), i.e., $\left(z_{d-t}^{(0)}, \ldots, z_{d-1}^{(0)}\right)$. This is required for compatibility of integration with the proposed MSD DL-FSIPO single GNB multiplier (see Section 5.3). Then, at the i-th iteration, $0 \leq i < k$, one has $X^{(i)} = A^{2^{(i+1)d-t}}$ and $Y^{(i)} = B^{2^{(i+1)d-t}}$, due to the d-fold right cyclic shifts which are applied to $\langle X \rangle$ and $\langle Y \rangle$ at each clock cycle, as shown in Figure 5.4a. According to this, bit $z_n^{(i)}$ of the i-th output digit in Figure 5.4a maps to the output bit $e_{m-(d(i+1)-t)+n}$ of the multiplication result $E = AB$. For all $0 \leq n < d$, the inner product in (5.5) is realized through an IP block in Figure 5.4a, while the d instances (for $0 \leq n < d$) of the $m-1$ bits of $\sum_{u=1}^{m-1} \left(\sum_{v=1}^{T} y_{((n+R[u,v]))} \right) \beta^{2^u}$ are generated through the R^d block, which is shown in Figure 5.4b. This figure shows the architecture of R^d before applying the group sub-expression sharing algorithm presented in [17], where each R block represents the matrix multiplication of the lower $m-1$ rows of the multiplication matrix $\mathbf{M}$ by the corresponding m-bits input vertical vector.

The following is the space and time complexity analysis of the proposed MSD DL-PISO single GNB multiplier.

5.2.2 Space and Time Complexities

Here, the space and time complexities of the proposed MSD DL-PISO single GNB multiplier in Figure 5.4a are discussed. In this figure, an IP block consists of m AND gates and $m-1$ XOR gates, as it is shown in Figure 5.4c. Furthermore, for area efficiency, the R^d block of Figure 5.4a realizes a group sub-expression shared version of the d R blocks in Figure 5.4b based on the algorithm presented in [17]. It is noted that one can find the number of eliminations due to this sharing through simulation. Therefore, the architecture of Figure 5.4a requires a total of $2m$ FFs, dm ANDs, and $\leq d\left[(T-1)\left((m-1) - \frac{d-1}{2}\right)\right] + d\,(m-1)$ XORs, as presented in Table 5.2, where $\leq (T-1)\,(m-1)$ is the number of XOR gates in each R block before sub-expression sharing (see Section 2.9.2.2) and the term $\frac{d(d-1)}{2}$ is due to the elimination of common rows between different $\mathbf{R}$ matrices which are used in implementing the d R blocks [16].

For the time complexity, one can see that the critical path of Figure 5.4a has a propagation delay equals to $T_A + (\lceil \log_2 T \rceil + \lceil \log_2 m \rceil)\, T_X$, as presented in Table 5.2, where the delay through the area optimized R^d block is $\lceil \log_2 T \rceil\, T_X$ [16], and that through an IP module is $T_A + \lceil \log_2 m \rceil\, T_X$.

In the following, the proposed MSD DL-FSIPO and DL-PISO single GNB multipliers are combined to construct an MSD DL-SIPO hybrid-double and a DL-PIPO hybrid-triple, GNB multipliers.

5.3 Proposed Digit-Level Hybrid-Double and Hybrid-Triple GNB Multipliers

A hybrid-double digit-level GNB multiplication architecture has been recently proposed by the authors of [16], which performs two field multiplications (multiplication of three field elements) using the same latency required for a single field multiplication (i.e. $k = \left\lceil \frac{m}{d} \right\rceil$ iterations for a digit size d). To accomplish this, the authors of [16] have extended the LSB BL-PISO GNB multiplier in [70] and the LSB BL-SIPO GNB multiplier in [20] to the digit-level, then, by combining these two digit-level single GNB multipliers, they constructed their DL-PIPO hybrid-double GNB multiplier.

In this section, four new architectures for $GF(2^m)$ digit-level hybrid multiplications are presented, two for an MSD DL-SIPO hybrid-double multiplication (a low area and a high speed designs) and, for the first time, another two (low area / high speed designs) for a DL-PIPO hybrid-triple multiplication (multiplication of four field elements), when the field elements are represented in the GNB. In order to construct the proposed hybrid-double multiplier, the MSD DL-PISO single GNB multiplier presented in Section 5.2.1 is combined with the proposed MSD DL-FSIPO single GNB multiplier of Figure 5.2a. On the other hand, the proposed hybrid-triple multiplier is constructed by combining the MSD DL-PISO single GNB multiplier presented in Section 5.2.1 with the MSD DL-SIPO hybrid-double GNB multiplier proposed in this section[1].

In the following, the proposed architectures of the MSD DL-SIPO hybrid-double GNB multiplier are first presented, followed by those of the proposed DL-PIPO hybrid-triple GNB multiplier. This section concludes by analyzing the space and time complexities of the two proposed hybrid multipliers.

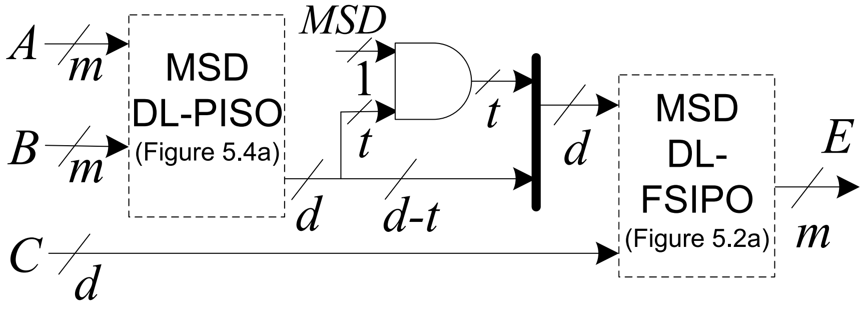
5.3.1 Proposed MSD DL-SIPO Hybrid-Double GNB Multiplier

This section presents the architecture of the proposed MSD DL-SIPO hybrid-double GNB multiplier. It is noted that the hybrid-double GNB multiplier proposed by the authors of [16]

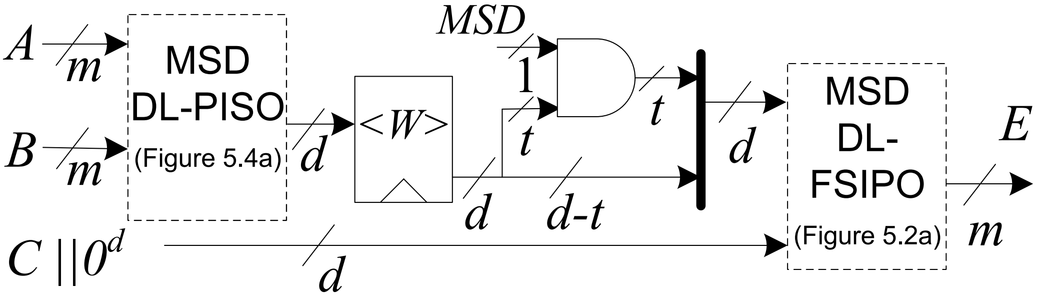
---

[1] It is noted that one can build an LSD DL-SIPO hybrid-double architecture, as well as a DL-PIPO hybrid-triple architecture, by combining the LSD DL-PISO GNB multiplier presented in [16] with the LSD DL-FSIPO GNB multiplier which is proposed in this chapter.

follows a DL-PIPO scheme of its inputs/outputs, while the proposed architecture in this work is the first DL-SIPO scheme for the digit-level hybrid-double GNB multiplication. Figure 5.5 shows two versions of the proposed MSD DL-SIPO hybrid-double GNB multiplier, one for a low area design (Figure 5.5a) and the other for a high speed design (Figure 5.5b). The appended



(a)



(b)

Figure 5.5: Architectures of the proposed MSD DL-SIPO hybrid-double GNB multiplier. (a) Low area design. (b) High speed design.

0^d (zero digit) in input C of part (b) of this figure balances the timing due to pipelining. It is noted that the most significant t-bits - where $t = k \times d - m$, $k = \left\lceil \frac{m}{d} \right\rceil$, and d is the digit size - of the first output digit (most significant digit) of the MSD DL-PISO single GNB multipliers in Figure 5.5 are set to zero through the MSD signal. As can be seen from the figure, the low area MSD DL-SIPO hybrid-double GNB multiplier is built from MSD DL-PISO and DL-FSIPO single GNB multipliers with the output of the former connected to one input of the latter. On the other hand, the high speed version follows from the low area version by inserting the d-bits register $\langle W \rangle$ between the output of the DL-PISO, and the input of the DL-FSIPO, single GNB multipliers, as can be seen from Figure 5.5b. This, in turn, shortens the propagation delay of the multiplier's critical path, which results in reaching higher operating frequencies compared to the low area version. However, it adds one extra clock cycle to the latency. Each one of the two versions of the proposed MSD DL-SIPO hybrid-double GNB multiplier takes three inputs, two of which are m-bits wide each (inputs A and B in Figure 5.5), while the third one has only d-bits (input C in Figure 5.5).

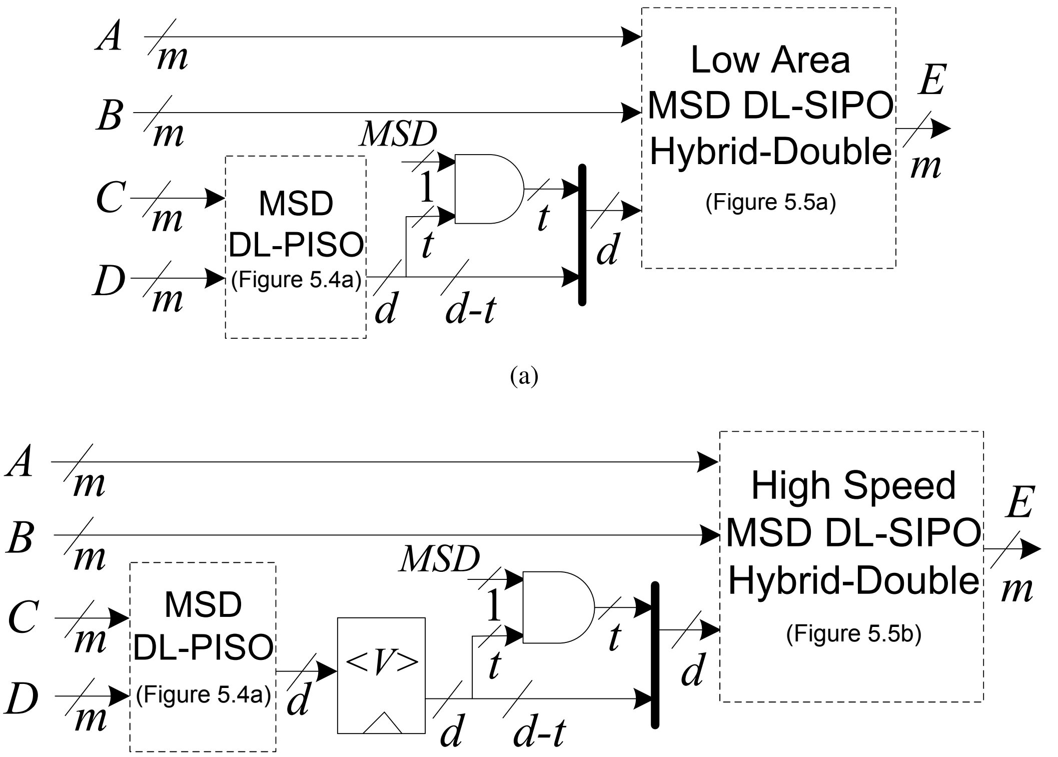
Initially, A and B are loaded to the input registers of the MSD DL-PISO single GNB multiplier, while the input/output registers of the MSD DL-FSIPO single GNB multiplier are cleared out. In the low area version, at the i-th clock cycle, $0 \leq i < k$, the MSD DL-PISO single GNB multiplier generates the $(k - 1 - i)$-th output digit for the multiplication AB, while the MSD DL-FSIPO single GNB multiplier generates $E^{(i)} = (AB)^{(i)} C^{(i)}$ (according to (5.1) and (5.2)). After k iterations, the output register of the MSD DL-FSIPO single GNB multiplier holds the result of the double multiplication, i.e., $E^{(k-1)} = (AB)^{(k-1)} C^{(k-1)}$. In the high speed version, an extra clock cycle is required at the beginning to store the MSD output digit of the DL-PISO single GNB multiplier to register $\langle W \rangle$.

In the following, the proposed architectures for the DL-PIPO hybrid-triple GNB multiplier are introduced.

5.3.2 Proposed DL-PIPO Hybrid-Triple GNB Multiplier

This section, presents the proposed architectures for the DL-PIPO hybrid-triple GNB multiplier. To the best of the author knowledge, this is the first digit-level hybrid GNB multiplier proposed in the literature which performs three $GF(2^m)$ multiplications using the same latency of a single digit-level field multiplication (multiplying of four field elements of A, B, C, and D together). Figures 5.6a and 5.6b present two variants of the proposed DL-PIPO hybrid-triple GNB multiplier. Figure 5.6a is a low area design, while Figure 5.6b shows a high speed design. The most significant t-bits, $t = k \times d - m$, of the first output digit of the MSD DL-PISO single GNB multipliers in these figures are set to zero through the MSD signal. In Figure 5.6a, the low area DL-PIPO hybrid-triple GNB multiplier is constructed from one MSD DL-PISO single, and one low area MSD DL-SIPO hybrid-double, GNB multipliers with the output of the former connected to the serial input of the latter. The high speed DL-PIPO hybrid-triple GNB multiplier instance uses a high speed MSD DL-SIPO hybrid-double GNB multiplier. Also, it has a d-bits register $\langle V \rangle$ inserted between the output of the MSD DL-PISO single GNB multiplier and the input of the high speed MSD DL-SIPO hybrid-double GNB multiplier (see Figure 5.6b). This leads to shorter critical path, and hence, results in reaching higher operating frequencies compared to the low area instance; however, at the expense of one extra clock cycle.

Each one of the two proposed hybrid-triple GNB multiplier's versions takes four m-bits inputs, denoted by A, B, C, and D, and generates an m-bits output, i.e. $E = ABCD$. Initially, A, B, C, and D are loaded to the input registers of the MSD DL-PISO single GNB multipliers



(a)

(b)

Figure 5.6: Architectures of the proposed MSD DL-PIPO hybrid-triple GNB multiplier. (a) Low area design. (b) High speed design.

(including the one in the DL-SIPO hybrid-double GNB multiplier), while the input/output registers of the MSD DL-FSIPO single GNB multiplier (in the hybrid-double multiplier) are cleared out. In the low area version, at the i-th clock cycle, $0 \leq i < k$, the MSD DL-PISO single GNB multipliers generate their $(k-1-i)$-th output digits for AB and CD at the same time, while the output register of the MSD DL-SIPO hybrid-double multiplier computes $E^{(i)} = (AB)^{(i)}\,(CD)^{(i)}$ (according to (5.1) and (5.2)). After k iterations, the output register of the low area DL-PIPO hybrid-double GNB multiplier holds $E^{(k-1)} = (AB)^{(k-1)}\,(CD)^{(k-1)} = ABCD$. On the other hand, the high speed version generates its final output after $k+1$ clock cycles, since an extra clock cycle is required, at the beginning, to store the MSD output digit of CD in register $\langle V \rangle$ and the MSD output digit of AB in register $\langle W \rangle$ (see Figure 5.5b).

Next, the space and time complexities for the different proposed architectures of the digit-level hybrid-double and hybrid-triple GNB multipliers are given.

5.3.3 Space and Time Complexity Analysis

Here, the space and time complexities of the proposed digit-level hybrid-double and hybrid-triple GNB multipliers which are presented in Sections 5.3.1 and 5.3.2, respectively, are derived. Table 5.6 shows the space complexities of the proposed hybrid GNB multipliers, while Table 5.7 presents their corresponding time complexities.

Multiplier	D-FF	AND	XOR[1]	2 : 1 MUX
DL-PIPO Hybrid-Double[2] (low area) [16]	$4m$	$2dm + t$	$\leq d\,(T-1)\,[2\,(m-1)-(d-1)] + d\,(2m-1)$	$3m$
DL-PIPO Hybrid-Double[2] (high speed) [16]	$4m + d$	$2dm + t$	$\leq d\,(T-1)\,[2\,(m-1)-(d-1)] + d\,(2m-1)$	$3m$
MSD DL-SIPO Hybrid-Double (low area) (Figure 5.5a)	$5m - 2d$	$d\,(3m-d) + t$	$\leq d\,(T-1)\left[2\,(m-1)-\frac{d-1}{2}\right] + d\,(3m-(d+1))$	$2m$
MSD DL-SIPO Hybrid-Double (high speed) (Figure 5.5b)	$5m - d$	$d\,(3m-d) + t$	$\leq d\,(T-1)\left[2\,(m-1)-\frac{d-1}{2}\right] + d\,(3m-(d+1))$	$2m$
DL-PIPO Hybrid-Triple (low area) (Figure 5.6a)	$7m - 2d$	$d\,(4m-d) + 2t$	$\leq d\,(T-1)\,[3\,(m-1)-(d-1)] + d\,(4m-(d+2))$	$4m$
DL-PIPO Hybrid-Triple (high speed) (Figure 5.6b)	$7m$	$d\,(4m-d) + 2t$	$\leq d\,(T-1)\,[3\,(m-1)-(d-1)] + d\,(4m-(d+2))$	$4m$

[1] without sub-expression elimination. [2]Note: the authors of [16] did not count for the t ANDs which are required for appending zeros .

Table 5.6: Space complexity of the digit-level hybrid-double and hybrid-triple GNB multipliers.

Multiplier	Propagation Delay	Serial Loading of Inputs Latency	Computation Latency
DL-PIPO Hybrid-Double (low area) [16]	$T_{PISO} + T_{SIPO}$	k	k
DL-PIPO Hybrid-Double (high speed) [16]	$\max\{T_{PISO}, T_{SIPO}\}$	k	$k+1$
MSD DL-SIPO Hybrid-Double (low area) (Figure 5.5a)	$T_{PISO} + T_{FSIPO} + T_A$	k	k
MSD DL-SIPO Hybrid-Double (high speed) (Figure 5.5b)	$\max\{T_{PISO}, T_{FSIPO} + T_A\}$	k	$k+1$
DL-PIPO Hybrid-Triple (low area) (Figure 5.6a)	$T_{PISO} + T_{FSIPO} + T_A$	k	k
DL-PIPO Hybrid-Triple (high speed) (Figure 5.6b)	$\max\{T_{PISO}, T_{FSIPO} + T_A\}$	k	$k+1$

Table 5.7: Time complexity of the digit-level hybrid-double and hybrid-triple GNB multipliers.

In table 5.6, $t = k \times d - m$, $k = \left\lceil \frac{m}{d} \right\rceil$, and d is the digit size. T is the GNB type. In Table 5.7, $T_{PISO} = T_A + (\lceil \log_2 T \rceil + \lceil \log_2 m \rceil)\, T_X$ denotes the delay in the DL-PISO single GNB multiplier, $T_{SIPO} = T_A + (\lceil \log_2 (d+1) \rceil + \lceil \log_2 T \rceil)\, T_X$ denotes the delay in the DL-SIPO single GNB multiplier, and $T_{FSIPO} = T_A + (1 + \lceil \log_2 (d+1) \rceil + \lceil \log_2 T \rceil)\, T_X$ denotes the delay in the DL-FSIPO single GNB multiplier.

Using the construction of Figure 5.5a, one obtains the space complexity of the low area version of the proposed MSD DL-SIPO hybrid-double GNB multiplier which is listed in Table 5.6. This is done by adding the corresponding space complexities of the MSD DL-PISO and DL-FSIPO single GNB multipliers, in addition to the t AND gates which are used for padding the most significant digit with zeros. Similarly, one can find the space complexity of the low area version of the proposed DL-PIPO hybrid-triple GNB multiplier. This is done by adding the corresponding gate count in its MSD DL-PISO single and DL-SIPO hybrid-double GNB multipliers, in addition to the t AND gates in Figure 5.6a, as can be seen from Table 5.6. Moreover, the space complexities of the high speed versions of the proposed hybrid-double and hybrid-triple GNB multipliers are achieved by adding d and $2d$ FFs, respectively, to the space complexities of the corresponding low area versions.

In addition, from Figures 5.5a and 5.6a, one finds that the low area architectures of the proposed MSD DL-SIPO hybrid-double GNB multiplier and the proposed DL-PIPO hybrid-triple GNB multiplier offer maximum propagation delays which are equivalent to $T_{PISO}+T_{FSIPO}+T_A$. In the latter formulation, T_{PISO} and T_{FSIPO} denote the propagation delay through the MSD DL-PISO single GNB multiplier and the MSD DL-FSIPO single GNB multiplier, respectively. On the other hand, due to the insertion of registers (Figures 5.5b and 5.6b), the propagation delays of the high speed architectures of the proposed digit-level hybrid-double and hybrid-triple GNB multipliers are reduced to $\max\{T_{PISO}, T_{FSIPO} + T_A\}$.

In the following, a brief discussion is given about the advantages of using digit-level hybrid GNB multipliers for accomplishing double and triple field multiplications, compared to using digit-level single GNB multipliers with $2d$ and $3d$ digit sizes, respectively.

5.3.4 Hybrid Versus Single Digit-Level GNB Multipliers

This section briefly discusses advantages for using the proposed digit-level hybrid-double and hybrid-triple GNB multipliers, instead of using digit-level single GNB multipliers, for accomplishing double and triple field multiplications, respectively.

It is noted that the proposed digit-level hybrid-double and hybrid-triple GNB multipliers are constructed using two and three digit-level single GNB multipliers, respectively. Hence, for a fair discussion, we compare the digit-level hybrid-double and hybrid-triple GNB multipliers, of digit size d each, to digit-level single GNB multipliers of digit sizes $2d$ and $3d$, respectively.

The main advantage of using the digit-level (digit size d) hybrid-double and hybrid-triple GNB multipliers, in case of double and triple field multiplications, respectively, is that one can obtain lower computational latency, and hence higher throughput, compared to using digit-level single GNB multipliers with digit sizes $2d$ and $3d$, respectively. For example, by using a digit-level hybrid-double GNB multiplier with $d = \left\lceil \frac{m}{3} \right\rceil$, one obtains the result of a double field multiplication after 3 clock cycles. However, a digit-level single GNB multiplier computes the double field multiplication over 4 clock cycles for a digit size $2d = 2\left\lceil \frac{m}{3} \right\rceil$ where $\left\lceil \frac{m}{2} \right\rceil < 2\left\lceil \frac{m}{3} \right\rceil < m$. In addition, by using the digit-level hybrid-double/triple GNB multiplier structures (for digit size d), one can achieve lower space/time complexities for some computational latencies, compared to using digit-level single GNB multipliers (for digit sizes $2d$ and $3d$, respectively).

For example, a digit-level hybrid-triple GNB multiplier with $d = \left\lceil \frac{m}{2} \right\rceil$ accomplishes a triple field multiplication over 2 clock cycles, while the same latency can only be achieved by using two bit-parallel GNB multipliers when multiplying four field elements.

Therefore, the proposed DL-PIPO hybrid-triple GNB multiplier accomplishes three field multiplications using the latency required for a single field multiplication, and hence, it can be used to increase the throughput of applications where such triple multiplications exist. In what follows, a new architecture for the eight-ary field exponentiation is presented, as an application for the digit-level hybrid-triple GNB multiplier presented in this section.

5.4 Proposed Architecture for Field Exponentiation

Exponentiation is a fundamental operation for the Diffie-Hellman key exchange algorithm [29] and is also used for other cryptographic applications such as random number generation [85]. The n-ary scheme is used to increase throughput of $GF\left(2^m\right)$ exponentiation [42, 83]. In the case of $n = 2^3$ (i.e. eight-ary scheme), to compute the exponentiation A^h for $A \in GF\left(2^m\right)$ and a positive integer $h = \sum_{i=0}^{\left\lceil \frac{m}{3} \right\rceil - 1} h_i 2^{3i}$, where $0 \leq h_i < 8$, one rewrites h as $h = \sum_{w=1}^{7} \lambda\left(w\right) w$, where $\lambda\left(w\right) = \sum_{\{i:h_i = w\}} 2^{3i}$. Then, this eight-ary exponentiation scheme requires finding the coefficients $\lambda\left(w\right)$ and precomputing and storing odd powers for $1 < w < 8$. This takes at most $\left\lceil \frac{m}{3} \right\rceil + 2$ iterations to complete [42, 83]. In this section, a new architecture for the eight-ary field exponentiation scheme when the $GF\left(2^m\right)$ elements are represented in the GNB is presented. The proposed architecture is based on the digit-level hybrid-triple GNB multiplier presented in Section 5.3.2 (Figure 5.6) and computes the exponentiation results using $\left\lceil \frac{m}{3} \right\rceil$ iterations, while it does not require any storage of precomputed values.

In the following, the proposed formulations for field exponentiation is first derived, followed by presenting its corresponding proposed architecture.

Proposition 5.4.1 *Let $F = A^h$ denotes the exponentiation of an arbitrary $GF\left(2^m\right)$ element A represented in the GNB, where $1 < h < 2^m$ is an arbitrary positive integer. Therefore, one can compute F using the following recurrence:*

$$F^{(i)} = A^{h_{k'-1-i}} \left(F^{(i-1)}\right)^{2^3}, \tag{5.6}$$

where $k' = \left\lceil \frac{m}{3} \right\rceil$, $h = \sum_{i=0}^{k'-1} h_{k'-1-i} 8^{k'-1-i}$, $0 \leq h_{k'-1-i} < 8$, $F^{(-1)} = 1$, and $F = F^{(k'-1)}$.

Proof By substituting for $i = 0, \ldots, k' - 1$ in (5.6), where $k' = \left\lceil \frac{m}{3} \right\rceil$ and $h_{k'-1-i} \in [0, 7]$ are the coefficients of the radix-8 representation of h, one gets

$$\begin{aligned} F^{(k'-1)} &= A^{(((h_{k'-1}8 + h_{k'-2})8 + h_{k'-3})8 + \cdots + h_1)8 + h_0} \\ &= A^{\sum_{i=0}^{k'-1} h_{k'-1-i} 8^{k'-1-i}}. \end{aligned}$$

That is, $F^{(k'-1)} = F$, since $h = \sum_{i=0}^{k'-1} h_{k'-1-i} 8^{k'-1-i}$.

Note that (5.6) reads h left-to-right. Similarly, one can read h right-to-left by using $F^{(i)} = A^{h_i}\left(F^{(i-1)}\right)^{2^{-3}}$, for $0 \leq i < k'$, where $h = 2^{3(k'-1)} \sum_{i=0}^{k'-1} h_i 2^{-3(k'-1-i)}$ and $F = \left(F^{(k'-1)}\right)^{2^{3(k'-1)}}$. Based on (5.6), the eight-ary exponentiation architecture of Figure 5.7 is proposed, which is constructed based on the proposed digit-level hybrid-triple GNB multiplier of Figure (5.6) (either Figure 5.6a or Figure 5.6b, depending on whether the target application requires a low area design or a high speed design, respectively). It is noted that, in this figure, the 1 inputs to multiplexers represent the field element $1 = (1, \ldots, 1)$ represented in the GNB. As shown in this figure, the architecture is composed of one DL-PIPO hybrid-triple GNB multiplier and four $2:1$ m-bits multiplexers. The first three multiplexers (0, 1, and 2), respectively, are controlled by the coefficients $s_0^{(i)}$, $s_1^{(i)}$, and $s_2^{(i)}$ of the binary representation of $h_{k'-1-i} = s_0^{(i)} + s_1^{(i)}2 + s_2^{(i)}2^2$, where $k' = \left\lceil \frac{m}{3} \right\rceil$ and $0 \leq h_{k'-1-i} < 8$ for all $0 \leq i < k'$ in (5.6). The last multiplexer, i.e. 3, passes the field element $1 = (1, \ldots, 1)$ during the first iteration, while it selects the 3-fold right cyclic shift of the multiplier's output during the remaining iterations. Therefore, by using this architecture one computes $F = A^h$ after k' runs of the hybrid-triple multiplication. This is equivalent to $k' \times (L + 1)$ clock cycles in the case of parallel preloading of the multiplier, where $L = k$ if a low area hybrid multiplier is used otherwise it becomes $L = k + 1$ for using a high speed hybrid multiplier ($k = \left\lceil \frac{m}{d} \right\rceil$, d is the digit size).

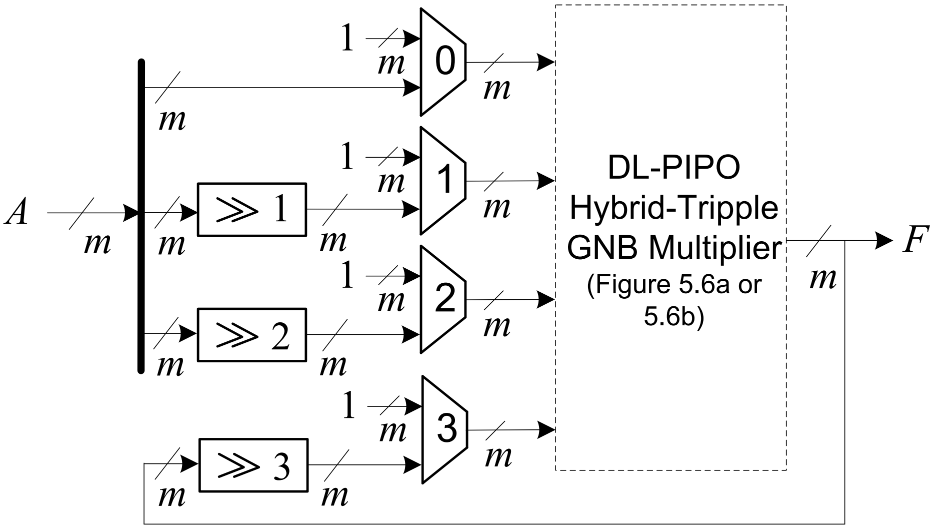


Figure 5.7: Architecture of the proposed eight-ary field exponentiation scheme.

One can see that, the proposed eight-ary exponentiation architecture does not require any storage of precomputed values, while it has almost the same latency, compared to the existing schemes. Also, it is noted that the proposed architecture uses the same latency regardless of the exponent's value. This in turn prevents leakage of time/power dissipation information.

5.5 Conclusion

In this paper, three new architectures for digit-level (DL) single multiplication using GNB have been proposed; two multipliers with fully serial-in-parallel-out (FSIPO) and one with parallel-in-serial-out (PISO). The two DL-FSIPO single GNB multipliers have been proposed for the first time in the literature. They do not require preloading of the inputs and, hence, are advantageous for applications where the parallel loading of inputs is not possible due to limited size of the data-path.

Using the proposed single digit-level multiplier architectures, a new digit-level serial-in-parallel-out (DL-SIPO) hybrid-double GNB multiplier and for the first time in the literature a new digit-level parallel-in-parallel-out (DL-PIPO) hybrid-triple GNB multiplier have been proposed. The proposed digit-level hybrid-double and hybrid-triple multipliers, perform two and three field multiplications, respectively, using the same latency as a single digit-level field multiplication.

As an application of the proposed hybrid-triple multiplier, a new digit-level eight-ary field exponentiation architecture has been presented which offers computational latency similar to the existing eight-ary schemes, however, without requiring storage of precomputed values.

Chapter 6

Digit-Level Architectures for $GF\left(2^{m}\right)$ Multiplication in the PB

In this chapter, and to the best of the author knowledge, two new architectures of $GF\left(2^{m}\right)$ digit-level FSIPO (DL-FSIPO) multipliers for dedicated PBs are proposed for the first time in literature. The new digit-level serial PB architectures generate the output bits in parallel after k iterations. In the new DL-FSIPO PB multiplication schemes, both inputs enter the multiplier digit-by-digit serially, one digit per a clock cycle starting from the most or least significant digit (MSD or LSD), as the computations are carried out. Therefore, the new MSD and LSD DL-FSIPO PB multiplication structures are expected to be advantageous for resource constrained applications where the data-path of the inputs might have limited capacity, specially, when the value of m is large. In addition, by using additional parallel-in-serial-out register, one can also generate the output bits of the proposed DL-FSIPO PB multipliers serially over $2k$ clock cycles. The latter scheme is advantageous over the serial-serial schemes in [46, 14] for performing n consecutive multiplications. The serial-serial schemes presented in [46, 14] require $2kn$ clock cycles to complete n consecutive $GF\left(2^{m}\right)$ digit-level multiplications. The same number of n consecutive digit-level multiplications can be run using only $k\left(n+1\right)$ clock cycles based on the proposed DL-FSIPO multiplication schemes with an additional parallel-in-serial-out output register.

It is noted that, a preliminary MSB bit-level version of this chapter appears in ARITH 22, the 22nd IEEE Symposium On Computer Arithmetic (June 2015). The rest of the chapter is organized as follows. Sections 6.1 and 6.2, respectively, present the proposed MSD and LSD DL-FSIPO PB multiplication schemes. Section 6.3, presents comparisons between the proposed MSD and LSD DL-FSIPO PB multiplication schemes and the other existing counterparts. Section 6.4 gives some conclusions.

6.1 Proposed MSD DL-FSIPO PB Multiplier

This section, presents a new MSD serial multiplier design for dedicated PB which follows a FSIPO inputs/output scheme. To the best of the author knowledge, the proposed MSD DL-FSIPO architecture of the serial PB multiplier is presented for the first time in the literature. The proposed MSD DL-FSIPO PB multiplier conducts the multiplication operation as the digits of the two inputs enter the multiplier in a digit-by-digit order, one digit per a clock cycle (for each input), starting from the most significant digit. Therefore, the proposed MSD DL-FSIPO PB multiplier is advantageous for achieving high throughput for applications where m is large and the parallel preloading of the inputs is not possible due to the limited sizes of the input data-paths. In the following, the required formulations for the MSD DL-FSIPO PB multiplication is first derived. Then, the proposed architecture is shown. The section concludes by studying the space and time complexities.

6.1.1 Formulations

In this section, the required formulations for the proposed MSD DL-FSIPO PB multiplication scheme is derived. First, a recursive digit-level construction of the $GF(2^m)$ elements when represented in the PB is given, by reading the field element digit-by-digit, starting from the most significant digit, as follows.

Definition 6.1.1 *Let α be the root of the field's defining irreducible polynomial of $GF(2^m)$. Let us divide the $GF(2^m)$ element, say $A = (a_{m-1}, \dots, a_0)$ represented in the PB, into $k = \left\lceil \frac{m}{d} \right\rceil$ digits of size d each. That is $A = (A_{k-1}, \dots, A_{k-1-i}, \dots, A_0)$, where $A_{k-1-i} = \sum_{j=0}^{d-1} a_{d(k-1-i)+j}\alpha^j$ is the $(k-1-i)$-th digit and $a_{d(k-1-i)+j} = 0$ for $d(k-1-i)+j \geq m$. Then, A can be constructed recursively, starting from the most significant digit A_{k-1}, as follows:*

$$A^{(i)} = A_{k-1-i} + A^{(i-1)}\alpha^d \tag{6.1}$$

starting at $i = 0$ and obtaining $A = A^{(k-1)}$ at $i = k-1$, given that $A^{(-1)} = 0$.

Proof By using (6.1), for $i = 0, 1, \dots, k-2$ one gets

$$
\begin{aligned}
A^{(0)} =& A_{k-1} + A^{(-1)}\alpha^d \\
=& \sum_{j=0}^{d-1} a_{d(k-1)+j}\alpha^j, \\
A^{(1)} =& A_{k-2} + A^{(0)}\alpha^d \\
=& \sum_{j=0}^{d-1} a_{d(k-2)+j}\alpha^j + \sum_{j=0}^{d-1} a_{d(k-1)+j}\alpha^{j+d} \\
=& \sum_{j=0}^{2d-1} a_{d(k-2)+j}\alpha^j, \\
& \vdots \\
A^{(k-2)} =& A_1 + A^{(k-3)}\alpha^d \\
=& \sum_{j=0}^{d-1} a_{d(1)+j}\alpha^j + \sum_{j=0}^{(k-2)d-1} a_{d(2)+j}\alpha^{j+d} \\
=& \sum_{j=0}^{(k-1)d-1} a_{d+j}\alpha^j,
\end{aligned}
$$

and hence, for $i = k - 1$

$$
\begin{aligned}
A^{(k-1)} =& A_0 + A^{(k-2)}\alpha^d \\
=& \sum_{j=0}^{d-1} a_j\alpha^j + \sum_{j=0}^{(k-1)d-1} a_{d+j}\alpha^{j+d} \\
=& \sum_{j=0}^{kd-1} a_j\alpha^j,
\end{aligned}
$$

that is $A^{(k-1)} = \sum_{i=0}^{m-1} a_i\alpha^i$ since $a_i = 0$ for $i \geq m$, which completes the proof.

Notice that the multiplication by α^d in (6.1) realizes a d-bit left shift and does not require any reduction for $0 \leq i < k$. Based on the recursive construction in (6.1), one obtains the multiplication of any two arbitrary $GF(2^m)$ elements A and B as follows.

Proposition 6.1.2 *Let A and B be two arbitrary* $GF(2^m)$ *elements represented in the PB which is generated by the degree m irreducible polynomial* $p(x) = x^m + \sum_{i=0}^{\omega-2} x^{t_i} + 1$ *with* ω *nonzero terms. Let us define* $C_i = A^{(i)}B^{(i)} \bmod p(\alpha)$*, where* $A^{(i)}$ *and* $B^{(i)}$ *are given in* (6.1) *and* α *is the* *root of $p(x)$. Then, one can compute the multiplication of A and B, i.e. $AB \bmod p(\alpha) = C_{k-1}$, based on the following recurrence on C_i:*

$$\begin{aligned} C_i = \sum_{j=0}^{d-1} \Big[& a_{d(k-1-i)+j} \left(B_{k-1-i} + B^{(i-1)}\alpha^d \right) + \\ & b_{d(k-1-i)+j} A^{(i-1)}\alpha^d \Big] \alpha^j \; mod\; p(\alpha) + \\ & C_{i-1}\alpha^{2d} \; mod\; p(\alpha), \end{aligned} \tag{6.2}$$

$i = 0, \ldots, k-1$, where $C_{-1} = A^{(-1)}B^{(-1)}$ mod $p(\alpha) = 0$.

Proof By using the definition (6.1) for $A^{(i)}$ and $B^{(i)}$ in evaluating $C_i = A^{(i)}B^{(i)} \bmod p(\alpha)$, one obtains

$$\begin{aligned} C_i = & \left(A_{k-1-i} + A^{(i-1)}\alpha^d\right)\left(B_{k-1-i} + B^{(i-1)}\alpha^d\right) \\ & \bmod p(\alpha) \\ = & A_{k-1-i}\left(B_{k-1-i} + B^{(i-1)}\alpha^d\right) + \\ & B_{k-1-i}A^{(i-1)}\alpha^d + C_{i-1}\alpha^{2d} \bmod p(\alpha). \end{aligned}$$

Now, by substituting for $A_{k-1-i} = \sum_{j=0}^{d-1} a_{d(k-1-i)+j}\alpha^j$ and $B_{k-1-i} = \sum_{j=0}^{d-1} b_{d(k-1-i)+j}\alpha^j$ in the above formulation, (6.2) is obtained.

Having $AB \bmod p(\alpha) = A^{(k-1)}B^{(k-1)} \bmod p(\alpha) = C_{k-1}$, then, by iterating for $i = 0, 1, \ldots, k-1$, one obtains the multiplication results after k iterations over (6.1) and (6.2).

Notice that, the left most $(kd - m)$ bits of the most significant digit of the input are zeros. Hence, the highest order coordinate in the intermediate variable elements $A^{(k-2)}$ and $B^{(k-2)}$ is $\alpha^{(k-1)d-1-(kd-m)} = \alpha^{m-d-1}$. Therefore, the multiplication by α^d which appears in the expressions $a_{d(k-1-i)+j}\left(B_{k-1-i} + B^{(i-1)}\alpha^d\right)$ and $b_{d(k-1-i)+j}A^{(i-1)}\alpha^d$ can be accomplished by a simple left shift of d bits without any reduction.

Based on (6.2), the multiplication of the two $GF(2^m)$ elements A and B, is reduced recursively to bit-wise AND operations, field additions, left shifts (for the multiplication by α^d), and multiplications with the fixed field elements α^{2d} and α^j, $0 < j < d$.

The following is an example for illustrating the proposed multiplication scheme.

Example 6.1.3 *Table 6.1 lists the steps for multiplying the two $GF\left(2^3\right)$ field elements $A = \alpha = (0, 1, 0)$ and $B = \alpha^2 = (1, 0, 0)$, represented in the PB $\left\{\alpha^2, \alpha, 1\right\}$ which is defined by the irreducible trinomial $p(x) = x^3 + x + 1$. In this example, $d = 1$ (bit-level multiplication).*

i	a_{2-i}	b_{2-i}	$A^{(i-1)}$	$B^{(i-1)}$
0	$a_2 = 0$	$b_2 = 1$	$A^{(-1)} = 0$	$B^{(-1)} = 0$
1	$a_1 = 1$	$b_1 = 0$	$A^{(0)} = a_2 + A^{(-1)}\alpha = 0$	$B^{(0)} = b_2 + B^{(-1)}\alpha = 1$
2	$a_0 = 0$	$b_0 = 0$	$A^{(1)} = a_1 + A^{(0)}\alpha = 1$	$B^{(1)} = b_1 + B^{(0)}\alpha = \alpha$

i	$a_{2-i}\left(b_{2-i} + B^{(i-1)}\alpha\right)$	$b_{2-i}A^{(i-1)}\alpha$	$C_{i-1}\alpha^2 \bmod p(\alpha)$	C_i
0	$a_2\left(b_2 + B^{(-1)}\alpha\right) = 0$	$b_2A^{(-1)}\alpha = 0$	$C_{-1}\alpha^2 \bmod p(\alpha) = 0$	$C_0 = 0$
1	$a_1\left(b_1 + B^{(0)}\alpha\right) = \alpha$	$b_1A^{(0)}\alpha = 0$	$C_0\alpha^2 \bmod p(\alpha) = 0$	$C_1 = \alpha$
2	$a_0\left(b_0 + B^{(1)}\alpha\right) = 0$	$b_0A^{(1)}\alpha = 0$	$C_1\alpha^2 \bmod p(\alpha) = \alpha + 1$	$C_2 = \alpha + 1 = \alpha^3$

Table 6.1: Example 6.1.3 for multiplying the two $GF\left(2^3\right)$ elements $A = \alpha = (0,1,0)$ and $B = \alpha^2 = (1,0,0)$ using (6.1) and (6.2).

The proposed MSD DL-FSIPO PB multiplication scheme in (6.1) and (6.2) can be implemented for an arbitrary irreducible polynomial considering any digit size d, $0 < d < m$. However, the following remark gives some conditions for efficient hardware realization, based on Theorems 2.9.4 and 2.9.5.

Remark 6.1.4 *Let* $p(x) = x^m + \sum_{i=1}^{\omega-2} x^{t_i} + 1$ *denotes the defining irreducible polynomial with* ω *nonzero terms for* $GF(2^m)$*. Then, by choosing the digit size* d *of the MSD DL-FSIPO PB multiplier such that*

$$d \leq \left\lfloor \frac{m - t_{\omega-2}}{2} \right\rfloor, \tag{6.3}$$

the multiplication of an arbitrary $GF(2^m)$ *element by a fixed field element* α^q*, where* $q \leq 2d$*, can be accomplished efficiently in a single step using using* $q(\omega - 2)$ *two-inputs XOR gates with a propagation delay equivalent to* $\lceil \log_2(q+1) \rceil$ *XOR gate delays.*

According to (6.3), efficient hardware realizations of the MSD DL-FSIPO PB multiplication scheme for the five fields recommended by NIST for ECDSA $GF\left(2^{163}\right)$, $GF\left(2^{233}\right)$, $GF\left(2^{283}\right)$, $GF\left(2^{409}\right)$, and $GF\left(2^{571}\right)$, respectively, offer maximum digit sizes of 78, 79, 135, 161, and 280.

6.1.2 Architecture

This section presents the proposed architecture of the MSD DL-FSIPO PB multiplier, as shown in Figure 6.1a, where $A, B \in GF(2^m)$ represent the inputs to the multiplier, $k = \left\lceil \frac{m}{d} \right\rceil$ and d is the digit size. Figure 6.1b shows the detailed architecture of $\triangle_j$ module at i-th iteration, $0 \leq j < d$ and $0 \leq i < k$. Figure 6.1c shows the architecture of $\otimes$ module.

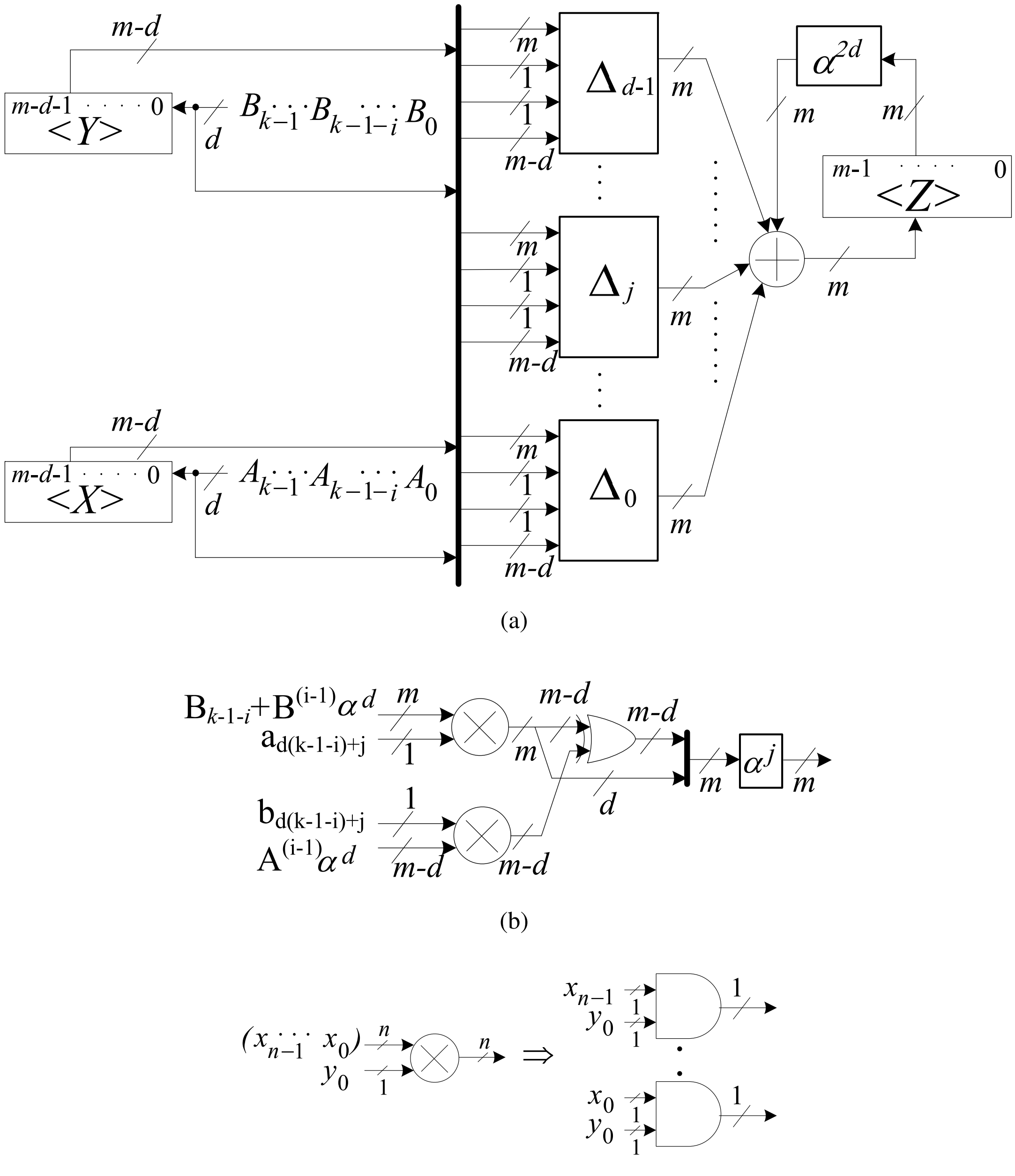


Figure 6.1: (a) Architecture of the proposed MSD DL-FSIPO PB multiplier. (b) Detailed architecture of $\triangle_j$. (c) Architecture of $\otimes$ module.

The architecture in Figure 6.1a is designed based on formulations (6.1) and (6.2). In this design, $\langle X \rangle$ and $\langle Y \rangle$ are left shift registers, which respectively store the bits of $A^{(i-1)}$ and $B^{(i-1)}$ (see (6.1)) during the i-th iteration, for $0 \leq i < k$. It is noted that, the $(m-d)$-th to $(m-1)$-th coordinates are omitted from $\langle X \rangle$ and $\langle Y \rangle$, since these correspond to zeros in all the intermediate elements $A^{(0)}$ to $A^{(k-2)}$ and $B^{(0)}$ to $B^{(k-2)}$ according to (6.1). Then, it is sufficient to have only $(m-d)$-bits, in each of $\langle X \rangle$ and $\langle Y \rangle$. Moreover, according to this, one obtains $A^{(i-1)}\alpha^d$ and

$B^{(i-1)}\alpha^d$ by simple left shifting of d-bits. During the i-th iteration, the vertical thick line in Figure 6.1a represents a $2m$-bits bus which contains the bits of A_{k-1-i}, B_{k-1-i}, $A^{(i-1)}\alpha^d$, and $B^{(i-1)}\alpha^d$. In Figure 6.1a, α^{2d} represents the multiplication of the contents of accumulator $\langle Z\rangle$ by the fixed field element α^{2d}. The vertical thick line in Figure 6.1b represents the concatenation of the lower d-bits of $a_{d(k-1-i)+j}\left(B_{k-1-i} + B^{(i-1)}\alpha^d\right)$ with the $(m-d)$-bit result of XORing the higher bits of $a_{d(k-1-i)+j}\left(B_{k-1-i} + B^{(i-1)}\alpha^d\right)$ to $b_{d(k-1-i)+j}A^{(i-1)}\alpha^d$. This is done in order to compute the expression $a_{d(k-1-i)+j}\left(B_{k-1-i} + B^{(i-1)}\alpha^d\right) + b_{d(k-1-i)+j}A^{(i-1)}\alpha^d$ in (6.2) (the multiplication by α^d is accomplished through the d-bit left shift). In the same figure, the block denoted by α^j represents the multiplication by the fixed field element α^j, $0 \leq j < d$. Hence, by adding the outputs of the field multiplications by all α^j and α^{2d}, one obtains $C_i = A^{(i)}B^{(i)} \bmod p(\alpha)$ in accumulator $\langle Z\rangle$, after the i-th clock trigger, according to (6.2). Therefore, by initializing the three registers $\langle X\rangle$, $\langle Y\rangle$, and $\langle Z\rangle$ of Figure 6.1a, with zeros, the result $C_{k-1} = AB = A^{(k-1)}B^{(k-1)} \bmod p(\alpha)$ is generated in accumulator $\langle Z\rangle$ after k iterations.

As a graphical illustration of Example 6.1.3, Figure 6.2 presents the state of the corresponding $GF\left(2^3\right)$ MSD DL-FSIPO PB multiplier during the different iterations of computations (for multiplying the two field elements $A = \alpha = (0,1,0)$ and $B = \alpha^2 = (1,0,0)$ when $d = 1$), based on the architecture which has been introduced in this section. Figure 6.2a shows the initial state $(i = 0)$. Figure 6.2b shows the state after first clock cycle $(i = 1)$. Figure 6.2c shows the state after second clock cycle $(i = 2)$. Figure 6.2d shows the state after third clock cycle, where the result $\alpha^3 = \alpha + 1$ is stored in the output register which is surrounded by the dotted rectangle.

It is noted that, in this figure, the underlined leftmost bits of $A^{(i-1)}$ and $B^{(i-1)}$, respectively, are always zero, which represent the missing (not required) leftmost FFs in registers $\langle X\rangle$ and $\langle Y\rangle$.

In the following, the space and time complexities of the proposed MSD DL-FSIPO PB multiplier will be studied.

6.1.3 Space and Time Complexities

This section gives the space and time complexities of the proposed MSD DL-FSIPO PB multiplier. Following the design guidelines of Remark 6.1.4, the space complexity of the proposed MSD DL-FSIPO PB multiplier in Figure 6.1a is given by the following proposition.

Proposition 6.1.5 *The total number of gates in the proposed MSD DL-FSIPO PB multiplier of Figure 6.1a is as follows:*

$$\begin{cases} \#ANDs = d\,(2m-d)\,, & \#FFs = 3m-2d, \\ \#XORs = d\left[2m + \frac{(d+3)(\omega-2)}{2} - d\right]. \end{cases} \tag{6.4}$$

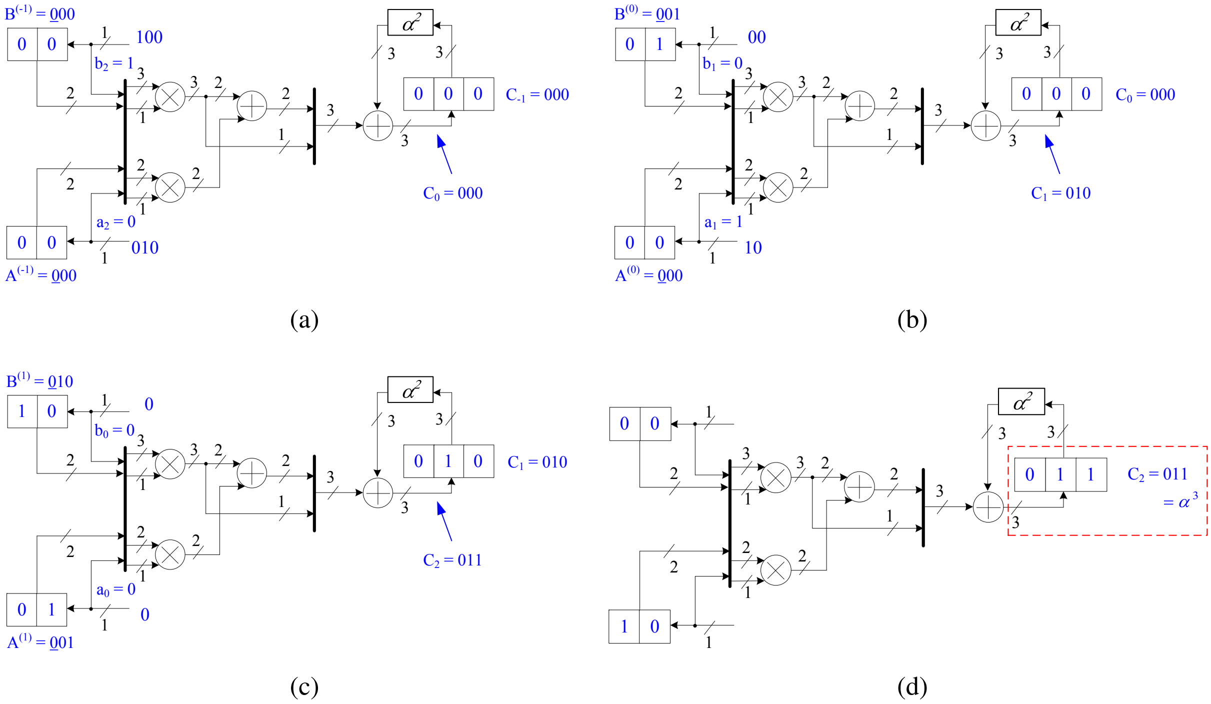


Figure 6.2: The state of the corresponding $GF\left(2^3\right)$ MSD DL-FSIPO PB multiplier for Example 6.1.3, throughout the different iterations of the computation. (a) initial state. $i = 0$. (b) state after first clock cycle. $i = 1$. (c) state after second clock cycle. $i = 2$. (d) state after third clock cycle.

Proof The total number of two-inputs AND gates which is required for the hardware realization of the proposed architecture in Figure 6.1a equals to $d\,(2m - d)$, where $2m - d$ two-inputs AND gate is contributed by each $\triangle_j$ block, $0 \leq j < d$. Similarly, and from the same figure, one finds the total number of FF to be $(m - d) + (m - d) + m = 3m - 2d$. For the total number of two-inputs XOR gates, it consists of the XOR gates in the field addition of $d + 1$ elements, the XOR gates in all the $\triangle_j$ modules, $0 \leq j < d$, in addition to the XOR gates which form the multiplication by the constant α^{2d}. Notice that, for $j = 0$, the multiplication by $\alpha^j = \alpha^0 = 1$ in $\triangle_0$ module is free. Therefore, the total number of two-inputs XOR gates is $dm + d\,(m - d) + \sum_{j=1}^{d-1} j\,(\omega - 2) + 2d\,(\omega - 2) = d\left[2m + \frac{(d+3)(\omega-2)}{2} - d\right]$.

For the time complexity of the proposed MSD DL-FSIPO PB multiplier, it is derived in terms of the propagation delay through the corresponding levels of two-inputs AND and two-inputs XOR gates along the multiplier's longest path, as follows.

Proposition 6.1.6 *The maximum propagation delay (PD) through the proposed MSD DL-*

FSIPO PB multiplier of Figure 6.1a is:

$$\begin{aligned} PD = \max\{&T_A + (1 + \lceil \log_2(d) \rceil + \lceil \log_2(d+1) \rceil)\, T_X, \\ &(\lceil \log_2(2d+1) \rceil + \lceil \log_2(d+1) \rceil)\, T_X\} \end{aligned} \tag{6.5}$$

where T_A denotes the propagation delay of a single two-inputs AND gate.

Proof As one can see from Figure 6.1a, there are two main paths in the proposed design of the MSD DL-FSIPO PB multiplier. The first path is between the shift registers $\langle X \rangle$ and $\langle Y \rangle$, from one side, and the accumulator $\langle Z \rangle$, from the other side. This path has a propagation delay of $T_A + (1 + \lceil \log_2(d) \rceil + \lceil \log_2(d+1) \rceil)\, T_X$, where the propagation delay contributed by a $\triangle_j$ block, $1 \leq j < d$, and the $(d+1)$-inputs field adder, respectively, are $\lceil \log_2(j+1) \rceil\, T_X$ and $\lceil \log_2(d+1) \rceil\, T_X$. The second path lies between the output and input of the accumulator $\langle Z \rangle$, which passes through the $(d+1)$-inputs field adder and the module α^{2d}. This path has a propagation delay equals to $(\lceil \log_2(2d+1) \rceil + \lceil \log_2(d+1) \rceil)\, T_X$, where $\lceil \log_2(2d+1) \rceil\, T_X$ is the propagation delay contributed by the α^{2d} module. Therefore, the propagation delay of the proposed MSD DL-FSIPO PB multiplier takes the value of the maximum propagation delay between these two paths.

In the following, the proposed LSD version of the DL-FSIPO PB multiplier is presented.

6.2 Proposed LSD DL-FSIPO PB Multiplier

In this section, the proposed LSD variant for our DL-FSIPO PB multiplier is presented. To the best of the author knowledge, the proposed LSD DL-FSIPO multiplier is the first such architecture presented for dedicated PB in the literature. The proposed LSD DL-FSIPO PB multiplier reads its two inputs digit-by-digit, one digit per a clock cycle (for each input) while the computations are being performed, starting from the least significant digit. This in return, removes the preloading requirement of the inputs, in advance to computations. It is noted that, the parallel loading of inputs might not be possible in resource constrained applications where the $GF(2^m)$ dimension m is large and the capacity of input data-paths is limited. Hence, the proposed LSD DL-FSIPO PB multiplier has the potential of achieving high output throughput in such applications. The following starts by deriving the required formulations for the LSD DL-FSIPO PB multiplication scheme. This is followed by constructing the corresponding architecture. At the end of this section, the space and time complexities will be studied.

6.2.1 Formulations

This section gives the necessary formulations for constructing the proposed scheme of LSD DL-FSIPO PB multiplication. The following introduces the recursive least significant digit first digit-level construction of the $GF(2^m)$ elements, based on the PB representation.

Definition 6.2.1 *Let α be the root of the $GF(2^m)$ defining irreducible polynomial. Let $A = \sum_{i=0}^{m-1} a_i \alpha^i \in GF(2^m)$ be an arbitrary field element represented in the PB. Divide A into $k = \left\lceil \frac{m}{d} \right\rceil$ digits of size d each. That is, $A = (A_{k-1}, \dots, A_i, \dots, A_0)$, where $A_i = \sum_{j=0}^{d-1} a_{di+j-r} \alpha^j$ is the i-th digit of A such that $a_{di+j-r} = 0$ for $di + j - r < 0$ ($r = kd - m$ represents the number of right padded zeros). Then, one constructs A recursively, starting from its least significant digit, as follows:*

$$A^{(i)} = A_i \alpha^{m-d} + A^{(i-1)} \alpha^{-d}, \tag{6.6}$$

for $i = 0, \dots, k-1$, given that $A^{(-1)} = 0$.

Proof Substituting for $i = 0, 1, \dots, k-2$ in (6.6), one gets

$$\begin{aligned}
A^{(0)} &= A_0 \alpha^{m-d} + A^{(-1)} \alpha^{-d} \\
&= \sum_{j=0}^{d-1} a_{j-r} \alpha^{m-d+j}, \\
A^{(1)} &= A_1 \alpha^{m-d} + A^{(0)} \alpha^{-d} \\
&= \sum_{j=0}^{d-1} a_{d+j-r} \alpha^{m-d+j} + \sum_{j=0}^{d-1} a_{j-r} \alpha^{m-2d+j} \\
&= \sum_{j=0}^{2d-1} a_{j-r} \alpha^{m-2d+j}, \\
&\vdots \\
A^{(k-2)} &= A_{k-2} \alpha^{m-d} + A^{(k-3)} \alpha^{-d} \\
&= \sum_{j=0}^{d-1} a_{(k-2)d+j-r} \alpha^{m-d+j} + \sum_{j=0}^{(k-2)d-1} a_{j-r} \alpha^{m-(k-1)d+j} \\
&= \sum_{j=0}^{(k-1)d-1} a_{j-r} \alpha^{m-(k-1)d+j},
\end{aligned}$$

and hence, for $i = k-1$ one has

$$\begin{aligned}A^{(k-1)} =&A_{k-1}\alpha^{m-d} + A^{(k-2)}\alpha^{-d}\\ =&\sum_{j=0}^{d-1} a_{(k-1)d+j-r}\alpha^{m-d+j} + \sum_{j=0}^{(k-1)d-1} a_{j-r}\alpha^{m-kd+j}\\ =&\sum_{j=0}^{kd-1} a_{j-r}\alpha^{m-kd+j}\\ =&\sum_{j=-r}^{kd-r-1} a_j\alpha^{m+r-kd+j}\end{aligned}$$

and by noticing that $kd - m = r$, then

$$\begin{aligned}A^{(k-1)} &= \sum_{j=-r}^{m-1} a_j\alpha^j\\ &= \sum_{j=0}^{m-1} a_j\alpha^j,\end{aligned}$$

since $a_j = 0$ for $j < 0$, which completes the proof.

It is noted that, the multiplication of $A^{(i-1)}$ by α^{-d} in (6.6) is realized as a d-bit right shift (no reduction is require for $0 \leq i < k$). The following theorem utilizes (6.6) in conducting multiplication of two arbitrary $GF(2^m)$ elements.

Proposition 6.2.2 *Let $C_i = A^{(i)}B^{(i)} \bmod p(\alpha)$, where A and B are two arbitrary $GF(2^m)$ elements, $A^{(i)}$ and $B^{(i)}$ are given in* (6.6), *and α is the root of the field irreducible polynomial $p(x)$. Then, based on* (6.6), *one computes $AB \bmod p(\alpha) = A^{(k-1)}B^{(k-1)} \bmod p(\alpha) = C_{k-1}$ according to the following recurrence on C_i*

$$\begin{aligned}C_i =&\left[\sum_{j=0}^{d-1}\left(a_{di+j-r}\left(B_i\alpha^{m-d} + B^{(i-1)}\alpha^{-d}\right) + b_{di+j-r}A^{(i-1)}\alpha^{-d}\right)\alpha^{j-(d-1)} \bmod p(\alpha)\right]\alpha^{m-1} \bmod p(\alpha) +\\ &C_{i-1}\alpha^{-2d} \bmod p(\alpha), \qquad (6.7)\end{aligned}$$

for $i = 0, \ldots, k-1$ given that $C_{-1} = A^{(-1)}B^{(-1)} \bmod p(\alpha) = 0$, where d is the digit size, $k = \left\lceil \frac{m}{d} \right\rceil$ is the number of iterations, and $B_i = \sum_{j=0}^{d-1} b_{di+j-r}\alpha^j$ is the i-th digit of B such that $b_{di+j-r} = 0$ for $di + j - r < 0$ ($r = kd - m$ represents the number of right padded zeros).

Proof By using definition (6.6) for $A^{(i)}$ and $B^{(i)}$ in evaluating $C_i = A^{(i)}B^{(i)} \bmod p(\alpha)$, one has

$$\begin{aligned}
C_i =& \left(A_i\alpha^{m-d} + A^{(i-1)}\alpha^{-d}\right)\left(B_i\alpha^{m-d} + B^{(i-1)}\alpha^{-d}\right) \\
& \bmod p(\alpha) \\
=& A_i\alpha^{m-d}\left(B_i\alpha^{m-d} + B^{(i-1)}\alpha^{-d}\right) \bmod p(\alpha) + \\
& B_i\alpha^{m-d}A^{(i-1)}\alpha^{-d} \bmod p(\alpha) + \\
& A^{(i-1)}B^{(i-1)}\alpha^{-2d} \bmod p(\alpha) \\
=& \sum_{j=0}^{d-1}\Big[a_{di+j-r}\left(B_i\alpha^{m-d} + B^{(i-1)}\alpha^{-d}\right) + \\
& b_{di+j-r}A^{(i-1)}\alpha^{-d}\Big]\alpha^{m-d+j} \bmod p(\alpha) + \\
& C_{i-1}\alpha^{-2d} \bmod p(\alpha),
\end{aligned}$$

where the last result is obtained by substituting for $A_i = \sum_{j=0}^{d-1} a_{di+j-r}\alpha^j$ in $A_i\alpha^{m-d}\left(B_i\alpha^{m-d} + B^{(i-1)}\alpha^{-d}\right) \bmod p(\alpha)$ and for $B_i = \sum_{j=0}^{d-1} b_{di+j-r}\alpha^j$ in $B_i\alpha^{m-d}A^{(i-1)}\alpha^{-d} \bmod p(\alpha)$, followed by taking α^{m-d+j} as a common factor. Then, by noticing that $\alpha^{m-d-j} = \alpha^{m-1}\alpha^{j-d+1}$, the proof is complete.

Notice that, the right most $r = (kd - m)$ bits in the least significant input digits A_0 and B_0 are zeros. According to this, the lowest coordinate in either $A^{(k-2)}$ or $B^{(k-2)}$ has an order of α^d. Therefore, it is sufficient to accomplish the multiplication by α^{-d} in expressions $B^{(i-1)}\alpha^{-d}$ and $A^{(i-1)}\alpha^{-d}$ of (6.7) by simple d-bit right shifts without any reductions. Now, since $A = A^{(k-1)}$ and $B = B^{(k-1)}$, then, by iterating on (6.7) for $i = 0, 1, \ldots, k-1$, one obtains $AB \bmod p(\alpha) = A^{(k-1)}B^{(k-1)} \bmod p(\alpha) = C_{k-1}$.

Based on (6.7), the multiplication of A and B is reduced, recursively, to bit-wise AND operations, field additions, right shifts (for the multiplication by α^{-d}), in addition to the multiplications with the constant elements α^{m-1} and α^{-q} where$q$ is a positive integer such that $q \leq 2d$. The following is an example illustrating the proposed multiplication scheme in (6.7).

Example 6.2.3 *Table 6.2 lists the steps (according to formulations* (6.6) *and* (6.7)*) for multiplying the two* $GF\left(2^3\right)$ *field elements* $A = \alpha = (0, 1, 0)$ *and* $B = \alpha^2 = (1, 0, 0)$*, represented in the PB* $\left\{\alpha^2, \alpha, 1\right\}$ *which is defined by the irreducible trinomial* $p(x) = x^3 + x + 1$*. In this example,* $d = 1$ *(bit-level multiplication), and hence,* $k = \left\lceil \frac{3}{1} \right\rceil = 3$ *and* $r = 3 \times 1 - 3 = 0$.

The following presents the formulations needed for realizing the operations of multiplying an arbitrary field element by the constants α^{m-1} and α^{-q}, where $q \leq 2d$ for some positive integer d.

i	a_i	b_i	$A^{(i-1)}$	$B^{(i-1)}$
0	$a_0 = 0$	$b_0 = 0$	$A^{(-1)} = 0$	$B^{(-1)} = 0$
1	$a_1 = 1$	$b_1 = 0$	$A^{(0)} = a_0\alpha^2 + A^{(-1)}\alpha^{-1} = 0$	$B^{(0)} = b_0\alpha^2 + B^{(-1)}\alpha^{-1} = 0$
2	$a_2 = 0$	$b_2 = 1$	$A^{(1)} = a_1\alpha^2 + A^{(0)}\alpha^{-1} = \alpha^2$	$B^{(1)} = b_1\alpha^2 + B^{(0)}\alpha^{-1} = 0$

	$X_i = a_i\left(b_i\alpha^2 + B^{(i-1)}\alpha^{-1}\right)$	$Y_i = b_iA^{(i-1)}\alpha^{-1}$	$Z_i = C_{i-1}\alpha^{-2} \bmod p(\alpha)$	$C_i = (X_i + Y_i)\,\alpha^2 \bmod p(\alpha) + Z_i$
0	$a_0\left(b_0\alpha^2 + B^{(-1)}\alpha^{-1}\right) = 0$	$b_0A^{(-1)}\alpha^{-1} = 0$	$C_{-1}\alpha^{-2} \bmod p(\alpha) = 0$	$C_0 = 0$
1	$a_1\left(b_1\alpha^2 + B^{(0)}\alpha^{-1}\right) = 0$	$b_1A^{(0)}\alpha^{-1} = 0$	$C_0\alpha^{-2} \bmod p(\alpha) = 0$	$C_1 = 0$
2	$a_2\left(b_2\alpha^2 + B^{(1)}\alpha^{-1}\right) = 0$	$b_2A^{(1)}\alpha^{-1} = \alpha$	$C_1\alpha^{-2} \bmod p(\alpha) = 0$	$C_2 = \alpha^3 \bmod p(\alpha) = \alpha + 1$

Table 6.2: Example 6.2.3 for multiplying the two $GF\left(2^3\right)$ elements $A = \alpha = (0,1,0)$ and $B = \alpha^2 = (1,0,0)$ using (6.6) and (6.7).

First, the multiplication by the constants α^{m-1} is considered. Let $p(x) = x^m + \sum_{i=1}^{\omega-2} x^{t_i} + 1$ be the generating irreducible polynomial of $GF(2^m)$, where α is its root. According to Theorem 2.9.5, the multiplication of an arbitrary $GF(2^m)$ element A by the constant element α^{m-1} can be accomplished efficiently in one step if $m - 1 \leq m - t_{\omega-2}$. This means that, $t_{\omega-2} = 1$, and hence, $p(x)$ is an irreducible trinomial of the form $x^m + x + 1$. Since this form of $p(x)$ is not common, the following general formulation for multiplying an arbitrary field element by the constant element α^{m-1} is considered.

Proposition 6.2.4 *Let the elements of $GF(2^m)$ be represented in the PB which is defined by the irreducible $p(x) = x^m + \sum_{i=1}^{\omega-2} x^{t_i} + 1$. Let α be a root of $p(x)$. Denote by $[\uparrow i]$ and $[\downarrow i]$ the operations of up and down i-bit shifts, as defined by Definition 2.9.2. Let the m-bits vertical vector $\left[a_0^{m-1} \;\; \ldots \;\; a_{m-1}^{m-1}\right]^T$ (T is the vector transposition) represents the coordinates of the result out of multiplying an arbitrary $GF(2^m)$ element $A = \sum_{i=0}^{m-1} a_i\alpha^i$ by α^{m-1}, that is $A\alpha^{m-1} \bmod p(\alpha) = \sum_{i=0}^{m-1} a_i^{m-1}\alpha^i$, then*

$$\begin{bmatrix} a_0^{m-1} \\ \vdots \\ a_{m-2}^{m-1} \\ a_{m-1}^{m-1} \end{bmatrix} = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ a_0 \end{bmatrix} + \sum_{j=0}^{\omega-2}\left(\sum_{i=0}^{n-1}\begin{bmatrix} a_1 \\ \vdots \\ a_{m-1} \\ 0 \end{bmatrix}[\uparrow l_i]\right)\left[\downarrow t_j\right]. \tag{6.8}$$

Here, $t_0 = 0$, n is the number of nonzero entries in column zero of the $(m-1) \times m$ binary reduction matrix **Q** *[72], and l_i denotes the row location of the i-th nonzero entry in this column.*

Proof From Section 2.9.1.1, by setting B in (2.2) and (2.3) to $B = \alpha^{m-1} = \left(1, \underbrace{0, \ldots, 0}_{m-1}\right)$, one obtains (6.8).

Next, the multiplication of an arbitrary field element A represented in the PB by the constant element α^{-q}, i.e. $A\alpha^{-q} \bmod p(\alpha)$, where q is a positive integer, is considered. The following are some conditions for the efficient hardware realization of this operation.

Proposition 6.2.5 *Assume $p(x) = x^m + \sum_{i=1}^{\omega-2} x^{t_i} + 1$ is the field irreducible polynomial which defines $GF(2^m)$. Let α denotes the root of $p(x)$. Therefore, for a positive integer $q \leq t_1$, the coordinates of α^{-q} are obtained in a single step, as follows*

$$\alpha^{-q}\, mod\, p(\alpha) = \left(\alpha^m + \sum_{i=1}^{\omega-2} \alpha^{t_i}\right)\alpha^{-q}. \tag{6.9}$$

Proof Since $p(\alpha) = 0$, then $\alpha^m + \sum_{i=1}^{\omega-2} \alpha^{t_i} = 1$, and by multiplying both sides by α^{-q} one gets

$$\alpha^{-q} \bmod p(\alpha) = \alpha^{m-q} + \sum_{i=1}^{\omega-2} \alpha^{t_i-q},$$

in which $t_i - q \geq 0$ for all $1 \leq i \leq \omega - 2$ if $q \leq t_1$. Then, the assertion is true.

Proposition 6.2.6 *Assume $p(x) = x^m + \sum_{i=1}^{\omega-2} x^{t_i} + 1$ is the field irreducible polynomial which defines $GF(2^m)$. Denote by α the root of $p(x)$. Let $A = (a_{m-1}, \ldots, a_0)$ be an arbitrary $GF(2^m)$ element represented in the PB. Therefore, for a positive integer $q \leq t_1$, the coordinates of $A\alpha^{-q}\, mod\, p(\alpha) = \sum_{i=0}^{m-1} a_i\alpha^{i-q}\, mod\, p(\alpha)$ are obtained in a single step, as follows:*

$$\begin{aligned} A\alpha^{-q}\, mod\, p(\alpha) = &\sum_{i=q}^{m-1} a_i\alpha^{i-q} \\ &\sum_{i=0}^{q-1} a_i\left(\alpha^m + \sum_{j=1}^{\omega-2} \alpha^{t_j}\right)\alpha^{i-q}. \end{aligned} \tag{6.10}$$

Proof Note that, $A\alpha^{-q} \bmod p(\alpha) = \sum_{i=q}^{m-1} a_i\alpha^{i-q} + \sum_{i=0}^{q-1} a_i\alpha^{i-q} \bmod p(\alpha)$. Since it is given that $q \leq t_1$, then, one can compute α^{-1} through α^{-q} using (6.9). This completes the proof.

The following is a remark about the selection of the digit size for efficient hardware implementation of the proposed LSD DL-FSIPO PB multiplier.

Remark 6.2.7 *Let $p(x) = x^m + \sum_{i=1}^{\omega-2} x^{t_i} + 1$ denotes the defining irreducible polynomial with ω nonzero terms for $GF(2^m)$. Then, by choosing the digit size d of the LSD DL-FSIPO PB multiplier such that*

$$d \leq \left\lfloor \frac{t_1}{2} \right\rfloor, \tag{6.11}$$

the multiplication of an arbitrary $GF(2^m)$ element by the fixed field element α^{-q}, where q is a positive integer satisfying $q \leq 2d$, can be accomplished in a single step.

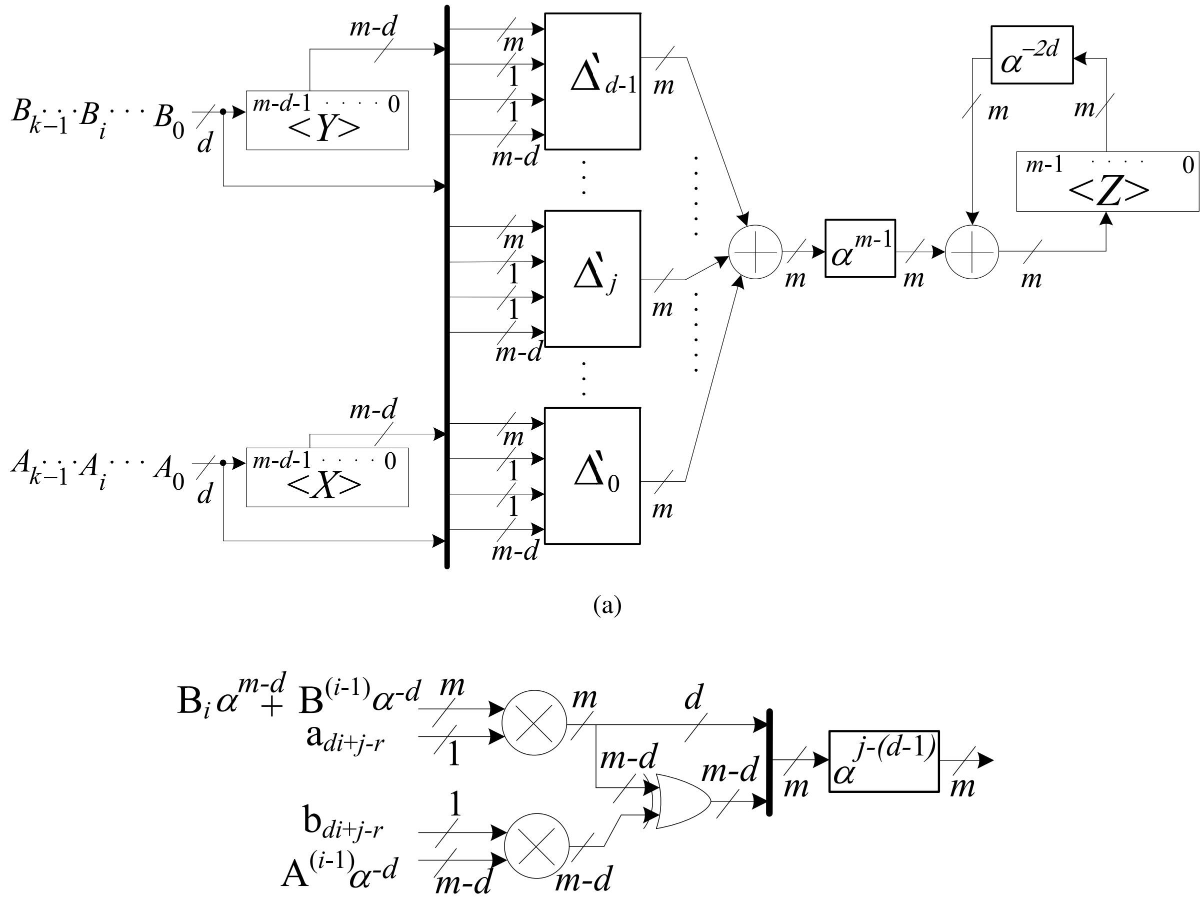
According to (6.11), efficient hardware realizations of the LSD DL-FSIPO PB multiplication scheme for the five fields recommended by NIST for ECDSA $GF\left(2^{163}\right)$, $GF\left(2^{233}\right)$, $GF\left(2^{283}\right)$, $GF\left(2^{409}\right)$, and $GF\left(2^{571}\right)$, respectively, offer maximum digit sizes of 1, 37, 2, 43, and 1. It is evident that the MSD version of the DL-FSIPO PB multiplier provides larger flexibility on the selection of digit sizes for ECDSA recommended fields.

In the following section, the architecture of the proposed LSD DL-FSIPO PB multiplier is presented.

6.2.2 Architecture

This section presents the proposed architecture of the LSD DL-FSIPO PB multiplier, as shown in Figure 6.3a. Figure 6.3b shows the detailed architecture of the $\triangle_j^{'}$ module at i-th iteration, $0 \leq j < d$ and $0 \leq i < k$. The component $\otimes$ is shown in more details in Figure 6.1b. Also, it is noted that $r = kd - m$ is the number of right padded zeros.

The architecture of Figure 6.3a is constructed based on (6.6) and (6.7). In the following illustration denotes by A and B the input field elements to the multiplier, while $A^{(i)}$ and $B^{(i)}$ are given in (6.6), and C_i is defined in (6.7). In Figure 6.3a, $\langle X \rangle$ and $\langle Y \rangle$ are right shift registers. $\langle X \rangle$ stores the bits of $A^{(i-1)}$, while $\langle Y \rangle$ stores the bits of $B^{(i-1)}$, during the i-th iteration of the k clock cycles of computations. Notice that, the least significant digit of either $A^{(i-1)}$ or $B^{(i-1)}$ is zero for all $i < k$ (only $A^{(k-1)} = A_{k-1}\alpha^{m-d} + A^{(k-2)}\alpha^{-d}$ and $B^{(k-1)} = B_{k-1}\alpha^{m-d} + B^{(k-2)}\alpha^{-d}$ have nonzero least significant digits). Also, the rightmost r bits of $A^{(k-2)}$ and $B^{(k-2)}$ are zeros (padding zeros). Therefore, during the last iteration $i = k - 1$, one has $\langle X \rangle = A^{(k-2)} = \left\langle a_{m-1-d}, \ldots, a_0, \underbrace{0, \ldots, 0}_{d} \right\rangle$ and $\langle Y \rangle = B^{(k-2)} = \left\langle b_{m-1-d}, \ldots, b_0, \underbrace{0, \ldots, 0}_{d} \right\rangle$, and hence, it is sufficient to have only $(m - d)$-bits in each of $\langle X \rangle$ and $\langle Y \rangle$. The vertical thick line in Figure 6.3a represents a $2m$-bit bus carrying the bits of A_i, B_i, $\langle X \rangle = A^{(i-1)}$, and $\langle Y \rangle = B^{(i-1)}$, during the i-th iteration, for $0 \leq i < k$. During the i-th iteration, the m-bit input $B_i\alpha^{m-d} + B^{(i-1)}\alpha^{-d}$ in Figure 6.3b represents $B^{(i)}$ (see (6.6)) and is constructed by concatenating the d-bits from B_i (higher bits) with the $(m - d)$-bits from $B^{(i-1)}$ (lower bits). In the same figure, the vertical thick line concatenates the d bits from $a_{di+j-r}B^{(i)}$ (higher bits) to the $m - d$ bits (lower) resulting from bit-wise XORing the lower $m - d$ bits of $a_{di+j-r}B^{(i)}$ with $b_{di+j-r}A^{(i-1)}\alpha^{-d}$. Here, j denotes the number of the block $\triangle_j^{'}$ in Figure 6.3a and its value satisfies $0 \leq j < d$. The multiplication of the latter concatenated m-bit signal (of Figure 6.3b) by $\alpha^{j-(d-1)}$ generates the m-bit output of the corresponding $\triangle_j^{'}$ block in Figure 6.3a (that is $\left[a_{di+j-r}\left(B_i\alpha^{m-d} + B^{(i-1)}\alpha^{-d}\right) + b_{di+j-r}A^{(i-1)}\alpha^{-d}\right]\alpha^{j-(d-1)}$). The output of block α^{m-1} represents the result of the fixed multiplication of the summation of the outputs of all $\triangle_j^{'}$ by



(a)

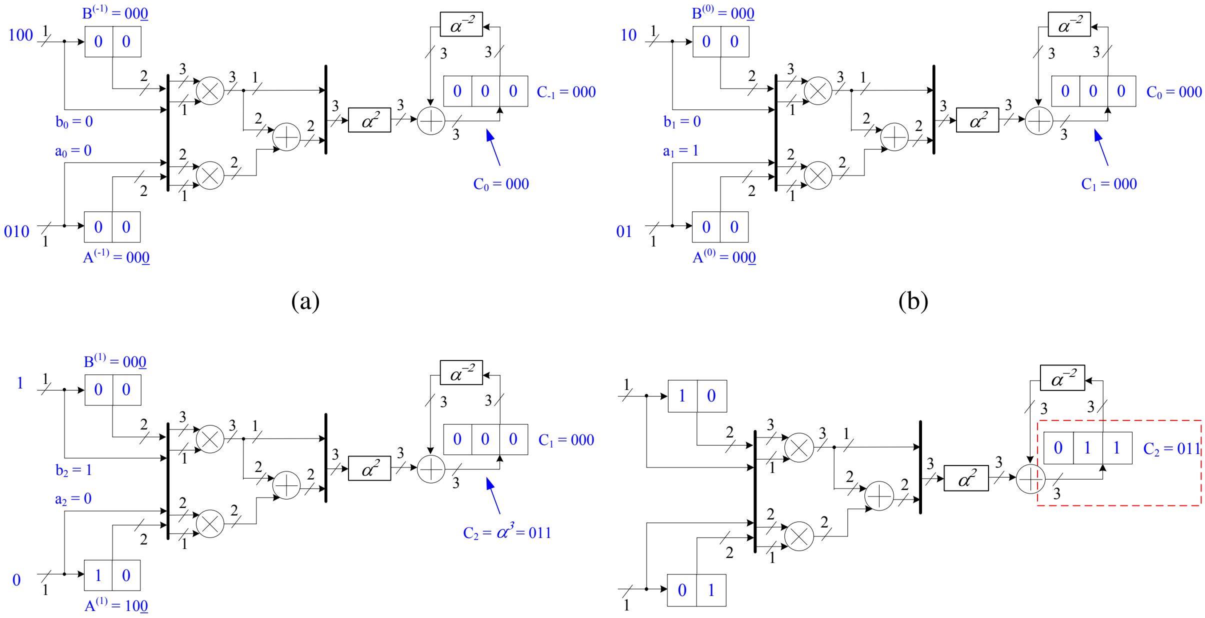
(b)

Figure 6.3: (a) Architecture of the proposed LSD DL-FSIPO PB multiplier. (b) Detailed architecture of $\triangle'_j$ at i-th iteration.

α^{m-1}. At the i-th clock trigger, the accumulator $\langle Z\rangle$ is updated by adding the output of block α^{m-1} to the output of block α^{-2d}. Block α^{-2d} represents the multiplication of the current state of register $\langle Z\rangle$ by the fixed element α^{-2d}. Therefore, after the i-th clock signal, $\langle Z\rangle = C_i$, according to (6.7). Then, by initializing the three registers $\langle X\rangle$, $\langle Y\rangle$, and $\langle Z\rangle$ in Figure 6.3a with zeros, one generates the multiplication result $AB \bmod p(\alpha) = A^{(k-1)}B^{(k-1)} \bmod p(\alpha) = C_{k-1}$ in accumulator $\langle Z\rangle$ after k iterations.

Figure 6.4 presents a graphical illustration showing the state of the $GF\left(2^3\right)$ least significant bit first bit-level (LSB BL-FSIPO) PB multiplier during the different iterations of computations, for multiplying the two field elements $A = \alpha$ and $B = \alpha^2$ in Example 6.2.3, based on the architecture of Figure 6.3a, where $d = 1$. Figure 6.4a shows the initial state ($i = 0$). Figure 6.4b shows the state after first clock cycle ($i = 1$). Figure 6.4c shows the state after second clock cycle ($i = 2$). Figure 6.4d shows the state after third clock cycle, where the result $\alpha^3 = \alpha + 1$

is stored in the output register which is surrounded by the dotted rectangle. It is noted that, in this figure, the underlined rightmost bit of each of $A^{(i-1)}$ and $B^{(i-1)}$, respectively, is always zero, which represents the missing (not required) rightmost FF in each of register $\langle X \rangle$ and register $\langle Y \rangle$, respectively.



(a) (b)

(c) (d)

Figure 6.4: The state of the corresponding $GF\left(2^3\right)$ LSB BL-FSIPO PB multiplier for Example 6.2.3, throughout the different iterations of the computation. (a) initial state. $i = 0$. (b) state after first clock cycle. $i = 1$. (c) state after second clock cycle. $i = 2$. (d) state after third clock cycle.

In the following, the space and time complexities of the proposed LSD DL-FSIPO PB multiplier will be studied.

6.2.3 Space and Time Complexities

This section starts by deriving the space and time complexities for the multiplication of an arbitrary field element, represented in the PB, by the constants α^{m-1} and α^{-q} (for $0 \leq q \leq t_1$), respectively, where $\alpha \in GF(2^m)$ is the root of the field's generating irreducible polynomial. After this, the space and time complexities of the proposed LSD DL-FSIPO PB multiplier are considered.

The following lemma gives the space and time complexities for the multiplication of an arbitrary field element by the constant element α^{m-1}.

Lemma 6.2.8 *The hardware realization of the multiplication of an arbitrary* $GF\left(2^m\right)$ *element* $A = \sum_{i=0}^{m-1} a_i\alpha^i$ *by the constant element* α^{m-1} *according to* (6.8) *requires the following number of two-inputs XOR gates*

$$N_{\alpha^{m-1}} = (m-1)(n+\omega-3) + (\omega-2) - \sum_{i=1}^{n-1} l_i - \sum_{j=1}^{\omega-2} t_j, \tag{6.12}$$

and a propagation delay of

$$T_{\alpha^{m-1}} = \left(\left\lceil \log_2(n)\right\rceil + \left\lceil \log_2(\omega-1)\right\rceil\right) T_X, \tag{6.13}$$

where $p(x) = x^m + \sum_{i=1}^{\omega-2} x^{t_i} + 1$ *is the field's generating irreducible polynomial with* ω *nonzero terms,* n *is the number of nonzero entries in column zero of the* $(m-1)\times m$ *binary reduction matrix* $\mathbf{Q}$ *[72], and* l_i *denotes the row location of the* i*-th nonzero entry in this column.*

Proof The generation of $\mathbf{v}_1 = \sum_{i=0}^{n-1}\begin{bmatrix} a_1 & a_2 & \ldots & a_{m-1} & 0\end{bmatrix}^T [\uparrow l_i]$ in (6.8) requires $\sum_{i=1}^{n-1}(m-1-l_i) = (n-1)(m-1) - \sum_{i=1}^{n-1} l_i$ two-inputs XOR gates. After this, one needs another $\sum_{j=1}^{\omega-2}\left(m-t_j\right) = (\omega-2)m - \sum_{j=1}^{\omega-2} t_j$ two-inputs XORs for the realization of $\mathbf{v}_2 = \mathbf{v}_3 + \sum_{j=0}^{\omega-2} \mathbf{v}_1\left[\downarrow t_j\right]$ in (6.8), where $\mathbf{v}_3 = \begin{bmatrix} 0 & 0 & \ldots & a_0\end{bmatrix}^T$ and $t_0 = 0$. Therefore, by adding these values we get (6.12). Similarly, one obtains (6.13) by adding the propagation delays contributed by the generation of $\mathbf{v}_1$ (that is, $\left\lceil \log_2(n)\right\rceil T_X$) and $\mathbf{v}_2$ (that is $\left\lceil \log_2(\omega-1)\right\rceil T_X$, since $\mathbf{v_1}[\downarrow 0] + \mathbf{v_3}$ does not require any XORing).

Corollary 6.2.9 *If* $1 < t_{\omega-2} \leq \frac{m+1}{2}$, *then,* $N_{\alpha^{m-1}}$ *and* $T_{\alpha^{m-1}}$ *in* (6.12) *and* (6.13)*, respectively, become*

$$N_{\alpha^{m-1}} = (\omega-2)(m-1), \tag{6.14}$$

and

$$T_{\alpha^{m-1}} = 2\left\lceil \log_2(\omega-1)\right\rceil T_X. \tag{6.15}$$

Proof Since $1 < t_{\omega-2} \leq \frac{m+1}{2}$, then, $n = \omega - 1$, $l_0 = 0$, and $l_i = m - t_i$ for $1 \leq i < \omega - 1$ (see Remark 2.9.3). Based on this, $N_{\alpha^{m-1}}$ becomes

$N_{\alpha^{m-1}}$

$$\begin{aligned}
&= (m-1)(n+\omega-3) + (\omega-2) \\
&\quad - \sum_{i=1}^{n-1} l_i - \sum_{j=1}^{\omega-2} t_j \\
&= (m-1)(\omega-1+\omega-3) + (\omega-2) \\
&\quad - \sum_{i=1}^{\omega-2} (m-t_i) - \sum_{j=1}^{\omega-2} t_j \\
&= (2\omega-4)m - (2\omega-4) + (\omega-2) \\
&\quad - \sum_{i=1}^{\omega-2} m \\
&= (\omega-2)(m-1).
\end{aligned}$$

Similarly, $T_{\alpha^{m-1}}$ becomes
$T_{\alpha^{m-1}}$

$$\begin{aligned}
&= \left(\lceil \log_2(n) \rceil + \lceil \log_2(\omega-1) \rceil\right) T_X \\
&= \left(\lceil \log_2(\omega-1) \rceil + \lceil \log_2(\omega-1) \rceil\right) T_X \\
&=2 \lceil \log_2(\omega-1) \rceil T_X.
\end{aligned}$$

The following targets efficient hardware implementation of the proposed LSD DL-FSIPO PB multiplier. Therefore, values of d which satisfy the condition of (6.11) are only considered. In this context, the following lemma gives the space and time complexities for the hardware realization of the multiplication of an arbitrary field element A by the constant element α^{-q}, based on the formulation (6.9).

Lemma 6.2.10 *The hardware realization of the multiplication of an arbitrary* $GF(2^m)$ *element* $A = \sum_{i=0}^{m-1} a_i \alpha^i$ *by the constant element* α^{-q}*, according to* (6.9)*, requires at most a number of two-inputs XOR gates equals to*

$$N_{\alpha^{-q}} = q(\omega-2), \tag{6.16}$$

and a propagation delay of

$$T_{\alpha^{-q}} = \lceil \log_2(q+1) \rceil T_X, \tag{6.17}$$

where $p(x) = x^m + \sum_{i=1}^{\omega-2} x^{t_i} + 1$ *is the field's generating irreducible polynomial with* ω *nonzero terms and* d *satisfies the condition of* (6.11).

Proof According to (6.9), we have
$A\alpha^{-q} \bmod p(\alpha)$

$$
\begin{aligned}
&= \sum_{i=0}^{m-1} a_i \alpha^{i-q} \\
&= \sum_{i=q}^{m-1} a_i \alpha^{i-q} + \sum_{i=0}^{q-1} a_i \left(\alpha^m + \sum_{j=1}^{\omega-2} \alpha^{t_j} \right) \alpha^{i-q}.
\end{aligned}
$$

As it is shown in Figure 6.5, the hardware realization of the above formulation requires a number of two-inputs XOR gates equals to $q(\omega-1) - q = q(\omega-2)$ and a propagation delay equivalent to $\lceil \log_2 (q+1) \rceil T_X$.

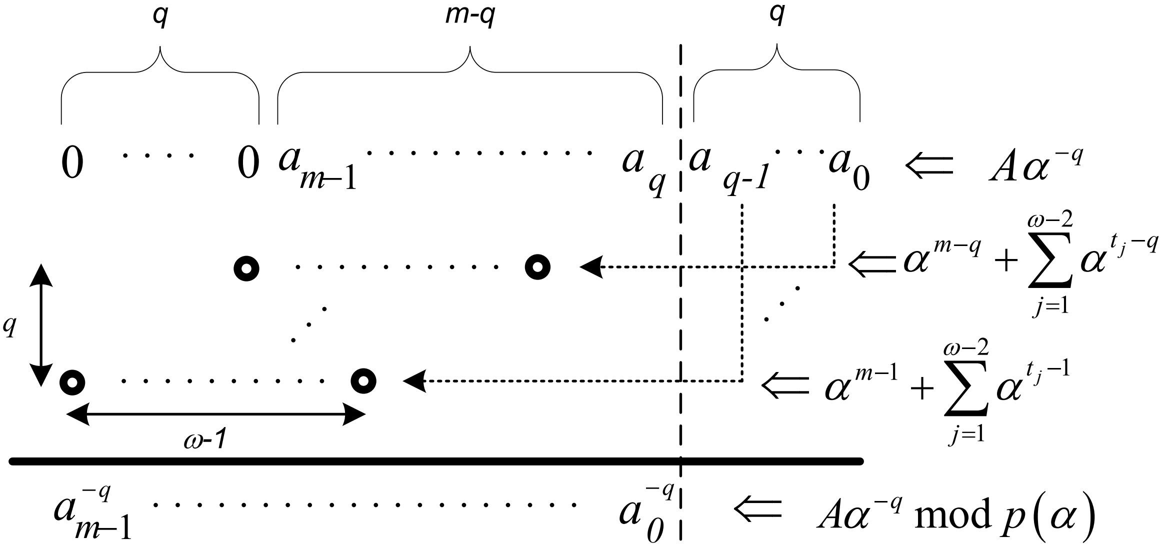

q
m-q
q
$0 \cdots\cdots 0 \; a_{m-1} \cdots\cdots a_q \; a_{q-1} \cdots a_0$
$\Leftarrow A\alpha^{-q}$
$\Leftarrow \alpha^{m-q} + \sum_{j=1}^{\omega-2} \alpha^{t_j - q}$
q
$\Leftarrow \alpha^{m-1} + \sum_{j=1}^{\omega-2} \alpha^{t_j - 1}$
ω-1
$a_{m-1}^{-q} \cdots\cdots a_0^{-q}$
$\Leftarrow A\alpha^{-q} \bmod p(\alpha)$

Figure 6.5: Multiplying an arbitrary $GF(2^m)$ element by the constant α^{-q} where$p(x) = x^m + \sum_{i=1}^{\omega-2} x^{t_i} + 1$ is the field's generating irreducible polynomial with ω nonzero terms and $q \le t_1$ (condition of (6.11)).

Now, the space complexity of the proposed LSD DL-FSIPO PB multiplier in Figure 6.3a is given. What follows assumes the conditions $1 < t_{\omega-2} \le \frac{m+1}{2}$ (which is true for the five binary extension fields recommended by NIST for ECDSA [12]) and $d \le \left\lfloor \frac{t_1}{2} \right\rfloor$ are valid.

Proposition 6.2.11 *By following the conditions* $1 < t_{\omega-2} \le \frac{m+1}{2}$ *and* $d \le \left\lfloor \frac{t_1}{2} \right\rfloor$*, then, the total number of gates in the hardware realization of the proposed LSD DL-FSIPO PB multiplier of*

Figure 6.3a is as follows:

$$\begin{cases} \#ANDs = & d\,(2m-d) \\ \#XORs = & (2d+\omega-2)\,m-(\omega-2)-d\left[\frac{(d-3)(\omega-2)}{2}\right]+1\,. \\ \#FFs = & 3m-2d \end{cases} \tag{6.18}$$

Proof The total number of two-inputs AND gates required for the hardware realization of the proposed architecture in Figure 6.3a equals to $d\,(m+m-d) = d\,(2m-d)$, where each $\triangle'_j$ block contributes $m+m-d = 2m-d$ two-inputs AND gates ($0 \leq j < d$). From the same figure, one finds the total number of FFs in registers $\langle X\rangle$, $\langle Y\rangle$, and $\langle Z\rangle$ to be $(m-d)+(m-d)+m = 3m-2d$. For the total number of two-inputs XOR gates, it consists of the XOR gates in the field addition of d elements plus those in the field addition of 2 elements (that is $(d-1)\,m+m = dm$ XORs), the XOR gates which form the multiplication by the constant α^{-2d} (that is $2d\,(\omega-2)$ see (6.16)), the XOR gates which form the multiplication by the constant α^{m-1} (given by (6.14)), in addition to the XOR gates in all the $\triangle'_j$ modules, $0 \leq j < d$. Notice that, each $\triangle'_j$ module requires $m-d+N_{\alpha^{j-(d-1)}}$ two-input XOR gates, out of which $N_{\alpha^{j-(d-1)}}$ (given by (6.16)) two-input XOR gates are required to realize the multiplication by $\alpha^{j-(d-1)}$. Therefore, the total number of two-inputs XOR gates is $dm + 2d\,(\omega-2) + d\,(m-d) + \sum_{j=0}^{d-1} N_{\alpha^{j-(d-1)}} + N_{\alpha^{m-1}}$, and by substituting for $\sum_{j=0}^{d-1} N_{\alpha^{j-(d-1)}} = \sum_{i=0}^{d-1} N_{\alpha^{-i}} = \sum_{i=0}^{d-1} i\,(\omega-2) = \frac{d(d-1)(\omega-2)}{2}$ and for $N_{\alpha^{m-1}} = (\omega-2)\,(m-1)$, according to (6.16) and (6.14), respectively, $(2d+\omega-2)\,m-(\omega-2)-d\left[\frac{(d-3)(\omega-2)}{2}\right]+1$ is obtained.

The time complexity of the proposed LSD DL-FSIPO PB multiplier, in terms of the propagation delay of the corresponding levels of two-inputs AND and two-inputs XOR gates along the multiplier's longest path, is as follows. Again, assuming the conditions $1 < t_{\omega-2} \leq \frac{m+1}{2}$ and $d \leq \left\lfloor \frac{t_1}{2} \right\rfloor$ are valid.

Proposition 6.2.12 *By following the conditions* $1 < t_{\omega-2} \leq \frac{m+1}{2}$ *and* $d \leq \left\lfloor \frac{t_1}{2} \right\rfloor$, *then, the maximum propagation delay for the hardware realization of the proposed LSD DL-FSIPO PB multiplier in Figure 6.3a is independent of the binary extension field's dimension (i.e., m), and is equal to*

$$PD = T_A + 2\,(1+\lceil \log_2 d \rceil + \lceil \log_2 (\omega-1) \rceil)\,T_X. \tag{6.19}$$

Proof There are two main paths in Figure 6.3a. The first path extends between the input registers ($\langle X\rangle$ and $\langle Y\rangle$) and the output accumulator $\langle Z\rangle$. The second path extends between the output and input of register $\langle Z\rangle$. For the former path, notice that the multiplication by $\alpha^{-(d-1)}$ requires higher propagation delay than the multiplications by the constants α^{-1} through $\alpha^{-(d-2)}$.

Therefore, the critical path in Figure 6.3a between input registers ($\langle X \rangle$ and $\langle Y \rangle$) and the output accumulator $\langle Z \rangle$ passes through module $\triangle_0^{'}$. This propagation delay consists of $T_A + T_X + T_{\alpha^{-(d-1)}}$ contributed by module $\triangle_0^{'}$, $\lceil \log_2 d \rceil T_X$ contributed by the d inputs field adder, T_X contributed by the 2 inputs field adder, in addition to $T_{\alpha^{m-1}}$ which is contributed by the multiplication with α^{m-1} (given by (6.15)). This adds up to $T_A + 2\left(1 + \lceil \log_2 d \rceil + \lceil \log_2 (\omega - 1) \rceil\right) T_X$. On the other hand, the propagation delay of the path between the output and input of accumulator $\langle Z \rangle$ is equivalent to $T_X + T_{\alpha^{-2d}} = \left(1 + \lceil \log_2 (2d + 1) \rceil\right) T_X$. Hence, the propagation delay in (6.19) is the maximum between these two paths.

In the following, a comparison between the proposed DL-FSIPO PB multipliers and other existing digit-level serial PB multiplication schemes is conducted.

6.3 Comparisons

In this section, the proposed DL-FSIPO PB multipliers are compared to other existing serial PB multipliers. For this purpose, the propagation delay and space complexity for the different serial PB multiplication schemes are listed in Table 6.3. In this table, space complexity is reported in terms of number of FF, two-inputs AND and XOR gates, and 2-to-1 1-bit multiplexers (for either logic implementation or inputs preloading). Time complexity appears in terms of number of levels of two-inputs AND (T_A) and XOR (T_X) gates, and 2-to-1 1-bit multiplexers (T_M). $p(x) = x^m + \sum_{i=1}^{\omega-2} x^{t_i} + 1$ is the field's irreducible polynomial, satisfying $\frac{m+1}{2} \geq t_{\omega-2}$ and $t_1 > 1$. Also, in the table, $T^{'} = \left(1 + \lceil \log_2 (\omega - 1) \rceil + \lceil \log_2 (m) \rceil\right) T_X$ and $T^{"} = \left(1 + \lceil \log_2 (\omega - 1) \rceil + \lceil \log_2 (m - 1) \rceil\right) T_X$ [75]. Moreover, d is the digit size, $k = \left\lceil \frac{m}{d} \right\rceil$, and for an integer x the function $\delta(x) = 0$ if $x \neq 1$.

Multiplier	FF	AND	XOR	2-to-1 1-bit MUX*	Propagation Delay	Parallel Loading		Serial Loading
						2-to-11-bit MUX	Latency	Latency
LSD DL-SIPO [57]	$2m + d - 1$	$dm + (2d - 1)(\omega - 1)$	$dm + (d - 1) +$ $(2d - 1)(\omega - 2)$	m	$T_A + \lceil \log_2 (d + 1) \rceil T_X$	m	$k + 1 - \delta(d)$	$2k + 1 - \delta(d)$
MSD DL-SIPO [80]	$2m + d$	$dm + (2d - 1)(\omega - 1)$	$dm + d(\omega - 2)$	0	$T_A + \lceil \log_2 (2d + 1) \rceil T_X$	m	$k + 1 - \delta(d)$	$2k + 1 - \delta(d)$
BL-PISO ($d = 1$) [75]	$3m + t_{\omega-2} - 1$	$2m - 1$	$(\omega - 1)(m - 1) +$ $\omega - 3 + \sum_{i=1}^{\omega-2} t_i$	0	$T_A + \Big(1 + \lceil \log_2 (\omega - 1) \rceil +$ $\lceil \log_2 (m) \rceil \Big) T_X$	$2m$	m	$2m$
PIPO [50]	$5m - 1$	$\frac{m^2+m}{2}$	$\frac{m^2+m}{2}$	$4m$	$T_A + \lceil \log_2 m \rceil T_X + 2T_M$	$2m$	$2t_{\omega-2} + 1$	$k + 2t_{\omega-2} + 1$
LSD DL-FSIPO (Figure 6.3a)	$3m - 2d$	$d(2m - d)$	$(2d + \omega - 2)m - (\omega - 2) -$ $d\left[\frac{(d-3)(\omega-2)}{2}\right] + 1$	0	$T_A + 2\Big(1 + \lceil \log_2 (d) \rceil$ $+ \lceil \log_2 (\omega - 1) \rceil \Big) T_X$	0	k	k
MSD DL-FSIPO (Figure 6.1a)	$3m - 2d$	$d(2m - d)$	$2dm + d\left[\frac{(d+3)(\omega-2)}{2} - d\right]$	0	$\lceil \log_2 (d + 1) \rceil T_X +$ $\max \Big\{ \lceil \log_2 (2d + 1) \rceil T_X,$ $T_A + (1 + \lceil \log_2 (d) \rceil) T_X \Big\}$	0	k	k

* These multiplexers are used in the multiplication logic which are different from the ones used for parallel preloading of inputs.

Table 6.3: Space and time complexities of the different digit-level $GF(2^m)$ PB multipliers.

While the DL-SIPO PB multipliers listed in this table offer best space complexities and propagation delays, one can see that, the proposed DL-FSIPO PB multipliers are advantageous for the case of serial inputs preloading since they offer lower latency for generating the m output bits. This feature of the proposed DL-FSIPO multipliers results in low-latency fast multiplication in resource constrained applications where the input data-path might have limited capacity for reading elements from large finite fields. In addition, and similar to the DL-SIPO PB multipliers in Table 6.3, the proposed MSD and LSD DL-FSIPO PB multipliers offer propagation delays that are independent of the dimension of $GF(2^m)$. Compared to the PIPO serial PB multiplier in Table 6.3, both of the proposed MSD and LSD DL-FSIPO PB multipliers show better space as well as time complexities.

Figures 6.6 and 6.7 plot the efficiency as a function of the digit size considering serial inputs loading and parallel inputs loading, respectively, for the different multipliers in Table 6.3 (except the BL-PISO), under the field $GF\left(2^{233}\right)$ recommended by NIST which is defined by an irreducible trinomial $p(x) = x^{233} + x^{t_1} + 1$, where $t_1 = 74$.

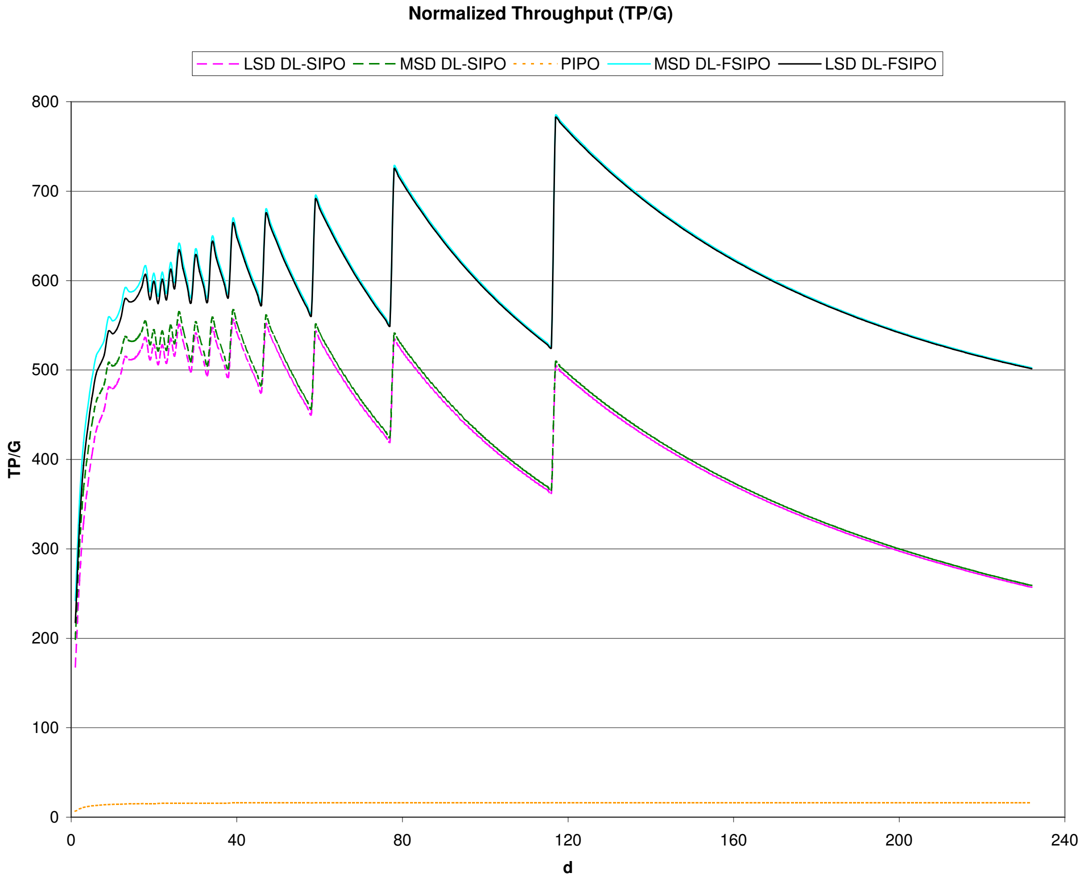


Figure 6.6: Normalized throughput as a function of the digit size for the serial inputs loading case.

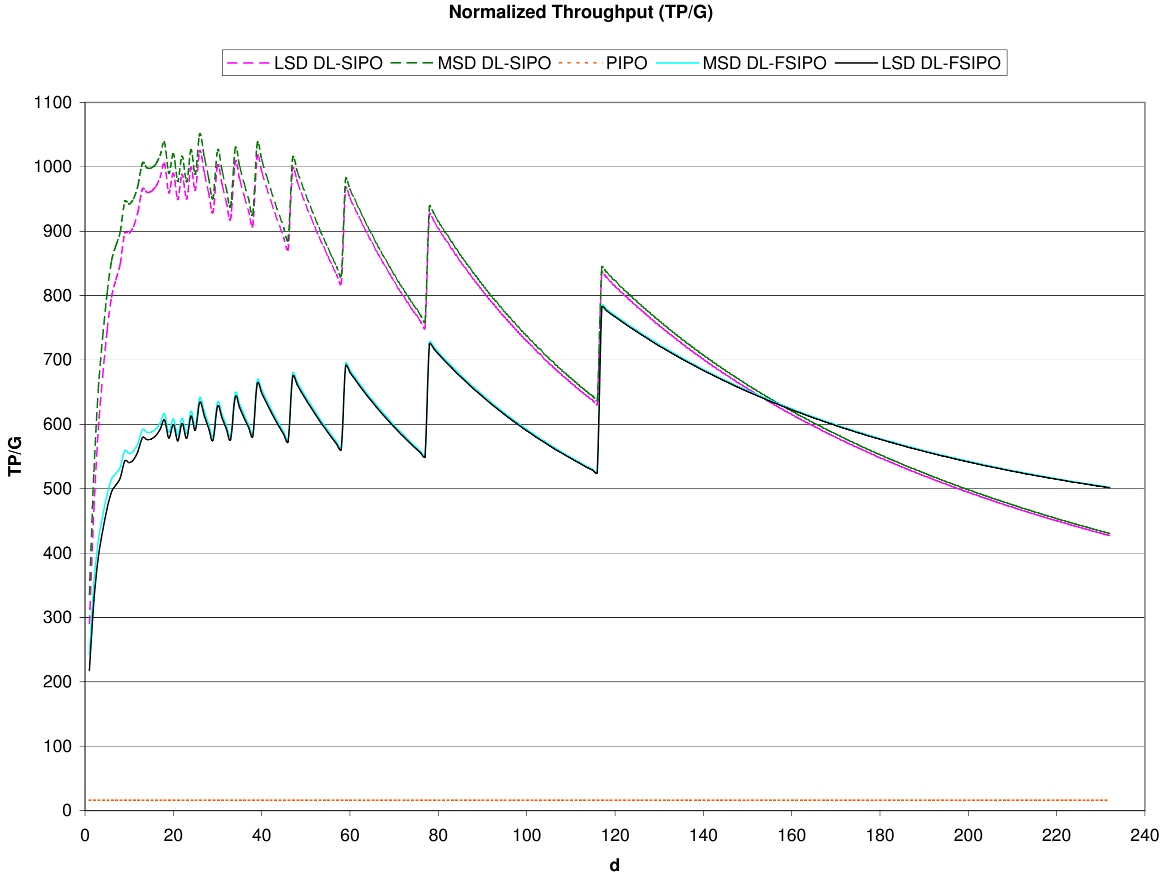


Figure 6.7: Normalized throughput as a function of the digit size for the parallel inputs loading case.

Here, efficiency denotes the normalized throughput, that is, throughput (computed at 1 GHz) per number of NAND gate equivalence (TP/G), measured in Kbps/Gate. The inclinations after each peak in these two plots are due to increasing the digit size while the latency (hence the throughput) is fixed. For instance, the efficiency peaks which start at $d = 117$ correspond to a computational latency of $k = \left\lceil \frac{233}{117} \right\rceil = 2$ clock cycles. Increasing d beyond the value of 117, say $d = 140$, increases the space complexities, however, the latency stays constant at $k = \left\lceil \frac{233}{140} \right\rceil = 2$.

From the two plots of Figures 6.6 and 6.7, one can see that, the proposed DL-FSIPO PB multipliers are advantageous for the case of serial inputs preloading since they offer better efficiencies, compared to the other multipliers (when running at the same clock speed). The DL-SIPO PB multipliers show higher efficiencies than the other multiplication schemes in the case of parallel inputs preloading. However, it is interesting to notice that the gap between the efficiencies of the proposed DL-FSIPO PB multipliers and those of the DL-SIPO PB multipliers decreases with increasing d in case of parallel preloading of inputs, as depicted in Figure 6.7. Although this is not a practical case, however, one can see from the chart that for values of $d \geq 160$, the efficiencies of the proposed DL-FSIPO PB multipliers beat those of the DL-SIPO PB multipliers. On the other side, for the case of serial inputs preloading, the efficiency gap increases in favor of the proposed DL-FSIPO PB multipliers. Considering the PIPO PB multiplier, it offers the lowest efficiency in both serial and parallel inputs preloading scenarios. Notice that, in the case of parallel inputs preloading, the efficiency of the PIPO PB multiplier is fixed since its latency depends on $t_{\omega-2}$ (second highest order amongst the orders of the terms forming the field defining polynomial), and not the digit size.

In Table 6.3, the PISO PB multiplier from [75] is a bit-level multiplier ($d = 1$). Hence, the digit size is set to $d = 1$ (one bit) for the different digit-level multipliers in this table, in order to conduct further comparisons. As a case study, the bit-level case of the field $GF\left(2^{233}\right)$ recommended by NIST which is defined by an irreducible trinomial $p(x) = x^{233} + x^{t_1} + 1$, where $t_1 = 74$, is investigated. Then, for this case, the resulting space and time complexities of the multipliers which are listed in Table 6.3 are reported in Table 6.4. Also, Table 6.5 estimates the corresponding space and time complexity readings based on the 65nm CMOS standard library's statistics. In this table, the total gate counts are estimated in terms of total NAND gate equivalence (GE) while MPD denotes the maximum propagation delay. Latency denotes the total number of clock cycles required to generate the 233-bits of output. TP is throughput (@ 1 GHz) and TP/G denotes throughput per total GE measured in Kbps/Gate. SIL and PIL denote "Serial Input Loading" and "Parallel Input Loading", respectively.

Multiplier	FF	AND	XOR	2-to-1 1-bit MUX*	Propagation Delay	Parallel Loading		Serial Loading
						2-to-11-bit MUX	Latency	Latency
LSB BL-SIPO [57]	466	235	234	233	$T_A + T_X$	233	233	466
MSB BL-SIPO [80]	467	235	234	0	$T_A + T_X$	233	233	466
BL-PISO [75]	772	465	538	0	$T_A + 10T_X$	466	233	466
PIPO [50]	1164	27261	27261	932	$T_A + 8T_X + 2T_M$	466	149	382
LSB BL-FSIPO (Figure 6.3a)	697	465	700	0	$T_A + 4T_X$	0	233	233
MSB BL-FSIPO (Figure 6.1a)	697	465	467	0	$3T_X$	0	233	233

* These multiplexers are used in the multiplication logic which are different from the ones used for parallel preloading of inputs.

Table 6.4: Space and time complexities for the NIST recommended field $GF\left(2^{233}\right)$ defined by the irreducible trinomial $x^{233} + x^{t_1} + 1$, where $t_1 = 74$ and the digit size is $d = 1$.

In the standard 65nm CMOS technology library, the NAND gate equivalences (GEs) for a two-inputs AND, two-inputs XOR, D-type FF, and a 2-to-1 1-bit Multiplexer, when reported based on synthesis results using the Synopsys Design Vision tool [4], are 1.25, 2, 3.75, and 2, respectively. In addition, and based on synthesis with the same tool using the same technology library, the maximum propagation delays (MPD) for a two-inputs AND, two-inputs XOR, and

Multiplier	MPD	GE		Latency		TP/G @ 1 GHz	
	ns	PIL	SIL	PIL	SIL	PIL	SIL
LSB BL-SIPO [57]	0.07	3441	2975	233	466	291	168
MSB BL-SIPO [80]	0.07	2979	2513	233	466	336	199
BL-PISO [75]	0.43	5484	4552	233	466	182	110
PIPO [50]	0.41	95759	94827	149	382	16	6
LSB BL-FSIPO (Figure 6.3a)	0.19	4595	4595	233	233	218	218
MSB BL-FSIPO (Figure 6.1a)	0.12	4129	4129	233	233	242	242

Table 6.5: Space and time complexity estimates for the multipliers which are listed in Table 6.4 based on on the standard 65nm CMOS library measures.

2-to-1 1-bit multiplexer are 0.03ns, 0.04ns, and 0.03ns respectively.

From Table 6.5, one can see that the listed PIPO serial PB multiplier offers the best latency in case of parallel preloading of its inputs. However, it has lowest efficiency (i.e. normalized throughput in terms of throughput per NAND gate equivalence measured at 1 GHz) in both parallel and serial preloading scenarios, compared to all the other listed multiplication schemes. This is mainly due to the relatively large space complexity of this PIPO serial PB multiplier.

It is noted that the BL-SIPO PB multipliers which are listed in Table 6.5 offer the best space complexity and highest operating frequency. In addition, the BL-SIPO PB multipliers in this table show the best efficiency, in case of parallel preloading of inputs.

However, the proposed MSB and LSB BL-FSIPO PB multipliers offer lower latencies, compared to the BL-SIPO, BL-PISO, and PIPO, in case of serial inputs loading. In this case, as a result of the low latencies, the proposed MSB and LSB BL-FSIPO PB multipliers show the best efficiency. In comparison to the BL-PISO and BL-PIPO PB multipliers, which are listed in Table 6.5, the proposed MSB and LSB BL-FSIPO PB multipliers are advantageous in terms of space complexity[1], operating frequency, and efficiency, as well as in terms of latency in case of serial loading of inputs. Furthermore, Table 6.5 show that the proposed MSB BL-FSIPO PB multiplier is superior to the proposed LSB BL-FSIPO PB multiplier in terms of space complexity, propagation delay, and hardware efficiency.

It is also worth noting that, in the case of parallel preloading of inputs, the BL-PISO PB multiplier generates its first output bit with a latency of 1 clock cycle, while the proposed BL-FSIPO, as well as the BL-SIPO PB multipliers, require 233 clock cycles after which all the output bits are generated in parallel. For the same case of parallel preloading of the inputs, the PIPO PB multiplier, which is listed in Table 6.5, requires 149 clock cycles to generate all the 233 output bits, in parallel.

---

[1] the LSB BL-FSIPO PB shows almost similar space complexity as the PISO for the case of serial loading of inputs

In cases where the multiplication results need to be communicated to other modules of the underlying system, one can convert the proposed DL-FSIPO PB multipliers into serial-in-serial-out schemes, if the output is transmitted using the same limited capacity data-path of inputs. This is done by running the proposed DL-FSIPO PB multipliers an additional $\left\lceil \frac{k}{2} \right\rceil$ clock cycles through which the inputs are set to zeros. During each one of the additional clock cycles, the proposed DL-FSIPO PB multipliers produce two digits of the multiplication result for transmission over the output data-path. Therefore, the proposed DL-FSIPO PB multipliers are advantageous over the serial-serial multipliers presented in [46, 14], in the sense that they fully utilize the output data-path by requiring only $2k - \left\lceil \frac{k}{2} \right\rceil$ clock cycles compared to $2k$ clock cycles required by the multipliers in [46, 14] which use only half the output data-path capacity. In addition, in case only one output digit is required to be transmitted per a clock cycle, one can accomplishes this by using a dedicated parallel-in-serial-out output register with the proposed DL-FSIPO PB multipliers. This scheme accomplishes n consecutive multiplications using only $(n + 1)\, k$ clock cycles, and hence, it is favoured over the serial-serial multiplication schemes in [46, 14] which require $2nk$ clock cycles for the same scenario.

The following section, concludes this chapter.

6.4 Conclusion

This chapter introduced two new digit-level multiplication schemes for the elements of $GF\,(2^m)$, based on the PB representation. The proposed formulations for the digit-level PB multiplications are based on recursive constructions of the field elements, which constructs an element digit-by-digit, one digit per a clock cycle, starting from either the most or the last significant digit. Based on these new formulations, and to the best of the author knowledge, the first architectures for digit-level fully-serial-in-parallel-out (DL-FSIPO) multiplier have been proposed for dedicated PB. The proposed MSD and LSD DL-FSIPO PB multipliers do not require any preloading of the inputs and, therefore, they are advantageous for achieving high throughput in applications where the parallel preloading of the inputs is not possible (if the input data-path size is limited, which is possible in resource constrained applications). For this specific case of serial preloading of the inputs, it has been shown, based on the provided theoretical analysis, that the proposed MSD and LSD DL-FSIPO PB multipliers offer the highest throughput and normalized throughput, when compared to other digit-level serial PB multiplication schemes.

Chapter 7

Summary and Future Work

This chapter summarizes the contributions of this work and presents some future goals.

7.1 Summary of Contributions

This thesis introduced efficient hardware designs of the WG stream ciphers in Chapters 3 and 4. The presented designs in Chapter 3 are for the multiple output bit MOWG$(29, 11, 17)$ and single output bit WG$(29, 11)$ based on the ONB-II representation of the $GF\left(2^{29}\right)$ elements. The hardware complexity of the MOWG$(29, 11, 17)$ has been reduced by one field multiplier through signal reuse techniques, while its time complexity has been slightly enhanced by removing some inverters from the critical path. On the other hand, the space complexity of the WG$(29, 11)$ has been significantly reduced to only five multipliers in its transform through the utilization of new trace properties. The new trace property generates the trace of the multiplication of two field elements represented in the ONB-II without performing the multiplication. The conducted ASIC and FPGA implementations showed superior performance of the proposed WG$(29, 11)$ compared to previous counterparts.

In Chapter 4, polynomial basis representation of the field elements has been considered for the first time for implementing WG stream ciphers. Nine new designs have been introduced. Three out of which are for the class of WG$(29, 11)$ including a standard, a serialized, and a pipelined versions. The other six designs are for the class of WG-16 including a standard, a serialized, and a pipelined versions, each implemented by a traditional PB multiplier and Karatsuba multiplier. The space complexity of these two classes of the WG cipher has been significantly reduced through using a new trace property for the PB. Similar to the trace method introduced in Chapter 3, the new trace property of this chapter generates the trace of the multiplication of two field elements represented in the PB without performing the multiplication. The different designs have been demonstrated by ASIC implementations and have shown a promising performance compared to the previous counterparts. In particular, this chapter showed that the proposed WG-16 designs comply with the bit rate requirements of the 4G mobile network domain, while offering a variety of optimization options.

In addition, new architectures for the digit-level multiplication in the GNB and PB representations of $GF(2^m)$ elements have been proposed in Chapters 5 and 6. In Chapters 5 and 6, new DL-FSIPO $GF(2^m)$ multiplication schemes have been proposed for both of the GNB and PB representations, respectively. These new architectures are shown to be advantageous for increasing the throughput in applications with limited data-path capacities. Both MSD as well as LSD variants have been constructed for all proposed DL-FSIPO multipliers.

In Chapter 5, an optimized MSD DL-PISO GNB multiplier has also been presented. The proposed MSD DL-FSIPO and DL-PISO GNB multipliers in Chapter 5 have been interleaved in order to construct new architectures for an MSD DL-SIPO GNB Hybrid-double and a DL-PIPO GNB Hybrid-triple multiplications. The latter two hybrid multiplication schemes conduct multiplication of three and four field elements, respectively, using the same latency required to multiply only two elements. Based on the proposed Hybrid-triple GNB multiplier, a new digit-level eight-ary exponentiation scheme has been presented in Chapter 5. This new exponentiation uses almost the same latency of existing eight-ary designs, however, it does not require pre-computations or storage of intermediate values.

The following section presents some future work for this thesis.

7.2 Future Work

In the future, the following projects can be considered as a continuation for this thesis:

- Generalized hybrid-n-ary FSIPO multipliers.
- ASIC and FPGA realizations of the proposed DL-FSIPO and the digit-level hybrid multipliers, optimized at the gate / transistor level.
- Implementations for fast field inversion designs using the new hybrid multipliers based on the recently published work about generalized k-chains [51].
- Ultra lightweight hardware designs for the WG-16 stream cipher suitable for RFIDs, based on the new hybrid-triple digit-level multipliers.
- Concurrent error control in the different presented designs.

Bibliography

[1] 3GPP Technical Specification Groups. `http://www.3gpp.org/Specification-Groups`.

[2] Xilinx. `http://www.xilinx.com/`.

[3] The Sage Notebook. `http://www.sagenb.org/`.

[4] Synopsys. `http://www.synopsys.com/`.

[5] IEEE Standard Specifications for Public-Key Cryptography. *IEEE Std 1363-2000*, page i, 2000.

[6] eSTREAM - The ECRYPT Stream Cipher Project, 2005.

[7] 3rd Generation Partnership Project; Long Term Evaluation Release 10 and Beyond (LTE-Advanced); Proposed to ITU at 3GPP TSG RAN Meeting, 2009.

[8] Adopted Bluetooth Core Specifications, Core Version 4.0. Bluetooth Special Interest Group, June 2010.

[9] CLP-41: SNOW 3G Flow through Core. Elliptic Technologies, 2011. `http://www.elliptictech.com/products-clp-41.php`.

[10] ZUC Key Stream Generator. Elliptic Technologies, 2011. `http://www.elliptictech.com/pdf/CLP-410ZUCKeyStreamGenerator.pdf`.

[11] 3GPP TS 33.401 v11.0.1. 3rd Generation Partnership Project; Technical Specification Group Services and Systems Aspects; 3GPP System Architecture Evolution (SAE): Security Architecture, June 2011 (Release 11).

[12] Digital Signature Standard (DSS). Federal Information Processing Standards (FIPS), July 2013.

[13] Gordon B. Agnew, Ronald C. Mullin, I. M. Onyszchuk, and Scott A. Vanstone. An Implementation for a Fast Public-Key Cryptosystem. *J. Cryptology*, 3:63–79, 1991.

[14] Abdulaziz Al-Khoraidly and Mohammad K. Ibrahim. Finite field serial-serial multiplication/reduction structure and method, US7519644 B2, Apr 2009.

[15] David W. Ash, Ian F. Blake, and Scott A. Vanstone. Low Complexity Normal Bases. *Discrete Applied Math.*, 25(3):191 – 210, 1989.

[16] R. Azarderakhsh and A. Reyhani-Masoleh. Low-Complexity Multiplier Architectures for Single and Hybrid-Double Multiplications in Gaussian Normal Bases. *IEEE Trans. Comput.*, 62(4):744–757, April 2013.

[17] Reza Azarderakhsh and Arash Reyhani-Masoleh. A Modified Low Complexity Digit-Level Gaussian Normal Basis Multiplier. In M.Anwar Hasan and Tor Helleseth, editors, *Arithmetic of Finite Fields*, volume 6087 of *Lecture Notes in Computer Science*, pages 25–40. Springer Berlin Heidelberg, 2010.

[18] Elaine Barker, William Barker, William Burr, William Polk, and Miles Smid. Nist Special Publication 800-57. *NIST Special Publication*, 800(57):1–142, 2007.

[19] Thomas C. Bartee and David I. Schneider. Computation With Finite Fields. *Information and Control*, 6(2):79 – 98, 1963.

[20] T. Beth and D. Gollman. Algorithm Engineering for Public Key Algorithms. *IEEE J. Sel. Areas Commun.*, 7(4):458–466, 1989.

[21] Alex Biryukov, Deike Priemuth-Schmid, and Bin Zhang. Differential Resynchronization Attacks on Reduced Round SNOW $3G^{\oplus}$. In MohammadS. Obaidat, GeorgeA. Tsihrintzis, and Joaquim Filipe, editors, *e-Business and Telecommunications*, volume 222 of *Communications in Computer and Information Science*, pages 147–157. Springer Berlin Heidelberg, 2012.

[22] M. Cenk, M.A Hasan, and C. Negre. Efficient Subquadratic Space Complexity Binary Polynomial Multipliers Based on Block recombination. *IEEE Trans. Comput.*, 63(9):2273–2287, September 2014.

[23] L. Chen, J. Franklin, and A. Regenscheid. Guidelines on Hardware-Rooted Security in Mobile Devices (Draft). In *Special Publication 800-164*. National Institute of Standards and Technology, October 2012.

[24] Lidong Chen and Guang Gong. *Communication System Security*. Chapman and Hall - CRC Press, 2012.

[25] Yanni Chen and Keshab K. Parhi. Small Area Parallel Chien Search Architectures for Long BCH Codes. *IEEE Trans. Very Large Scale Integr. Syst.*, 12(5):545–549, May 2004.

[26] Chao Cheng and K.K. Parhi. High-Speed Parallel CRC Implementation Based on Unfolding, Pipelining, and Retiming. *IEEE Trans. Circuits and Systems II*, 53(10):1017–1021, 2006.

[27] A. Cilardo. Fast Parallel GF(2^m) Polynomial Multiplication for All Degrees. *IEEE Trans. Comput.*, 62(5):929–943, May 2013.

[28] J.P. Deschamps, J.L. Imaña, and G.D. Sutter. *Hardware Implementation of Finite-Field Arithmetic*. McGraw-Hill Education, 2009.

[29] W. Diffie and M.E. Hellman. New Directions in Cryptography. *IEEE Trans. Inf. Theory*, 22(6):644–654, November 1976.

[30] V. Dimitrov and K. Jarvinen. Another Look at Inversions Over Binary Fields. In *2013 21st IEEE Symposium on Computer Arithmetic (ARITH)*, pages 211–218, April 2013.

[31] H. El-Razouk, A Reyhani-Masoleh, and G. Gong. New Implementations of the WG Stream Cipher. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, 22(9):1865–1878, September 2014.

[32] H. El-Razouk, A. Reyhani-Masoleh, and G. Gong. New Hardware Implementations of WG(29,11) and WG-16 Stream Ciphers Using Polynomial Basis. *IEEE Trans. Comput.*, to appear.

[33] Serdar S. Erdem, Tugrul Yanik, and Çetin K. Koç. Polynomial Basis Multiplication over GF(2^m). *Acta Applicandae Mathematica*, 93(1-3):33–55, September 2006.

[34] X. Fan and G. Gong. Specification of the Stream Cipher WG-16 Based Confidentiality and Integrity Algorithms. Technical Report CACR 2013-06, University of Waterloo, Waterloo, ON, Canada, 2013.

[35] X. Fan, N. Zidaric, M. Aagaard, and G. Gong. Efficient Hardware Implementation of the Stream Cipher WG-16 with Composite Field Arithmetic. Technical Report CACR 2013-23, University of Waterloo, Waterloo, ON, Canada, 2013.

[36] G.-L. Feng. A VLSI Architecture for Fast Inversion in GF(2^m). *IEEE Trans. Comput.*, 38(10):1383–1386, 1989.

[37] L. Gao and G.E. Sobelman. Improved VLSI Designs for Multiplication and Inversion in GF(2M) Over Normal Bases. In *ASIC/SOC Conference, 2000. Proceedings. 13th Annual IEEE International*, pages 97–101, 2000.

[38] Willi Geiselmann and Dieter Gollmann. Symmetry and Duality in Normal Basis Multiplication. In Teo Mora, editor, *Applied Algebra, Algebraic Algorithms and Error-Correcting Codes*, volume 357 of *Lecture Notes in Computer Science*, pages 230–238. Springer Berlin Heidelberg, 1989.

[39] Guang Gong and Yassir Nawaz. The WG Stream Cipher. eSTREAM, ECRYPT Stream Cipher Project, Report 2005/033, 2005.

[40] Guang Gong and A.M. Youssef. Cryptographic Properties of the Welch-Gong Transformation Sequence Generators. *IEEE Trans. Inf. Theory*, 48(11):2837 – 2846, Nov. 2002.

[41] T. Good and M. Benaissa. Hardware Results for Selected Stream Cipher Candidates. In *Workshop Record of the State of The Art of Stream Ciphers 2007 (SASC 2007)*, pages 191–204, 2007.

[42] Daniel M. Gordon. A Survey of Fast Exponentiation Methods. *Journal of Algorithms*, 27(1):129–146, April 1998.

[43] S.S. Gupta, A. Chattopadhyay, K. Sinha, S. Maitra, and B.P. Sinha. High-Performance Hardware Implementation for RC4 Stream Cipher. *IEEE Trans. Comput.*, 62(4):730–743, 2013.

[44] A. Halbutogullari and C.K. Koc. Mastrovito Multiplier for General Irreducible Polynomials. *IEEE Trans. Comput.*, 49(5):503 –518, May 2000.

[45] A. Hariri and A. Reyhani-Masoleh. Digit-Level Semi-Systolic and Systolic Structures for the Shifted Polynomial Basis Multiplication Over Binary Extension Fields. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, 19(11):2125 –2129, nov. 2011.

[46] M.A. Hasan and V.K. Bhargava. Division and Bit-Serial Multiplication over GF(q^m). *Computers and Digital Techniques, IEE Proceedings E*, 139(3):230–236, May 1992.

[47] M.A. Hasan, M.Z. Wang, and V.K. Bhargava. A Modified Massey-Omura Parallel Multiplier for a Class of Finite Fields. *IEEE Trans. Comput.*, 42(10):1278 –1280, Oct. 1993.

[48] Jenn-Shyong Horng, I.-Chang Jou, and Chiou-Yng Lee. Low-Complexity Multiplexer-Based Normal Basis Multiplier Over GF(2^m). *Journal of Zhejiang University SCIENCE A*, 10(6):834–842, May 2009.

[49] Junxian Huang, Feng Qian, Alexandre Gerber, Z. Morley Mao, Subhabrata Sen, and Oliver Spatscheck. A Close Examination of Performance and Power Characteristics of 4G LTE Networks. In *Proceedings of the 10th international conference on Mobile systems, applications, and services*, MobiSys '12, pages 225–238, New York, NY, USA, 2012. ACM.

[50] J.L. Imaña. Low Latency GF(2^m) Polynomial Basis Multiplier. *IEEE Trans. Circuits Syst. I, Reg. Papers*, 58(5):935–946, May 2011.

[51] K. Jarvinen, V. Dimitrov, and R. Azarderakhsh. A Generalization of Addition Chains and Fast Inversions in Binary Fields. *IEEE Transactions on Computers*, to appear.

[52] D. Johnson, A. Menezes, and S. Vanstone. The Elliptic Curve Digital Signature Algorithm (ECDSA). *Int'l J. Information Security*, 1(1):36–63, 2001.

[53] Anatolii Karatsuba and Yuri Ofman. Multiplication of Multidigit Numbers on Automata. *Soviet Physics-Doklady*, 7:595–596, 1963.

[54] Paris Kitsos, George Selimis, and Odysseas Koufopavlou. High Performance ASIC Implementation of the SNOW 3G Stream Cipher. In *IFIP/IEEE VLSISOC 2008 - International Conference on Very Large Scale Integration (VLSI SOC)*, Rhodes Island, Greece, Oct. 13-15 2008.

[55] C.K. Koc and B. Sunar. Low-Complexity Bit-Parallel Canonical and Normal Basis Multipliers for a Class of Finite Fields. *IEEE Trans. Comput.*, 47(3):353–356, 1998.

[56] E. Krengel. Fast WG Stream Cipher. In *IEEE Region 8 Int. Conf. on Computational Technologies in Elect. and Electron. Eng., 2008. SIBIRCON 2008.*, pages 31 –35, Jul. 2008.

[57] S. Kumar, T. Wollinger, and C. Paar. Optimum Digit Serial GF(2^m) Multipliers for Curve-Based Cryptography. *IEEE Transactions on Computers*, 55(10):1306–1311, October 2006.

[58] C. Lam, M. Aagaard, and G. Gong. Hardware Implementations of Multi-output Welch-Gong Ciphers. Technical Report CACR 2011-01, University of Waterloo, Waterloo, ON, Canada, 2009.

[59] Rudolf Lidl and Harald Niederreiter. *Introduction to Finite Fields and their Applications*. Cambridge University Press, New York, NY, USA, 1986.

[60] Yiyuan Luo, Qi Chai, Guang Gong, and Xuejia Lai. A Lightweight Stream Cipher WG-7 for RFID Encryption and Authentication. In *Global Telecommunications Conference (GLOBECOM 2010), 2010 IEEE*, pages 1 –6, Dec. 2010.

[61] James L. Massey and Jimmy K. Omura. Computational Method and Apparatus for Finite Field Arithmetic, May 1986.

[62] Edoardo D. Mastrovito. VLSI Designs for Multiplication Over Finite Fields GF(2^m). In Teo Mora, editor, *Applied Algebra, Algebraic Algorithms and Error-Correcting Codes*, volume 357 of *Lecture Notes in Computer Science*, pages 297–309. Springer Berlin Heidelberg, 1989.

[63] Edoardo D. Mastrovito. VLSI Designs for Multiplication over Finite Fields GF(2m). In *Proceedings of the 6th International Conference, on Applied Algebra, Algebraic Algorithms and Error-Correcting Codes*, AAECC-6, pages 297–309, London, UK, UK, 1989. Springer-Verlag.

[64] A. Mirzaei, M. Dakhilalian, and M. Modarres-Hashemi. An Improved Attack on WG Stream Cipher. *IJSNS International Journal of Computer Science and Network Security*, 10(4):45–52, apr. 2010.

[65] R. C. Mullin, I. M. Onyszchuk, S. A. Vanstone, and R. M. Wilson. Optimal Normal Bases in $GF(p^n)$. *Discrete Applied Math.*, 22(2):149–161, Feb. 1989.

[66] S.H. Namin, Huapeng Wu, and M. Ahmadi. Power Efficiency of Digit Level Polynomial Basis Finite Field Multipliers in GF(2^{283}). In *2012 19th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, pages 897–900, December 2012.

[67] Yassir Nawaz. *Design of Stream Ciphers and Cryptographic Properties of Nonlinear Functions*. PhD thesis, University of Waterloo, 2007.

[68] Yassir Nawaz and Guang Gong. WG: A Family of Stream Ciphers with Designed Randomness Properties. *Inf. Sci.*, 178(7):1903 – 1916, 2008.

[69] C. Paar. Optimized Arithmetic for Reed-Solomon Encoders. In *, 1997 IEEE International Symposium on Information Theory. 1997. Proceedings*, pages 250–, June 1997.

[70] A. Reyhani-Masoleh. Efficient Algorithms and Architectures for Field Multiplication Using Gaussian Normal Bases. *IEEE Trans. Comput.*, 55(1):34–47, Jan 2006.

[71] A. Reyhani-Masoleh and M.A. Hasan. A New Construction of Massey-Omura Parallel Multiplier Over $GF(2^m)$. *IEEE Trans. Comput.*, 51(5):511 –520, May. 2002.

[72] A. Reyhani-Masoleh and M.A. Hasan. Low Complexity Bit Parallel Architectures for Polynomial Basis Multiplication Over GF(2^m). *IEEE Trans. Comput.*, 53(8):945 – 959, Aug. 2004.

[73] A. Reyhani-Masoleh and M.A. Hasan. Low Complexity Word-Level Sequential Normal Basis Multipliers. *IEEE Trans. Comput.*, 54(2):98–110, 2005.

[74] A. Reyhani-Masoleh and M.A. Hasan. Low Complexity Word-Level Sequential Normal Basis Multipliers. *IEEE Trans. Comput.*, 54(2):98–110, 2005.

[75] Arash Reyhani-Masoleh. A New Bit-Serial Architecture for Field Multiplication Using Polynomial Bases. In Elisabeth Oswald and Pankaj Rohatgi, editors, *Cryptographic Hardware and Embedded Systems - CHES 2008*, number 5154 in Lecture Notes in Computer Science, pages 300–314. Springer Berlin Heidelberg, Jan 2008.

[76] Arash Reyhani-Masoleh and M. Anwar Hasan. Efficient Digit-Serial Normal Basis Multipliers Over Binary Extension Fields. *ACM Trans. Embed. Comput. Syst.*, 3(3):575–592, August 2004.

[77] Sondre Ronjom and Tor Helleseth. Attacking the Filter Generator Over $GF(2^m)$. eSTREAM, ECRYPT Stream Cipher Project, Report 2007/011, 2007.

[78] P.A Scott, S.E. Tavares, and L.E. Peppard. A Fast VLSI Multiplier for GF(2^m). *IEEE J. Sel. Areas Commun.*, 4(1):62–66, January 1986.

[79] George N. Selimis, Apostolos P. Fournaris, Harris E. Michail, and Odysseas Koufopavlou. Improved Throughput Bit-Serial Multiplier for GF(2^m) Fields. *Integration, the VLSI Journal*, 42(2):217 – 226, 2009.

[80] Leilei Song and Keshab K. Parhi. Low-Energy Digit-Serial/Parallel Finite Field Multipliers. *Journal of VLSI signal processing systems for signal, image and video technology*, 19(2):149–166, July 1998.

[81] Leilei Song and K.K. Parhi. Efficient Finite Field Serial/Parallel Multiplication. In *Proceedings of International Conference on Application Specific Systems, Architectures and Processors, 1996. ASAP 96*, pages 72–82, August 1996.

[82] W. Stallings. *Cryptography and Network Security: Principles and Practice*. Prentice Hall, 2011.

[83] D. Stinson. Some Observations on Parallel Algorithms for Fast Exponentiation in $GF(2^n)$. *SIAM J. Comput.*, 19(4):711–717, August 1990.

[84] B. Sunar and C.K. Koc. Mastrovito Multiplier for All Trinomials. *IEEE Trans. Comput.*, 48(5):522 –527, May 1999.

[85] C.C. Wang and D. Pei. A VLSI Design for Computing Exponentiations in $GF(2^m)$ and its Application to Generate Pseudorandom Number Sequences. *IEEE Trans. Comput.*, 39(2):258–262, February 1990.

[86] C.C. Wang, T.K. Troung, H.M. Shao, L.J. Deutsch, J. Omura, and Irving S. Reed. VLSI Architectures for Computing Multiplications and Inverses in GF(2m). *IEEE Trans. Comput.*, C-34(8):709–717, 1985.

[87] Hongjun Wu, Tao Huang, PhuongHa Nguyen, Huaxiong Wang, and San Ling. Differential Attacks Against Stream Cipher ZUC. In Xiaoyun Wang and Kazue Sako, editors, *Advances in Cryptology - ASIACRYPT 2012*, volume 7658 of *Lecture Notes in Computer Science*, pages 262–277. Springer Berlin Heidelberg, 2012.

[88] Hongjun Wu and Bart Preneel. Resynchronization Attacks on WG and LEX. In Matthew Robshaw, editor, *Fast Software Encryption*, volume 4047 of *Lecture Notes in Computer Science*, pages 422–432. Springer-Verlag, 2006.

[89] Huapeng Wu. Bit-Parallel Finite Field Multiplier and Squarer Using Polynomial Basis. *IEEE Trans. Comput.*, 51(7):750 –758, July 2002.

[90] Teng Wu and Guang Gong. The Weakness of Integrity Protection for LTE. In *the Proceedings of Sixth ACM Conference on Security and Privacy in Wireless and Mobile Networks, WiSec13*, pages 79–88. Also, appeared as Technical Report, CACR 2013–03, 2013, University of Waterloo, Canada., Budapest, Hungary, Apr. 17-19 2013.

Curriculum Vitae

Name:	Hayssam El-Razouk
Post-Secondary Education and Degrees:	University of Western Ontario London, ON, Canada 2011 - 2015 Ph.D.
	University of Western Ontario London, ON, Canada 2004 - 2006 M.E.Sc.
	Beirut Arab University Beirut, Lebanon 1997 - 2002 B.E.
Honours and Awards:	NSERC CGS D (Canada) 2012-2015.
	Jamal Abdul Nasir (Lebanon) 1999-2002.
Related Work Experience:	Teaching / Research Assistant University of Western Ontario 2004 - 2006 and 2011 - 2015.
	Software Engineer RedIron Technologies (Canada) 2006 - 2011.