11-18-2011 12:00 AM

High Speed and Low-Complexity Hardware Architectures for Elliptic Curve-Based Crypto-Processors

Reza Azarderakhsh, *University of Western Ontario*

Supervisor: Dr. 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

High Speed and Low-Complexity Hardware Architectures for Elliptic Curve-Based Crypto-Processors

(Spine Title: Hardware Architectures for Elliptic Curve Cryptography)

(Thesis Format: Monograph)

by

Reza Azarderakhsh

Faculty of Engineering
Department of Electrical and Computer Engineering

Submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy

School of Graduate and Postdoctoral Studies
The University of Western Ontario
London, Ontario, Canada
November, 2011

Certificate of Examination

The University of Western Ontario
School of Graduate and Postdoctoral Studies

Supervisor

Dr. Arash Reyhani-Masoleh

Examining Board

Dr. Ali Miri

Dr. Abdallah Shami

Dr. Anestis Dounavis

Dr. Éric Schost

The thesis by

Reza Azarderakhsh

entitled:

High Speed and Low-Complexity Hardware Architectures for Elliptic Curve-Based Crypto-Processors

is accepted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy

November 18 2011

Date

Chair of The Thesis Examining Board

Abstract

The elliptic curve cryptography (ECC) has been identified as an efficient scheme for public-key cryptography. This thesis studies efficient implementation of ECC crypto-processors on hardware platforms in a bottom-up approach. We first study efficient and low-complexity architectures for finite field multiplications over Gaussian normal basis (GNB). We propose three new low-complexity digit-level architectures for finite field multiplication. Architectures are modified in order to make them more suitable for hardware implementations specially focusing on reducing the area usage. Then, for the first time, we propose a hybrid digit-level multiplier architecture which performs two multiplications together (double-multiplication) with the same number of clock cycles required as the one for one multiplication. We propose a new hardware architecture for point multiplication on newly introduced binary Edwards and generalized Hessian curves. We investigate higher level parallelization and lower level scheduling for point multiplication on these curves. Also, we propose a highly parallel architecture for point multiplication on Koblitz curves by modifying the addition formulation. Several FPGA implementations exploiting these modifications are presented in this thesis. We employed the proposed hybrid multiplier architecture to reduce the latency of point multiplication in ECC crypto-processors as well as the double-exponentiation. This scheme is the first known method to increase the speed of point multiplication whenever parallelization fails due to the data dependencies amongst lower level arithmetic computations. Our comparison results show that our proposed multiplier architectures outperform the counterparts available in the literature. Furthermore, fast computation of point multiplication on different binary elliptic curves is achieved.

Keywords: Elliptic curve cryptography, Gaussian normal basis, digit-level finite field multiplication, hybrid multiplier, point multiplication, FPGA, ASIC.

Dedication

To Olfat and Ava.
To my parents.

Acknowledgements

All praise is due to God. I would like to express my appreciation and gratitude to Prof. Arash Reyhani-Masoleh for supervising my research during my Ph.D. studies at the University of Western Ontario. I also would like to thank my colleagues, Dr. Arash Hariri, Dr. Mehran Mozaffari-Kermani, and Christopher Kennedy for their suggestions, comments, and sharing their knowledge with me. I would like to thank my lab-mates Mohsen Bahramali, Adam Aksoy, Ebrahim Hasan, and S. Behdad Hosseini.

I would like to thanks my committee members, Dr. Ali Miri, Dr. Eric Schost, Dr. Adallah Shami, and Dr. Anestis Dounavis, for taking their time and reading this thesis and providing constructive comments.

Last but not least, I am grateful to my wife, for her love, kind support and having patience during working on this research. I would like to thank my parents, my sisters and my brother for their wisdom and moral supports. Special thanks go to my brother, Alireza Azarderakhsh, for his dedication and unflagging support.

Contents

Certificate of Examination ii

Abstract iii

Dedication iv

Acknowledgements v

Contents vi

List of Tables xi

List of Figures xiii

List of Algorithms xvi

Nomenclature xvii

1 Introduction 1

1.1 Problem Statement and Motivation 2

1.2 Objectives of the Thesis 2

1.3 Thesis Outline 3

2 Preliminaries and Literature Review 4

2.1 Finite Fields 4

2.2 Binary Fields Arithmetic 6

2.2.1 Polynomial Basis 6

2.2.2 Normal Basis 7

2.2.3 Finite Field Multiplication 7

2.2.3.1 Multiplication Using Normal Basis 8

2.2.3.2 Multiplication Using Gaussian Normal Basis 9

2.2.3.3 Inversion 10
2.2.3.4 Trace and Quadratic Equation Solution 11
2.2.4 Multiplier Architectures 11
2.2.4.1 Bit-Level NB Multiplication 12
2.2.4.2 An Example 16
2.2.4.3 Digit-level GNB multiplication 16
2.2.4.4 Digit-level PISO GNB multiplier 16
2.2.4.5 Digit-level PIPO GNB Multiplier 18
2.3 Elliptic Curve Cryptography 18
2.3.1 Elliptic Curve Arithmetic 20
2.3.2 Inversion free Coordinates 21
2.3.2.1 Standard Projective Coordinates 22
2.3.2.2 Lopez-Dahap Projective Coordinates 22
2.3.2.3 Jacobian Projective Coordinates 22
2.3.3 Point Multiplication 23
2.3.3.1 Double-And-Add Point Multiplication 23
2.3.3.2 Montgomery Point Multiplication 24

3 Low-Complexity Architectures for Digit-level and Bit-parallel GNB Multipliers over $GF(2^m)$ 26
3.1 An Improved Architecture for Digit-level PIPO GNB Multiplier . . . 27
3.1.1 Complexities 30
3.1.2 An Example over $GF(2^7)$ 31
3.1.3 Simulation Results for the DL-PIPO GNB Multiplier over $GF(2^{163})$ and $GF(2^{283})$ 33
3.2 New Architecture for Digit-Level SIPO GNB Multiplier 34
3.2.1 Formulation 35
3.2.2 New Architecture 36
3.2.2.1 Complexities 39
3.2.2.2 Complexity Reduction 40
3.2.3 An Illustrative Example 40
3.2.4 Simulations 43
3.3 New Architecture for Digit-Level PISO GNB multiplier 45
3.3.1 Low-Complexity Digit-Level PISO GNB Multiplier 45
3.3.1.1 Improved Architecture 45
3.3.1.2 Complexities 46

3.3.2 Complexity Comparison 46
3.4 An Extension to Bit-Parallel GNB Multiplier 48
3.4.1 Comparison 49
3.5 FPGA and ASIC Implementations 52
3.6 Conclusion 53

4 Efficient FPGA Implementation of Point Multiplication over Binary Edwards and Generalized Hessian Curves Using Gaussian Normal Basis 55
4.1 Preliminaries 56
4.1.1 Arithmetic over Binary Edwards and Generalized Hessian Curves 56
4.1.2 Point Addition and Doubling Using Differential Formulations in w-coordinates 58
4.2 Point Multiplication on Binary Edwards and Generalized Hessian Curves 60
4.2.1 Point Multiplication 60
4.2.2 Parallelism in Point Multiplication Algorithm 61
4.2.2.1 Scheduling Field Operations for PA and PD 62
4.2.2.2 Parallelization for Binary Edwards Curve (BEC) . . 63
4.2.2.3 Parallelization for Generalized Hessian Curve (GHC) 64
4.2.2.4 Parallelization for Binary Generic Curve (BGC) . . . 64
4.2.3 Recovering the Final Coordinates of x and y 65
4.2.4 Latency of Point Multiplication Operations 66
4.3 Architecture of the Proposed Elliptic Curve Crypto-Processor 67
4.3.1 Field Arithmetic Unit (FAU) 68
4.3.2 A Fast and Low-Complexity Digit-Level GNB Multiplier over $GF(2^m)$ 69
4.3.2.1 Hardware Architecture 69
4.3.2.2 Complexities 70
4.3.2.3 LUT-based Critical-path Delay Analysis 71
4.3.2.4 Implementation 72
4.3.3 Memory and Control Unit 73
4.3.3.1 Memory 73
4.3.3.2 Control Unit 74
4.4 Comparisons and Implementations 75

4.4.1 Side-Channel Analysis . . . 75
4.4.2 Implementation Results and Discussion . . . 78
4.5 Conclusions . . . 83

5 New Architecture for Double-Multiplication Using GNB and Its Applications for Exponentiation and Elliptic Curve Cryptography 84
5.1 Hybrid Multiplication . . . 85
5.1.1 Traditional Multiplication Scheme . . . 86
5.1.2 Hybrid Multiplication Scheme . . . 86
5.1.2.1 Analysis . . . 87
5.2 Applications of the Proposed Hybrid Multiplier . . . 87
5.2.1 Double-Exponentiation . . . 87
5.2.2 Reducing the Latency of Point Multiplication on Binary Curves 90
5.2.2.1 Binary Edwards Curves . . . 90
5.2.2.2 Generalized Hessian Curves . . . 92
5.2.2.3 Binary Koblitz Curves . . . 93
5.2.2.4 Attacking ECC2K-130 . . . 94
5.3 Implementations . . . 95
5.4 Conclusion . . . 96

6 Highly Parallel and Fast Crypto-Processor for Point Multiplication on Koblitz Curves 98
6.1 Properties of Koblitz Curves . . . 99
6.1.1 Point Addition on Koblitz Curves . . . 100
6.1.1.1 Lopez-Dahab Projective Coordinates . . . 101
6.1.2 Point Multiplication on Koblitz Curves . . . 101
6.2 High-Speed Parallelization of Point Addition . . . 102
6.2.1 Latency of Point Multiplication . . . 104
6.3 Proposed Crypto-processor for Point Multiplication . . . 105
6.3.1 Field Arithmetic Unit (FAU) . . . 105
6.3.2 Control Unit and the Register File . . . 106
6.3.3 Coordinate Converter . . . 107
6.4 FPGA Implementations . . . 107
6.4.1 Comparisons . . . 108
6.5 Conclusion . . . 111

7 Summary and Future Work **112**
7.1 Thesis Contributions 112
7.2 Future Work 114

Bibliography **115**

Vita **124**

List of Tables

2.1 The Sequence of F for type 4 GNB over $GF(2^7)$ 10
2.2 The values of F for type 2 GNB over $GF(2^5)$ 15
2.3 Content of Variables in the LSB-first and MSB-first multiplication of $A = (01110)$ and $B = (10101)$ over $GF(2^5)$. 15

3.1 Comparison of number of XOR gates between bit-parallel GNB multipliers for $GF(2^{163})$ and $GF(2^{283})$. 34
3.2 Contents of variables in the proposed architecture for LSD-first DL-SIPO type 4 GNB multiplier over $GF(2^7)$. 41
3.3 Comparison of the most recently proposed type T digit-level GNB multipliers over $GF(2^m)$ with parallel outputs. 47
3.4 Area and time complexity comparison of bit-parallel GNB multipliers over $GF(2^m)$. Note that for Type T GNB: $C_N \leq Tm - T + 1$. 51
3.5 FPGA implementation of BL-SIPO (Fig. 2.1) multiplier for type 4 over $GF(2^{163})$ on xc4vlx100-ff1148 device. 52
3.6 ASIC synthesis results for BL-SIPO (Fig. 2.1) multiplier for type 4 over $GF(2^{163})$. 52
3.7 FPGA (Xilinx® Virtex™-4 xc4vlx100-ff1148 device) and ASIC (65-nm CMOS library) synthesis results for the improved DL-SIPO (Fig. 3.3) multiplier architectures for type 4 GNB over $GF(2^{163})$ for different digit sizes. 53
3.8 FPGA (Xilinx® Virtex™-4 xc4vlx100-ff1148 device) and ASIC (65-nm CMOS library) synthesis results for the improved DL-PISO (Fig. 3.5) multiplier architecture for type 4 GNB over $GF(2^{163})$ for different digit sizes. 53

3.9 FPGA (Xilinx® Virtex™-4 xc4vlx100-ff1148 device) and ASIC (65-nm CMOS library) synthesis results for the improved DL-PIPO (Fig. 3.1) multiplier architecture for type 4 GNB over $GF(2^{163})$ for different digit sizes. 54

4.1 Cost of point operations on binary Edwards curves (BECs), generalized Hessian curves (GHCs), and binary generic curves (BGCs) over $GF(2^m)$ [1], [2], and [3]. 58
4.2 Multiplier Utilization factors for data dependency graph of different curves. 65
4.3 Latency of the operations in the point multiplication with $\mathcal{M} = 1, 2, 3$, where M is the number of clock cycles required for multiplication of two arbitrary field elements. 66
4.4 Critical-path delay of the pipelined and non-pipelined architecture of presented digit-level type 4 GNB multiplier over $GF(2^{163})$. 71
4.5 LUT-based critical-path delay (CPD) (T_{LUT}) of the presented pipelined multiplier for different digit sizes (d) and levels of accumulation (ℓ) for type 4 GNB multiplier over $GF(2^{163})$ where $K = \lceil \frac{d}{\ell} \rceil$. 72
4.6 FPGA implementation results for BECs over $GF(2^{163})$ and $\mathcal{M} = 2$. . 76
4.7 FPGA implementation results for GHC over $GF(2^{163})$ and $\mathcal{M} = 2$. . 77
4.8 FPGA implementation results for BGC over $GF(2^{163})$ and $\mathcal{M} = 2$. . 77
4.9 Comparison of ECC implementations on FPGA over $GF(2^{163})$. . . . 80

5.1 Time delay evaluation of the proposed structure for type 4 GNB over $GF(2^{163})$. 88
5.2 ASIC and FPGA implementation results for the proposed low-complexity hybrid multiplier architecture (Fig. 5.1) over $GF(2^{163})$ for different digit sizes. 97

6.1 Comparison of the latency for performing point addition in the main loop on Koblitz curves in terms of number of multipliers 105
6.2 The implementation results of the point multiplication on Koblitz curves on Altera® Stratix™ II EP2S180F1020C3 FPGA device. 107
6.3 Comparison of related works for FPGA implementations of point multiplication on Koblitz curves using digit-level finite field multipliers. . 110

List of Figures

2.1 The architecture of (a) LSB-first bit-level SIPO (b) MSB-first bit-level normal basis multipliers [4] (c) The architecture of P module for type T GNB 14
2.2 The architecture of the digit-level PISO GNB multiplier [5]. 17
2.3 The architecture of Digit-level PIPO GNB multiplier proposed in [5], [6], where the i-fold right cyclic shift is denoted by $\overset{i}{\gg}$ and r is a number $0 \leq r \leq d-1$ such that $m = qd - r$. 19
2.4 Group law on Elliptic curve over $\mathbb{R}$. 21

3.1 The proposed improved architecture for DL-PIPO GNB multiplier . 29
3.2 Comparison between the number of XOR gates required in the DL-PIPO and the improved DL-PIPO for (a): $m = 163$ ($T = 4$), (b): $m = 283$ ($T = 6$). 33
3.3 (a) The proposed architecture for LSD-first DL-SIPO multiplier. (b) an example of the proposed multiplier for type 4 GNB over $GF(2^7)$ and $d = 2$. 37
3.4 Comparison among the numbers of XOR gates required in the original and the improved digit-level SIPO multiplier architectures [7] for (a) type $T = 4$ GNB over $GF(2^{163})$ and (b) type $T = 6$ GNB over $GF(2^{283})$. 43
3.5 (a) The architecture of the improved digit-level PISO GNB multiplier architecture with the LSD-first output. (b) The improved architecture of type 4 GNB multiplier over $GF(2^7)$ and $d = 2$. 44
3.6 Comparison among the numbers of XOR gates required in the original and improved digit-level PISO multiplier architectures for (a) type $T = 4$ GNB over $GF(2^{163})$ and (b) type $T = 6$ GNB over $GF(2^{283})$. 45
3.7 The architecture of proposed bit-parallel GNB multiplier 48

4.1 Data dependency graphs for parallel computing of the combined PA and PD operations on binary Edwards curves (a): $d_1 \neq d_2$ and (b): $d_1 = d_2$ assuming $\mathcal{M} = 2$. It requires five registers of T_1, T_2, T_3, T_4, and T_5. The constant parameters, $c_1 = \sqrt{d_1}$, $c_2 = \sqrt{d_2/d_1 + 1}$, $c_3 = \sqrt{c_1}$, and $c_4 = \sqrt{c_2}$ are assumed to be precomputed and stored in the memory. 62
4.2 Data dependency graph for parallel computing of the combined PA and PD operations for $\mathcal{M} = 2$ available multipliers on (a) generalized Hessian curves, assuming $c_1 = d^3$, and $c_2 = \frac{1}{\sqrt{d^3}}$ and (b) binary generic curves (BGCs) [8]. 63
4.3 Architecture of the proposed elliptic curve crypto-processor for binary Edwards, generalized Hessian, and binary generic curves. 67
4.4 The pipelined architecture of the low-complexity type T digit-level GNB multiplier with parallel-output [9]. 68
4.5 Time-Area ratio of the presented pipelined low-complexity digit-level GNB multiplier for type 4 over $GF(2^{163})$ for different digit sizes d. . 73
4.6 Configuration of BRAMs for the proposed architecture. 74
4.7 Implementation results of point multiplication for binary Edwards, generalized Hessian, and binary generic curves reported in Tables 4.6, 4.7, and 4.8 on Xilinx® Virtex™-5 xc5vlx110-2ff1760 FPGA device. The points are related to digit sizes of $d = 21, 24, 28, 33, 41, 55, 82$. . 81

5.1 (a) Proposed structure for the hybrid multiplier. (b) Two digit-level multipliers with parallel output operating in two separate steps. (c) A hybrid multiplier operating in one step and composed of an improved DL-PISO and an improved LSD-first DL-SIPO multipliers. 85
5.2 Architectures for multiplexer based double-exponentiation. (a) with one multiplier (b) with incorporating the proposed hybrid multiplier. 90
5.3 Data dependency graph for fast computation of combined PA and PD for binary Edwards curves (a): employing four different PIPO multipliers. (b): employing proposed hybrid multiplier. $c_1 = \sqrt{d_1}$, $c_2 = \sqrt{d_2/d_1 + 1}$, $c_3 = \sqrt{c_1}$, and $c_4 = \sqrt{c_2}$. 91
5.4 generalized Hessian curves with $c_1 = d^3$, and $c_2 = \frac{1}{\sqrt{d^3}}$, employing the proposed hybrid multiplier.Generalized Hessian curves 93

5.5 Parallel computation of point addition on Koblitz curves using Jacobian coordinates (a): with three finite field multipliers and (b): employing hybrid multiplier and three parallel multipliers. 95

6.1 Data dependency graph for parallel computation of point addition on Koblitz curves (a): using three finite field multipliers adopted from [10] (b): proposed scheme employing four multipliers. 103
6.2 The architecture of point multiplication crypto-processor 105
6.3 (a): Latency of point computation on Koblitz curves over $GF(2^{163})$ for different digit sizes. (b): Latency-area product of the proposed architecture for point multiplication. 108

List of Algorithms

2.1 Solving quadratic equation $X^2 + X = A$ using normal basis [11]. . . 12
2.2 Left-to-right Double-and-add point multiplication algorithm [11] . . . 23
2.3 Lopez-Dahab Scalar Multiplication [12] 24
4.1 Montgomery's algorithm [13] for point multiplication using w-coordinates. 61
6.1 Point multiplication on Koblitz curves using Double-and-add-or-subtract algorithm [11]. 102

List of Abbreviations

ASIC	Application-Specific Integrated Circuit
FPGA	Field Programmable Gate Arrays
CMOS	Complementary Metal-Oxide-Semiconductor
PA	Point Addition
PB	Point Doubling
PIPO	Parallel-in Parallel-out
SIPO	Serial-in Parallel-out
PISO	Parallel-in Serial-out
GNB	Gaussian Normal Basis
CPD	Critical Path Delay
ECC	Elliptic Curve Cryptography
GF	Galois Field
LSB	Least Significant Bit
LSD	Least Significant Digit
BEC	Binary Edwards Curve
BGC	Binary Generic Curve
BKC	Binary Koblitz Curve
GHC	Generalized Hessian Curve
MSB	Most Significant Bit
MSD	Most Significant Digit
NIST	National Institute of Standards and Technology
ECDLP	Elliptic Curve Discrete Logarithm Problem
ECDH	Elliptic Curve Diffie-Hellman
VHDL	Very-high-speed integrated circuit Hardware Description Language
VLSI	Very Large Scale Integrated
KDC	Key Distribution Center

Chapter 1

Introduction

THE history of cryptography is back to 2000 years ago (time of Julius Caesar) when it was required that two communicating parties to share a common secret, i.e., the symmetric key for encryption and decryption. The main problem of this approach is that the two parties must somehow met each other and agree on the common key. In 1976, Diffie and Hellman [14] demonstrated an algorithm for secure key exchange and lead to the development of today's public key cryptography systems known as RSA [15]. Recent technology of small and always connected devices such as mobile hand-held devices, RFID tags, near field communication (NFCs) devices, smart cards, and wireless sensor nodes (WSNs), to name a few, require efficient and high-performance computation of cryptographic protocols. The traditional schemes such as RSA is determined to be infeasible for these devices which resulted in adopting of a new technology based on elliptic curves which is called elliptic curve cryptography (ECC). ECC is proposed independently proposed by Neil Koblitz [16] and Victor Miller [17] for public-key cryptography and has gained significant attention in the recent researches available in the literature. The use of ECC has been identified as an efficient and suitable methodology to achieve public key cryptography in embedded and resource-constrained environments and approved by IEEE [18] and NIST [19] standards. The main advantage of ECC is that it offers similar security level compared to the RSA, employing smaller key size and providing efficient implementations for resource-constrained devices with limited storage, bandwidth, and silicon area. The security of ECC based cryptosystems relies on the difficulty of solving elliptic curve discrete logarithm problem (ECDLP) [19].

All these topics can be viewed as an applied science in the overlap between mathematics, computer science, and computer engineering.

1.1 Problem Statement and Motivation

Security in resource-constrained environments (such as smart cards, WSNs, Hand-held devices, and RFID tags) and high-performance web server (such as secure e-commerce transactions and online banking) highly requires efficient cryptographic computations (such as ECC). The former applications are suffering from availability of silicon area, while the latter ones are suffering from low speed of the current security protocols. Moreover, due to increasing number of small and connected devices to the servers efficient computation of cryptographic protocols are crucial.

Elliptic curves over finite fields can be represented using prime fields and binary extension fields. There are several implementations in the literature considering implementation of ECC over both fields. However, depending to the application and available resources prime fields have been chosen for software implementations and binary fields provide better performance over binary fields. Recently proposed schemes available in the the literature (for example, [20], [21], [10], [6], [22], [23], [24], [25], [26], and [27]) did not consider a systematic implementations of ECC over binary fields. For instance, they have employed available finite field multipliers in the literature without considering their performance for the proposed crypto-processors. The hierarchy of ECC computations requires an efficient computations in the lower level, i.e., finite fields and then the curve and protocol levels. Therefore, a bottom-up approach to design an ECC crypto-processor targeting the certain applications is one of most important task that one need to explore.

Also, in some of the previous researches parallelism is known as the only method to reduce the latency of curve level arithmetic computations to increase the speed of overall point multiplication on ECC-based crypto-processors. However, one should note that due to the data dependencies between curve level computations, parallelism is not applicable in several situations such as point multiplication on binary Edwards curves and double-exponentiation for elliptic curve digital signature verifications. These dependencies will limit the speed of the designs according to the number of parallel processors.

1.2 Objectives of the Thesis

In this thesis, efficient and low complexity ECC-based crypto-processors are proposed. A bottom-up approach is proposed in designing a crypto-processor with devising low complexity finite field arithmetic units. This thesis, not only considers

standard curves available in the literature, but it also describes efficient implementation of newly introduced complete binary elliptic curves such as binary Edwards and generalized Hessian curves. The objectives of this thesis are to design high performance and fast ECC-based crypto-processors for web servers and as well as designing low-complexity and efficient ones for small and hand-held devices based on different security level and key size.

1.3 Thesis Outline

This thesis is organized as follows. In Chapter 2, we will provide a literature review on some of the existing works in the literature on normal basis multiplication and elliptic curve cryptography.

In Chapter 3, we present low-complexity Gaussian normal basis multiplier architectures including parallel-in-parallel-out, parallel-in-serial-out, and serial-in-parallel-out. Also, we propose a low-complexity architecture for bit-parallel multiplication in this chapter.

In Chapter 4, we propose an efficient ECC-based crypto-processor on binary Edwards and generalized Hessian curves employing a parallel-in parallel-out digit-level GNB multiplier proposed in Chapter 3. The implementation results are provided and compared with the counterparts in the literature.

In Chapter 5, based on the low-complexity digit-level multiplier architectures proposed in Chapter 3, a new hybrid multiplier to perform double-multiplication is proposed. Also, in this chapter we evaluate the efficiency of the new hybrid multiplier and its application for reducing the latency of double-exponentiation and point multiplication on binary elliptic curves.

In Chapter 6, a highly parallel and fast ECC crypto-processor for point multiplication on Koblitz curves is presented. The implementation results are reported and compared with the leading ones in the literature.

Finally, in Chapter 7, we summarize our contributions and provide possible directions for future works.

Chapter 2

Preliminaries and Literature Review

IN this chapter, we provide preliminaries and review the previous works available in the literature on farithmetic of finite fields and elliptic curve cryptography. The following discussion is based on comprehensive presentations given in [28], [29], [11], and [30].

2.1 Finite Fields

Finite fields are usually referred to as Galois fields (to honor Evariste Galois 1811-1832, a French mathematician) and have importance in many applications such as cryptography, network coding, and error control theory. Due to these applications their implementations have been studied extensively by computer engineers and computer scientists. A finite field consists of a finite set of objects called field elements together with the description of two operations (addition and multiplication) that can be performed on pairs of field elements. Finite field arithmetic plays an important role in ECC and all the low-level operations are carried out in these fields. It is important to describe these fields in order to closely specify cryptographic methods based on ECC.

A set G and a binary operation $\star$ form a group $(G, \star)$ if they satisfy the following five properties:

1. The operation $\star$ is closed (i.e., $a \star b \in G$ for all $a, b \in G$).
2. The operation $\star$ is associative (i.e., $a \star (b \star c) = (a \star b) \star c$ for all $a, b, c \in G$).
3. The operation $\star$ is commutative (i.e., $a \star b = b \star a$ for all $a, b \in G$). In this case set $(G, \star)$ called Abelian.

4. There exists an identity element $e \in G$ such that $e \star a = a \star e = a$ for all $a \in G$.

5. For every $a \in G$, there exists an inverse element $b \in G$ such that $a \star b = e$.

The group $(G, \star)$ with group operation to be multiplication $\times$ is known as multiplicative group $(G, \times)$ which its identity element is 1 and the inverse element is denoted by $a^{-1} \in G$. Similarly, for group operation with addition $(G, +)$ the identity element is 0 and inverse element is $-a$. The order of the group, $\text{ord}(G)$, is the number of elements in the set$G$. The group G is finite if $\text{ord}(G)$ is finite. The order of an element $a \in G$, i.e., $\text{ord}(a)$, is the smallest positive integer, n, for which $a^n = e$.

The group G is cyclic if all its of the group can be generated by applying the group operation repeatedly to an element a and hence$a$ is a generator of G.

A field $\mathbb{F}$ is a set of elements with two binary operators, denoted as $+$ (addition) and $\times$ (multiplication) which exhibits the following properties:

1. $\mathbb{F}$ is an abelian group under the addition $+$ operation.

2. The non-zero elements of $\mathbb{F}$ form an abelian group under the operation $\times$.

3. The operation $\times$ is distributive over the operation $+$, i.e., $a \times (b + c) = (a \times b) + (a \times c)$ and $(b + c) \times a = (b \times a) + (c \times a)$ for all $a, b, c \in \mathbb{F}$.

A field $\mathbb{F}$ with q elements is said to be finite if q is finite and is denoted by $\mathbb{F}_q$ which is also referred to as Galois field as $GF(q)$. The order of $\mathbb{F}_q$ is the number of elements in $\mathbb{F}_q$, and $\mathbb{F}_q$ exists if and only if q is prime or a power of a prime, i.e., $q = p^m$ for $m \geq 1$. Then, for $m = 1$ it is called a prime field and for $m \geq 2$ it is called an extension field. Extension fields with $p = 2$, i.e., $\mathbb{F}_{2^m}$ or $GF(2^m)$ are called binary fields (or fields with characteristic two) which can be seen as a vector space of dimension m over the field $\mathbb{F}_2$ which has only 0 and 1.

As defined above $\mathbb{F}_q$ has two main operations, i.e., addition and multiplication. Subtraction and inversion can be defined through addition (i.e., $a - b = a + (-b)$ where $b + (-b) = 0$) and multiplication ($a/b = a \times b^{-1}$ where $b \times b^{-1} = 1$ and $b \in \mathbb{F}_q - \{0\}$), respectively.

Definition 2.1. An element α in a finite field $\mathbb{F}_q$ is called a primitive element (or generator) of $\mathbb{F}_q$ if $\mathbb{F}_q = \{0, \alpha, \alpha^2, \cdots . \alpha^{q-1}\}$.

Definition 2.2. The order of a non-zero element $\alpha \in \mathbb{F}_q$ denoted by $\text{ord}(\alpha)$, is the smallest positive integer k such that $\alpha^k = 1$.

Definition 2.3. The non-zero elements in $\mathbb{F}_q$ form a multiplicative group of $\mathbb{F}_q$ denoted by $\mathbb{F}_q^\star$ which is cyclic with $\text{ord}(\mathbb{F}_q^\star) = q - 1$. Hence

$$a^q = a, \tag{2.1}$$

for all $a \in \mathbb{F}_q$. This is also known as Fermat's Little Theorem as $a^p \equiv a(\text{mod}p)$.

Then, for $\mathbb{F}_{2^m}$ the order of a multiplicative group is $2^m - 1$ and for an element $A \in \mathbb{F}_{2^m}$ one has $A^{2^m-1} = 1$. In this thesis, we use $GF(2^m)$ to indicate binary Galois fields instead of $\mathbb{F}_{2^m}$.

2.2 Binary Fields Arithmetic

The binary field of characteristic two, $GF(2^m)$ is a finite field [30] that contains 2^m different elements. The elements of $GF(2^m)$ are represented as a vector space over $GF(2)$ which contains 0 and 1 with respect to a basis. As the two elements of $GF(2)$ can be represented with a bit, m bits are required to represent elements of $GF(2^m)$. The binary field, $GF(2^m)$, is associated with an irreducible polynomial (i.e., can not be represented as a product of two polynomials with positive degrees) $F(z)$, with $\deg(F(Z)) = m$ over $GF(2)$, i.e.,

$$F(z) = f_m z^m + f_{m-1} z^{m-1} + \cdots + f_1 z + f_0, f_i \in GF(2). \tag{2.2}$$

If $f_m = 1$ the $\deg[F(z)] = m$. Addition of two elements in $GF(2^m)$ is simply performed bit-wise (modulo 2) XOR operation but the multiplication depends on the field basis and dependencies between the field elements. From implementation point of view binary fields are faster than prime fields as they provide carry-free operations. The field elements can be represented using polynomial (or standard) basis, normal basis, dual basis, and redundant basis. However, polynomial and normal bases are two common type of bases that has been used in conventional hardware and software applications and approved and recommended by the international standards such as IEEE and NIST. In the following, we review briefly polynomial basis and explain normal basis in detail as it is used in this thesis.

2.2.1 Polynomial Basis

Let $\alpha \in GF(2^m)$ be a root of the primitive polynomial $F(z)$, i.e., $F(\alpha) = 0$. Then the set $\{1, \alpha, \alpha^2, \cdots, \alpha^{m-1}\}$ is known as the polynomial basis and an element $A \in GF(2^m)$

can be represented as linear combinations of this set with a polynomial of degree $m-1$ over $GF(2)$, as $A = \sum_{i=0}^{m-1} a_i \alpha^i$, where $a_i \in GF(2)$. For simplicity, a bit-vector representation is commonly used and so that $A = (a_{m-1}, a_{m-2}, \cdots, a_1, a_0)$, where a_{m-1} and a_0 are the most significant bit (MSB) and least significant bit (LSB), respectively. In polynomial basis the identity element of addition, i.e., 0, is $(0, 0, \cdots, 0, 0)$ and the identity element of multiplication, i.e.,1, is $(0, 0, \cdots, 0, 1)$.

Addition of two field elements, say, $A = (a_{m-1}, \cdots, a_1, a_0)$ and $B = (b_{m-1}, \cdots, b_1, b_0)$ in $GF(2^m)$ represented by polynomial basis is $C = A + B$ and can be obtained by pair-wise addition of the coordinates of A and B over $GF(2)$ (i.e., modulo 2 addition) as $c_i = a_i \oplus b_i$. Multiplication of two field elements $A, B \in GF(2^m)$ is complicated. First, A and B are multiplied by using ordinary polynomial multiplication and then the intermediate product needs further reduction by $F(x)$, i.e., $A \cdot B \bmod F(x)$. The squaring in polynomial basis is also complicated and its complexity depends on the irreducible polynomial $F(x)$ [31, 32, 33, 34].

2.2.2 Normal Basis

It is shown that there exists a normal basis for the binary extension field $GF(2^m)$ for all positive integers m. The normal basis is constructed by finding a normal element $\beta \in GF(2^m)$, where β is a root of an irreducible polynomial of degree m. Then set $N = \{\beta, \beta^2, \cdots, \beta^{2^{m-1}}\}$ is a basis for $GF(2^m)$ and its elements are linearly independent. In this case, $A \in GF(2^m)$, can be represented as $A = \sum_{i=0}^{m-1} a_i \beta^{2^i}$, where $a_i \in GF(2)$. The identity element of addition, i.e., 0, is $(0, 0, \cdots, 0, 0)$ and the identity element of multiplication, i.e., 1, is $(1, 1, \cdots, 1, 1)$ as $1 = \beta + \beta^2 + \beta^{2^2} + \cdots + \beta^{2^{m-1}}$.

Normal basis is attractive mainly because it provides efficient computation for squaring. For an element, say, $A \in GF(2^m)$ its power of two can be written as $A^2 = \sum_{i=0}^{m-1} a_i \beta^{2^{i+1}}$ and one can get $\beta^{2^m} = \beta$ from (2.1). Then, squaring is a linear operation and for $A = (a_0, a_1, \cdots, a_{m-1}) \in GF(2^m)$ one can obtain it by a right cyclic shift operation as $A^2 = (a_{m-1}, a_0, a_1, \cdots, a_{m-2})$. Similar to the polynomial basis the addition can be obtained by bit-wise XOR operation for two given elements A and B as $A + B = \sum_{i=0}^{m-1} (a_i \oplus b_i) \beta^{2^i}$.

2.2.3 Finite Field Multiplication

Among finite field representations, normal basis is more efficient in hardware implementations since squaring of a field element over $GF(2^m)$ can be performed by a simple cyclic shift. This makes normal basis more attractive for the cryptosys-

tems that utilize frequent squarings (e.g., point multiplication on Koblitz curves and exponentiation-based cryptosystems).

2.2.3.1 Multiplication Using Normal Basis

Let $A = (a_0, a_1, \cdots, a_{m-1}) = \sum_{i=0}^{m-1} a_i \beta^{2^i}$ and $B = (b_0, b_1, \cdots, b_{m-1}) = \sum_{j=0}^{m-1} b_j \beta^{2^j}$ be two field elements in $GF(2^m)$. Let $C \in GF(2^m)$ be their product, i.e., $C = (c_0, c_1, \cdots, c_{m-1}) = AB = \sum_{i=0}^{m-1} \sum_{j=0}^{m-1} a_i b_j \beta^{2^i+2^j}$. Let us represent the field element $\beta^{2^i+2^j} \in GF(2^m)$, $0 \leq i, j \leq m-1$, with respect to $N = \{\beta, \beta^2, \cdots, \beta^{2^{m-1}}\}$ as $\beta^{2^i+2^j} = \sum_{l=0}^{m-1} \mu_{i,j}^{(l)} \beta^{2^l}$.Then, one can find C as

$$C = \sum_{i=0}^{m-1} \sum_{j=0}^{m-1} a_i b_j \sum_{l=0}^{m-1} \mu_{i,j}^{(l)} \beta^{2^l} = \sum_{l=0}^{m-1} \sum_{i=0}^{m-1} \sum_{j=0}^{m-1} a_i b_j \mu_{i,j}^{(l)} \beta^{2^l}. \tag{2.3}$$

By representing C with respect to N, i.e., $C = \sum_{l=0}^{m-1} c_l \beta^{2^l}$, and equating with (2.3), the l-th coordinate of C can be written as $c_l = \sum_{i=0}^{m-1} \sum_{j=0}^{m-1} a_i b_j \mu_{i,j}^{(l)}$. Then, it can be written in a matrix form as

$$c_l = \underline{a} \mathbf{M}^{(l)} \underline{b}^{tr},\ 0 \leq l \leq m-1, \tag{2.4}$$

where $\mathbf{M}^{(l)} = [\mu_{i,j}^{(l)}]_{i,j=0}^{m-1}$, $\mu_{i,j}^{(l)} \in GF(2)$, $0 \leq i, j \leq m-1$, $\underline{a} = [a_0, a_1, \cdots, a_{m-1}]$ and $\underline{b}^{tr}$ denotes the matrix transpose of row vector $\underline{b} = [b_0, b_1, \cdots, b_{m-1}]$. In (2.4), $\mathbf{M}^{(l)}$ is obtained from the l-fold right and down circular shifts of the *multiplication matrix* $\mathbf{M} = \mathbf{M}^{(0)}$. The computation of entries of $\mathbf{M}$ can be found from [18]. Massey and Omura in [35] have proposed a bit-level PISO multiplier by implementing (2.4) for one coordinate, say $c_0 = \underline{a} \mathbf{M} \underline{b}^{tr} = F(A, B)$. Then, the l-th coordinate of C is obtained by left cyclic shifts of the coordinates of A and B, i.e., $c_l = F(A \ll l, B \ll l)$ [35]. The number of ones, C_N, $2m-1 \leq C_N \leq m^2$, in $\mathbf{M}$ defines the complexity of the multiplication. It is well known that for $C_N = 2m-1$, the normal basis is called optimal normal basis (ONB) [36]. There are two types of ONBs, referred to as Type I and Type II ONBs. It should be noted that ONB does not exist for all m, for example $m = 163$. As an extension of the work on ONBs a low complexity of normal bases of type T, $T > 1$, is proposed by Ash et al. which are referred to as Gaussian normal basis (GNB). For $T = 1$ and 2, the GNBs become the two types of ONBs of [36] and hence, $C_N \leq Tm - T + 1$. In Chapter 2, we will discuss multiplication on GNB in more details as it is the one that has been employed in this thesis and has been included in many international standards [18] and [19].

2.2.3.2 Multiplication Using Gaussian Normal Basis

GNB has been constructed by Ash *et al.* [37] and is a special class of normal basis which is included in the IEEE 1363 [18] and NIST [19] standards and exists for every $m > 1$ that is not divisible by eight [29].

Definition 2.4. [29] Let $p = mT + 1$ be a prime number and $gcd(mT/k,\ m) = 1$, where k is the multiplication order of 2 module p. Then, the normal basis $N = \{\beta, \beta^2, \cdots, \beta^{2^{m-1}}\}$ over $GF(2^m)$ is called the Gaussian normal basis (GNB) of type T, $T > 1$.

The complexities of type T GNB multiplier in terms of time and area depend on $T > 1$. In this thesis, we only consider the GNBs with odd values of m which implies that T is an even number. Such GNBs cover all five binary fields, i.e., $m \in \{163, 233, 283, 409, 571\}$, recommended by the IEEE 1363 [18] and NIST [19] standards for ECDSA The corresponding types for these fields are $T = 4, 2, 6, 4$, and 10, respectively.

Let $A = (a_0, a_1, \cdots, a_{m-1}) = \sum_{i=0}^{m-1} a_i \beta^{2^i}$ and $B = (b_0, b_1, \cdots, b_{m-1}) = \sum_{j=0}^{m-1} b_j \beta^{2^j}$ be two field elements over $GF(2^m)$ and assume $C \in GF(2^m)$ be their product, i.e., $C = (c_0, c_1, \cdots, c_{m-1}) = AB$. Then, the first coordinate of C, i.e., c_0 can be obtained from an explicit formula given in [18] as follows

$$\begin{aligned} c_0 &= a_0 b_1 + \sum_{k=2}^{p-2} a_{F(k)} b_{F(k+1)}, \\ &= a_0 b_1 + \sum_{i=1}^{m-1} a_i \left(\sum_{F(k)=i} b_{F(k+1)} \right), 2 \le k \le p-2, \end{aligned} \tag{2.5}$$

where in (2.5), the sequence $F(1), F(2), \cdots, F(p-1)$ can be obtained by precomputation using

$$F(k) = F(2^i u^j \bmod p) = i,\ 1 \le i \le m-1,\ 0 \le j < T, \tag{2.6}$$

where u is an integer of order T mod p and $p = Tm + 1$ [18]. In Table the sequence of F for type 4 GNB over $GF(2^7)$ is given. It is noted that for each i, $1 \le i \le m-1$, $F(k+1)$, $2 \le k \le p-2$ in (2.5), can be used as entries of a $(m-1) \times T$ matrix $\mathbf{R}$. Let us denote the (i, j)-th element of this matrix as $R(i, j)$, $0 \le R(i, j) \le m-1$, $1 \le i \le m-1$, $1 \le j \le T$. Each row of the matrix $\mathbf{R}$, contains T entries of integer in $[0, m-1]$. Then, one can write c_0 as [5]

Table 2.1: The Sequence of F for type 4 GNB over $GF(2^7)$

k	1	2	3	4	5	6	7	8	9	10	11	12	13	14
$F(k)$	0	1	5	2	1	6	5	3	3	2	4	0	4	6
K	15	16	17	18	19	20	21	22	23	24	25	26	27	28
$F(k)$	6	4	0	4	2	3	3	5	6	1	2	6	1	0

$$c_0 = a_0 b_1 + \sum_{i=1}^{m-1} a_i \left(\sum_{j=1}^{T} b_{R(i,j)} \right). \tag{2.7}$$

Note that, to obtain the lth coordinates of C, i.e., c_l one needs to add "l mod m" to all indices in (2.7). Therefore, one can find all coordinates of C as follows:

Lemma 2.1. [5] *The product of A and B in $GF(2^m)$ is*

$$C = (A \odot (B \ll 1)) \oplus \sum_{i=1}^{m-1} (A \ll i) \odot S(i, B), \tag{2.8}$$

where

$$S(i, B) = ((B \ll R(i,1)) \oplus (B \ll R(i,2)) \oplus \cdots \oplus (B \ll R(i,T))), 1 \leq i \leq m-1. \tag{2.9}$$

and $(X \ll i)$ is the i-fold left cyclic shift of $X \in GF(2^m)$ and $X \odot Y = (x_0 y_0, \cdots, x_{m-1} y_{m-1})$ and $X \oplus Y = (x_0 + y_0, \cdots, x_{m-1} + y_{m-1})$ denote bit-wise AND and XOR operations between coordinates of X and Y, respectively.

Remark 2.1. From (2.6) one can realize that for $T > 2$ there are situations (for example $F(k) = \frac{m-1}{2}$ and $F(k) = \frac{m+1}{2}$ for $T = 4$) where matrix $\mathbf{R}$ contains (two) equal entries.

2.2.3.3 Inversion

Inversion, i.e., for a given element $A \in GF(2^m)$ finding an element $A^{-1} \in GF(2^m)$ such that $A \cdot A^{-1} = 1$, is considered an expensive operation. It is commonly required in cryptographic applications of finite fields and its efficient implementation is important. There are two ways to compute inversion over finite fields: extended Euclidean algorithm and Fermat's Little Theorem [38]. The inversion based on Fermat's Little Theorem uses consecutive squarings and multiplication and is more suitable while field elements are represented by normal basis. Based on Definition 2.3, it follows

that $A^{2^m-2} = A^{-1}$ and its computation (i.e., exponentiation) requires $m-1$ squarings and $m-2$ multiplications as $2^m - 2 = (11, \cdots, 110)_2$. However, Itoh and Tsuji [38] proposed an efficient algorithm which reduces the number of multiplications to $\lfloor log_2(m-1) \rfloor + H(m-1) - 1$, where $H(m-1)$ represents the Hamming weight of $(m-1)$.

2.2.3.4 Trace and Quadratic Equation Solution

The trace function Tr: $GF(2^m) \rightarrow GF(2)$ is a linear map and for an element $A = (a_0, a_1, \cdots, a_{m-1}) \in GF(2^m)$ is defined as $\mathrm{Tr}(A) = \sum_{i=0}^{m-1} A^{2^i} \in \{0, 1\}$. For normal basis, when m is odd trace of element A can be computed as $\mathrm{Tr}(A) = \sum_{i=0}^{m-1} a_i$, which is bit-wise XOR operation of all bits of vector A.

The quadratic equation $X^2 + X = A$ for $X = (x_0, x_1, \cdots, x_{m-1}) \in GF(2^m)$ has a solution if and only if $\mathrm{Tr}(A) = 0$, and hence if X is a solution, then $X+1$ is a solution. In normal basis the solution can be found bit-wise. However, in polynomial basis it is complicated and needs half-trace computations which requires $m-1$ squarings and $(m-1)/2$ additions [11]. In Algorithm 2.1, an efficient algorithm to solve quadratic equation using normal basis is presented. The cost of solving quadratic equation using normal basis is only $m-2$ additions.

Example 2.1. Let element $A = \beta + \beta^{16} = (10001)$ in the finite field $GF(2^5)$ for type 2 GNB. Then, the solutions of the quadratic equation $X^2 + X = A$ can be obtained using Algorithm 2.1. First, we check that $\mathrm{Tr}(A) = \sum_{i=0}^{4} a_i = 1 + 0 + 0 + 0 + 1 = 0$. Then, X can be obtained bit-wise as $x_0 = 1$, $x_1 = 1$, $x_2 = 1$, $x_3 = 1$, and $x_4 = 0$ so $X = (11110)$. Also, $X + 1$ is solution too, i.e., $X + 1 = (11110) + (11111) = (00001)$. These two solutions satisfy the quadratic equation. As seen the cost of solving this equation is only 3 module two additions (i.e., XORing).

In Chapter 4, we employ this algorithm to solve a quadratic equation for recovering final point of point multiplication algorithm.

2.2.4 Multiplier Architectures

The implementation of finite field multipliers using normal basis and more specifically GNB can be categorized, in terms of their structures, into three groups: (i) bit-level which includes: parallel-in serial-out (PISO) [35], serial-in parallel-out (SIPO) [39], [4], [40], and parallel-in parallel-out (PIPO) [41], [42], (ii) digit-level including the structures of: parallel-in serial-out (PISO) [43], parallel-in parallel-out (PIPO) [44],

Algorithm 2.1 Solving quadratic equation $X^2 + X = A$ using normal basis [11].

Input: $A = (a_0, a_1, \cdots, a_{m-1}) \in GF(2^m)$.
Output: $X = (x_0, x_1, \cdots, x_{m-1}) \in GF(2^m)$.
Step 1: $x_0 \leftarrow a_0$.
Step 2: For i from 1 to $m-2$ do
$x_i \leftarrow a_i \oplus x_{i-1}$.
end for
Step 3: $x_{m-1} \leftarrow 0$.
Step 4: Return X.

---

[5], [45], and serial-in parallel-out (SIPO) [46], and (iii) bit-parallel which includes: [47], [48], [49], and [50] multipliers.

2.2.4.1 Bit-Level NB Multiplication

Bit-level multipliers provide the lowest possible area complexity. The first bit-level normal basis multiplier has been invented by Massey and Omura [35] which all coordinates of both input operands should be presented during multiplication operation. It is also known as a sequential multiplier with serial output in the literature [43]. Bit-level SIPO multipliers have been studied for normal basis and two different structures, namely Least Significant Bit (LSB) first and Most Significant Bit (MSB) first structures, have been proposed by Beth and Gollmann in [4]. A PIPO version of their multiplier is also presented in [41] and its time and area complexities are derived.

Based on the way the input bits are processed and the output bits are produced there are four kinds of of bit-level normal basis multipliers. They are called the LSB-first and the MSB-first bit-level SIPO multipliers [31] and the LSB-first and the MSB-first PISO normal basis multipliers [35].

LSB-first bit-level SIPO normal basis multiplier

In an LSB-first bit-level multiplication, having all elements of one operand, say B, to be present, the other operand, i.e., A, is processed from its LSB, i.e., a_0, and in each clock cycle one bit is processed. In [4], Beth and Gollmann presented an architecture for bit-level multiplication using normal basis. The key formulation of this multiplier is presented below.

Lemma 2.2. [4] *Let A and B be two elements of $GF(2^m)$ and C be their multiplica-*

tion, i.e., $C = AB$ as

$$C = \sum_{i=0}^{m-1} \left(a_i \beta^{2^i}\right) B = \sum_{i=0}^{m-1} \left(a_i \cdot \beta B^{2^{-i}}\right)^{2^i}$$
$$= a_0 \beta B + a_1 \left(\beta B^{2^{-1}}\right)^2 + \cdots + a_{m-1} \left(\beta B^{2^{-(m-1)}}\right)^{2^{m-1}}, \tag{2.10}$$

then similar to Horner's rule one can obtain

$$C = \left(\left(\cdots \left((a_0 \beta B)^{2^{-1}} + a_1 \beta B^{2^{-1}} \right)^{2^{-1}} + \cdots \right)^{2^{-1}} + a_{m-1} \beta B^{2^{-(m-1)}} \right)^{2^{-1}}.$$

Let us denote $P(B) = \beta B \in GF(2^m)$ as a field element in GNB. In [5], $P(B)$ can be obtained for GNB multiplier based on the $\mathbf{R}$ matrix as

$$P(B) = (b_1, s_0(1, B), s_0(2, B), \cdots, s_0(m-1, B)), \tag{2.11}$$

where $s_0(i, B) = \sum_{j=1}^{T} b_{R(i,j)} \in \{0, 1\}$, $1 \leq i \leq m-1$. Then using (2.11) and Lemma 1, we can state the following.

Corollary 2.1. *For GNB, the product of $A = (a_0, a_1, \cdots, a_{m-1}) \in GF(2^m)$, given in bit-serial fashion, and $B \in GF(2^m)$ can be written as*

$$C = \left(\left(\cdots \left(\left(a_0 P(B)\right) \ll 1 + a_1 P(B \ll 1) \right) \ll 1 + \cdots \right) \ll 1 + a_{m-1} P(B \ll m-1) \right) \ll 1, \tag{2.12}$$

where "$\ll$" denotes a left cyclic shift.

Equation (2.12) can be realized by an architecture depicted in Fig. 2.1a. The implementation of $P(B) \in GF(2^m)$ given in (2.11) is performed by a P module shown in Fig. 2.1c for type T GNB. The product of $a_i P(B)$ in Fig. 2.1a. denotes bit-wise AND operation between a_i and elements of $P(B)$ and is performed using m 2-input AND gates. Also the sum (adder block in Fig. 2.1a) is implemented using m 2-input XOR gates. As one can see from Fig. 2.1a. all bits of the operand B are available, while the coordinates of the operand A should be available in serial fashion with the LSB first, i,e, a_0. In this architecture, both m-bit registers $\langle Y \rangle = \langle y_0, y_1, \cdots, y_{m-1} \rangle$ and $\langle Z \rangle = \langle z_0, z_1, \cdots, z_{m-1} \rangle$ should be initialized with operand $B = (b_0, b_1, \cdots, b_{m-1})$

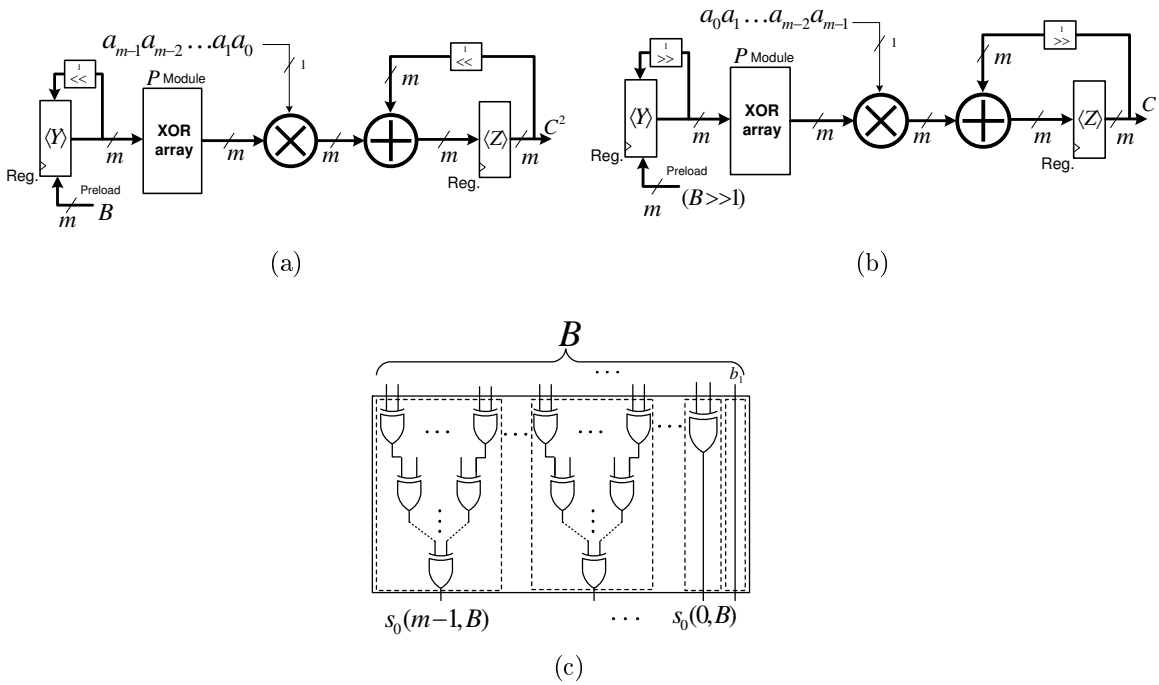


Figure 2.1: The architecture of (a) LSB-first bit-level SIPO (b) MSB-first bit-level normal basis multipliers [4] (c) The architecture of P module for type T GNB .

and $0 = (0, 0, \cdots, 0)$ (i.e., $Y(0) = B$ and $Z(0) = 0$), respectively. Let $Z(0)$ denotes the initial value of the register $\langle Z \rangle$ and $Z(i)$, $1 \leq i \leq m$, be the content of the register $\langle Z \rangle$ in the clock cycle i. After one clock cycle the content of $\langle Z \rangle$ is $Z(1) = a_0 P(B) \in GF(2^m)$. Then, the registers $\langle Y \rangle$ and $\langle Z \rangle$ are cyclically shifted to the left according to (2.12). A one can verify, after m-th clock cycle the register $\langle Z \rangle$ contains the coordinates of $Z(m) = C^2 = (c_{m-1}, c_0, c_1, \cdots, c_{m-2})$ (see (2.12)). Thus, C can be obtained by a left cyclic shift of register $\langle Z \rangle$, i.e., $C = (Z(m) \ll 1)$. The presented architecture requires at most $(T-1)(m-1)$ XOR gates in the P module, m XOR gates for the adder, m AND gates, and two m-bit registers. Also, its critical-path delay due to delays through the P module ($\lceil \log_2 T \rceil T_X$), AND gates ($T_A$), and XOR gates ($T_X$) is $T_A + (1 + \lceil \log_2 T \rceil) T_X$.

The MSB-first bit-level SIPO normal basis multiplier

In a MSB-first bit-level SIPO GNB multiplication, the operand A is processed from its MSB, i.e., a_{m-1}, and in each clock cycle one bit is considered.

Let A, B be two elements of $GF(2^m)$ and C be their product, i.e., $C = AB$, then similar to Horner's rule one can obtain [4]:

Table 2.2: The values of F for type 2 GNB over $GF(2^5)$

k	1	2	3	4	5	6	7	8	9	10
$F(k)$	0	1	3	2	4	4	2	3	1	0

Table 2.3: Content of Variables in the LSB-first and MSB-first multiplication of $A = (01110)$ and $B = (10101)$ over $GF(2^5)$.

j	LSB-first			MSB-first		
	Y	A	Z	Y	A	Z
0	10101	–	00000	11010	–	00000
1	10101	0	00000	11010	0	00000
2	01011	1	11011	01101	1	10100
3	10110	1	10000	10110	1	01101
4	01101	1	10101	01011	1	01101
5	11010	0	$C^2 = 01011$	10101	0	$C = 10110$

$$C = AB = \left(\cdots \left(\left(a_{m-1} \beta B^{2^{-(m-1)}} \right)^2 + a_{m-2} \beta B^{2^{-(m-2)}} \right)^2 + \cdots \right)^2 + a_0 \beta B. \quad (2.13)$$

To realize the implementation of (2.13), one needs to perform multiplication by β as $\beta B = \underline{\beta}^{tr} \cdot (\underline{\beta} \cdot \underline{b}^{tr}) = (\underline{\beta}^{tr} \cdot \underline{\beta}) \cdot \underline{b}^{tr} = \mathbf{M} \cdot \underline{b}^{tr}$ which is a matrix-by-vector multiplication for GNB and then compute C as

$$C = \Big(\cdots \Big(\big(a_{m-1} \odot P(Y) \big) \gg 1 + a_{m-2} \odot P(Y \gg 1) \Big) \gg 1 + \cdots \Big) \gg 1 + a_0 \odot P(Y \gg m-1),$$

where $Y = B^{2^{-(m-1)}} = B^2$. The architecture for the MSB-first SIPO GNB multiplication is depicted in Fig. 2.1b. As one can see every bit of operand B is available, while operand A should be available in serial with the MSB first. In this multiplier structure, both registers $\langle Y \rangle$ and $\langle Z \rangle$ are initialized to $Y = (B \gg 1) = (b_{m-1}, b_0, b_1, \cdots, b_{m-2})$ and $0 = (0, 0, \cdots, 0)$, respectively. In the first clock cycle, the register $\langle Z \rangle$ contains $Z(1) = a_{m-1} \odot P(B \gg 1)$. Then, registers $\langle Y \rangle$ and $\langle Z \rangle$ should be cyclically shifted to the right. Thus, as one can verify, after m-th clock cycle the register $\langle Z \rangle$ contains the coordinates of C, i.e., $Z(m) = C$.

2.2.4.2 An Example

Consider the finite field $GF(2^5)$ generated for type 2 GNB and we have the following multiplication matrix from Table 2.2 given in [19] as

$$\mathbf{M} = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 \end{pmatrix}_{5\times5}, \mathbf{R} = \begin{pmatrix} 0 & 3 \\ 3 & 4 \\ 1 & 2 \\ 2 & 4 \end{pmatrix}_{4\times2}.$$

Let $A = (01110)$ and $B = (10101)$ be two field elements in $GF(2^5)$. Based on the the architectures depicted in Fig. 2.1, Table 2.3 illustrates the contents of various variables of registers $\langle Y \rangle$ and $\langle Z \rangle$ which are updated with the clock cycles. For the MSB-first structure, first, registers $\langle Y \rangle$ and $\langle Z \rangle$ are initialized (in row with j being 0) with $B^{2^{-4}} = B^2 = 11010$ and 00000, respectively. Then, after $j = 5$ clock cycles the register $\langle Z \rangle$ contains the product, i.e., $C = 10110$. For the LSB-first structure, in the initialization step, registers $\langle Y \rangle$ and $\langle Z \rangle$ are loaded with operand B and 00000, respectively. Then, after 5 clock cycles the register $\langle Z \rangle$ contains $C^2 = 01011$. Therefore, after a left cyclic shift (i.e., rewiring) one can obtain the result of the multiplication as $C = 10110$.

2.2.4.3 Digit-level GNB multiplication

Digit-level multipliers are alternatives for bit-level and bit-parallel multipliers in which the digit size can be chosen depending on the amount of the resources available. A digit-level PIPO version of Massey-Omura multiplier [51] and its improved version [44] are used in ECC based crypto-processors in [10], [6], and [26]. It has been mentioned that in order to satisfy high speed and low complexity requirements of cryptographic applications, there is a need to design efficient architectures for finite field multiplication using normal basis. In [5], two efficient digit-level PISO and PIPO GNB multipliers are presented in [9], a subexpression sharing algorithm is introduced and applied to obtain the least number of gates for the digit-level PIPO multiplier. In the following, we summarize the contributions of this work.

2.2.4.4 Digit-level PISO GNB multiplier

In [5], a digit-level PISO GNB multiplier architecture is proposed. This architecture which uses the following formulation is depicted in Fig. 2.2.

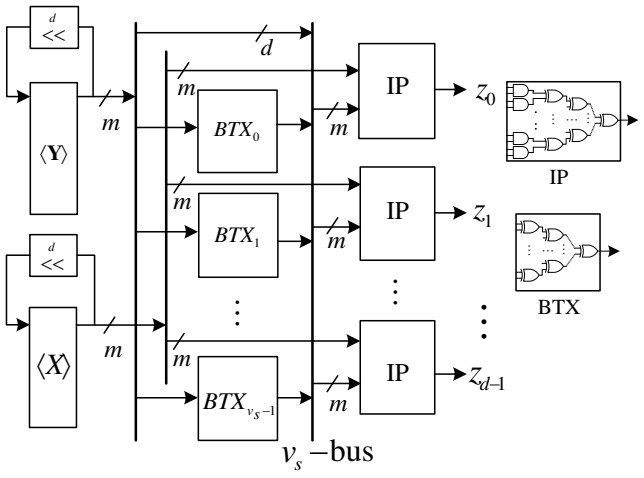


Figure 2.2: The architecture of the digit-level PISO GNB multiplier [5].

Lemma 2.3. [5] *Let us denote* $z_l = \underline{x}\mathbf{M}^{(l)}\underline{y}^{tr}$, *where* $\mathbf{M}^{(l)}$ *denotes* l*-fold right and down circular shift of multiplication matrix* $\mathbf{M}$. *Then, for a digit level architecture one needs to implement all entries of* d *vectors of*

$$\mathbf{v}^{(l)} = [v_0^{(l)}, v_1^{(l)}, \cdots, v_{m-1}^{(l)}]^{tr} = \mathbf{M}^{(l)}\underline{y}^{tr},\ 0 \le l \le d-1, \tag{2.14}$$

Then, by $\underline{y} = \underline{b}$

$$z_l = \underline{x}\mathbf{v}^{(l)} = \sum_{i=0}^{m-1} x_i v_i^{(l)}. \tag{2.15}$$

for $\underline{x} = \underline{a}$ *and* $\underline{c_l} = \underline{z_l}$. *Consecutive* d *coordinates of* $C = AB$ *can be obtained from (2.14) and (2.15) by* d*-fold left cyclic shift of* $\underline{x}$ *and* $\underline{y}$. *This multiplier requires* $q = \lceil \frac{m}{d} \rceil$, $1 \le q \le m$, $1 \le d \le m$, *clock cycles to generate all the* m *coordinates of the* $C = AB$.

The architecture which realizes (2.14) and (2.15) is shown in Fig. 2.2. A d-fold left cyclic shift is denoted by "$\overset{d}{\ll}$" in this figure.

It is noted that the presented $\mathbf{R}$ matrix in (2.7) can be easily obtained from the $\mathbf{M}$. Specifically, the (i,j)-th, $1 \le i \le m-1$,$1 \le j \le T$, entry of the matrix $\mathbf{R}$, i.e., $R(i,j)$, $0 \le R(i,j) \le m-1$ contains the column index of the non-zero entries in row i of $\mathbf{M}$. If the number of 1s in row i of $\mathbf{M}$ is T, then all $R(i,j)$, $1 \le j \le T$, contain an integer in $[0,\ m-1]$. Otherwise, rows of $\mathbf{R}$ with even number of entries should be initialized with a constant value [5]. Therefore, one can obtain

$$c_l = a_l b_{l+1 \bmod m} + \sum_{i=1}^{m-1} a_{l+i \bmod m} \left(\sum_{j=1}^{T} b_{l+R(i,j) \bmod m} \right), \tag{2.16}$$

and implement d copies of c_l in hardware to achieve a digit-level architecture for $0 \leq l \leq d-1$.

2.2.4.5 Digit-level PIPO GNB Multiplier

In [5] and [6] a digit-level GNB multiplier with parallel output (DL-PIPO) is proposed. It requires q, $1 \leq q \leq m$, clock cycles to generate all m coordinates of $C = AB$ simultaneously at the end of the final clock cycle. The original multiplier structure of DL-PIPO is shown in Fig. 2.3. Let $\langle X \rangle$ and $\langle Y \rangle$ be the input registers of this multiplier. Then, it implements [5]

$$J(X,\ Y) = \sum_{k=0}^{m-1} x_{m-k} s_0^{'}(k,\ Y) \beta^{2^i}, \tag{2.17}$$

where

$$s_0^{'}(k,\ Y) = \sum_{i \in R_k} y_{i-k}, \tag{2.18}$$

and R_k is a set containing the locations of non-zero entries of row $2k$, $0 \leq 2k \leq m-1$, of the multiplication matrix $\mathbf{M} = \mathbf{M^{(0)}}$ defined in (2.4). Based on the properties of $\mathbf{M}$ for GNB, one can find $s_0^{'}(0, Y) = y_1$ and $s_0^{'}(k, Y) = s_0^{'}(m-k, Y)$, $1 \leq k \leq \frac{m-1}{2}$ [5]. Also, it is shown in [44] and [5] that the number of elements in R_k is even and less than or equal to T, i.e., $|R_k| \leq T$. The J block in Fig. 2.3 performs (2.17) using m AND gates. For the multiplication operation, the registers $\langle X \rangle$ and $\langle Y \rangle$ of this figure are initially loaded by the coordinates of A and B, respectively. Also, the output register $\langle Z \rangle$ should be cleared before starting the multiplication operation. Then, after q clock cycles, the output register $\langle Z \rangle$ contains the coordinates of $C = AB$. In the following section, we modify this multiplier to reduce the number of XOR gates.

2.3 Elliptic Curve Cryptography

To date, several forms of elliptic curves over finite fields of characteristic two have been considered for hardware implementation of such cryptosystems in the literature; see for example, [20], [21], [10], [6], [22], [23], [24], [25], [26], and [27]. They cover

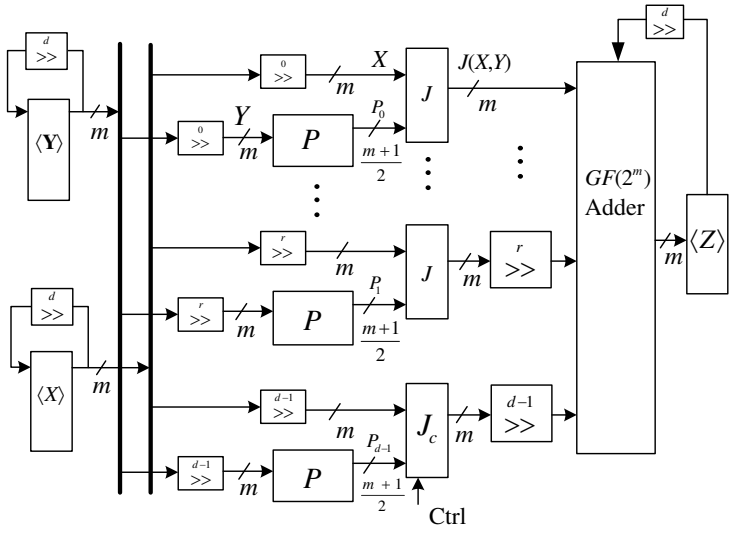


Figure 2.3: The architecture of Digit-level PIPO GNB multiplier proposed in [5], [6], where the i-fold right cyclic shift is denoted by $\overset{i}{\gg}$ and r is a number $0 \leq r \leq d-1$ such that $m = qd - r$.

a wide variety of cases regarding different basis representations (e.g., polynomial basis and normal basis), different coordinate systems (e.g., affine, projective, mixed, etc.), and different curve forms (e.g., generic and Koblitz). In these implementations, various hardware platforms such as field-programmable gate array (FPGA) and application-specific integrated circuit (ASIC) have been utilized. For different target applications, efficient implementations of ECC on these platforms with a balance between complexity of computations and availability of the resources are crucial to provide highly efficient cryptographic systems.

Binary Edwards curves have been introduced recently by Bernstein, Lange, and Farashahi in [1]. They showed that all generic elliptic curves over binary fields can be written in Edwards form to obtain efficient complete and unified addition formulas which work for all pairs of inputs. In [2], a generalized form of binary Hessian curves is proposed which has similar characteristics to the binary Edwards curves. Both of these curves offer unified and complete formulas for point operations which provides resistance against side-channel attacks (SCAs). Despite the efficiency of binary Edwards and generalized Hessian curves, a limited number of articles in the literature such as [52], [53], and [54] have investigated their implementations. In [52], an ASIC implementation of point multiplication on a special case of binary Edwards curves has been presented addressing energy consumption and simple power analysis attacks over $GF(2^m)$ using polynomial basis representation. A SCA resistance evaluation of

binary Edwards curves has been discussed in [53] employing unified addition formula for doubling. The work presented in [54] mainly focuses on software implementation of point multiplication on these curves employing different curve parameters.

2.3.1 Elliptic Curve Arithmetic

In this thesis we mainly focus on binary fields and limit definitions of elliptic curves on $GF(2^m)$.

Let $E_{W,a,b}$ be a non-supersingular binary generic elliptic curve (short Weierstrass) defined as

$$E_{W,a,b}: \; y^2 + xy = x^3 + ax^2 + b, \tag{2.19}$$

where $a, b \in GF(2^m)$, and $b \neq 0$. A set of points (x, y) and a special point at infinity $\mathcal{O}$ (group identity) form a finite Abelian group under a defined addition operation that satisfy (2.19) and the so called chord-and-tangent rule (as shown in Fig. 2.4) is used to define the group operation [11]. For all $P \in E_{W,a,b}(GF(2^m))$, $P+\mathcal{O} = \mathcal{O}+P = P$. The negative of point $P = (x, y)$ is -$P = (x, x + y)$, where $(x, y) + (x, x + y) = \mathcal{O}$.

Then, for two points P_1,$P_2 \in E_{W,a,b}(GF(2^m))$, the third point $P_3 = P_1 + P_2 \in E_{W,a,b}(GF(2^m))$ exist and can be produced using arithmetic operations in $GF(2^m)$ which is called point addition. Also, for point $P = (x_1, y_1)$ and $P \neq -P$ the point doubling is $P_4 = 2P = (x_4, y_4)$.

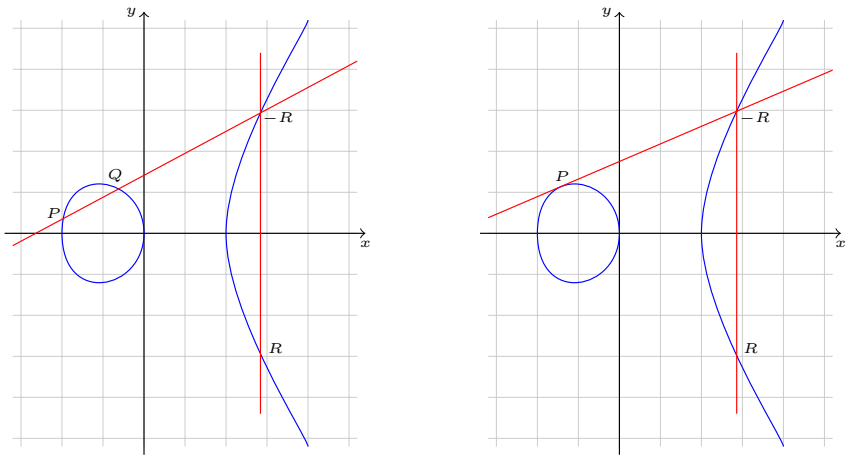
Elliptic curve point multiplication is defined over the Abelian group and it is

$$Q = kP = \underbrace{P + P + \cdots + P}_{k}, \tag{2.20}$$

where P and Q are two points on $E_{W,a,b}$ and $k > 1$ is an integer. The point P is called the base point and Q is the result point.

Definition 2.5. Given the cyclic additive group generated by P on $E_{W,a,b}(GF(2^m)$, the order of point P, ord(P), is the smallest integer r, for which $rP = \mathcal{O}$. Then, the integer k is bounded as $1 < k \leq \text{ord}(P) - 1$.

Although the point multiplication of the form (2.20) is the most common operation in elliptic curve cryptosystems, but in some applications (such as digital signature) a double point multiplication with the form of $mP + nQ$ is required to be performed, where $P, Q \in E_{W,a,b}(GF(2^m))$ are points of order r and $1 \leq m, n \leq r - 1$.



(a) Point addition (b) Point doubling

Figure 2.4: Group law on Elliptic curve over $\mathbb{R}$.

Definition 2.6. Given two points P and Q, where $Q = kP$, it is computationally infeasible to obtain k which is known as elliptic curve discrete logarithm problem (ECDLP). The ECDLP currently has exponential complexity and has no polynomial-time solutions (without considering quantum computers).

Point addition in affine coordinates $P_3 = (x_3, y_3) = P_1 + P_2$, where $P_1 \neq P_2$ is given by [28]:

$$\begin{cases} x_3 &= \lambda^2 + \lambda + x_1 + x_2 + a, \ \lambda = \frac{y_2 - y_1}{x_2 + x_1}, \\ y_3 &= \lambda\,(x_1 + x_3) + x_3 + y_1, \end{cases} \tag{2.21}$$

where it costs $\mathbf{I} + 2\mathbf{M} + \mathbf{S} + 8\mathbf{A}$. Point doubling is $P_4 = (x_4, y_4) = 2P_1$ as given by

$$\begin{cases} x_4 &= x_1^2 + \frac{b}{x_1^2} \\ y_4 &= x_1^2 + \left(x_1 + \frac{y_1}{x_1}\right) x_4 + x_4, \end{cases} \tag{2.22}$$

and it costs $\mathbf{I} + 2\mathbf{M} + \mathbf{S} + 4\mathbf{A}$. As computing the inversion is costly in the finite fields and as a result, some alternative approaches have been considered.

2.3.2 Inversion free Coordinates

Inversion is known as an expensive operation in finite fields. Therefore, instead of having point coordinates represented in affine coordinate, it is efficient to define them in projective coordinates. In the following, different types of projective coordinates are presented.

2.3.2.1 Standard Projective Coordinates

In standard projective coordinates, a point is represented with the triple (X, Y, Z) to represent $(X/Z, Y/Z)$ in affine with $Z \neq 0$ and $\mathcal{O} = (0, 1, 0)$. Then, the curve equation will be

$$Y^2Z + XYZ = X^3 + aX^2Z + bZ^3,$$

where the cost of point addition and doubling is $16\mathbf{M} + 2\mathbf{S} + 6\mathbf{A}$ and $8\mathbf{M} + 4\mathbf{S} + 5\mathbf{A}$, respectively.

2.3.2.2 Lopez-Dahap Projective Coordinates

For Lopez-Dahab coordinates, [3] the triple (X, Y, Z) is used to represent $(X/Z, Y/Z^2)$ in affine when $Z \neq 0$ and $\mathcal{O} = (1, 0, 0)$. The curve equation is

$$Y^2 + XYZ = X^3Z + aX^2Z^2 + bZ^4,$$

where the cost of point addition and doubling is $13\mathbf{M} + 4\mathbf{S} + 9\mathbf{A}$ and $5\mathbf{M} + 4\mathbf{S} + 5\mathbf{A}$, respectively. In Lopez-Dahap coordinates when one of the points represented in affine the cost of mixed projective point addition, i.e., $(X_3, Y_3, Z_3) = (X_1, Y_1, Z_1) + (x_2, y_2)$, reduces to $9\mathbf{M} + 5\mathbf{S} + 9\mathbf{A}$ [55].

2.3.2.3 Jacobian Projective Coordinates

In Jacobian projective coordinates, the triple (X, Y, Z) corresponds to the affine point $(X/Z^2, Y/Z^3)$ with the curve equation as

$$Y^2 + XYZ = X^3 + aX^2Z^2 + bZ^6,$$

where the costs of mixed point addition and doubling are $10\mathbf{M}+3\mathbf{S}+7\mathbf{A}$ and $5\mathbf{M}+5\mathbf{S}$, respectively.

Algorithm 2.2 Left-to-right Double-and-add point multiplication algorithm [11]

Inputs: An integer $k > 1$, $k := (k_{l-1} \cdots k_1 k_0)_2$, and $P = (x, y) \in E(GF(2^m))$
Output: $Q = kP \in E(GF(2^m))$
Initialize: $Q = P$
For $i := l - 2$ down to 0 do
    $Q = 2Q$
    if $k_i = 1$ then
      $Q = Q + P$
      end if
end for
return Q

2.3.3 Point Multiplication

The elliptic curve point multiplication is defined in the Abelian group as $Q = k \cdot P = P + P + \cdots + P$, ($k$ times), where k is a positive integer, and Q and P are two points on the elliptic curve $Q, P \in E(GF(2^m))$ [3]. The efficiency of point multiplication depends on finding the minimum number of steps to reach kP from a given point P. In the following two mostly used algorithm for point multiplication is presented.

2.3.3.1 Double-And-Add Point Multiplication

The simplest method to perform point multiplication is the double-and-add method as shown in Algorithm 2.2. As one can see, the scalar k is given in binary form, i.e., $k = \sum_{i=0}^{l-1} k_i 2^i$ and the algorithm iterates through each bit of k. For each iteration a point doubling is performed and when k is one, a point addition is also performed. Clearly, the computational cost of the double-and-add method depends on the number of ones in the binary expansion of k, i.e., $H(k)$ which is the Hamming weight of k. Therefore, this method requires $l - 1$ point doublings and $H(k) - 1$ ($H(k) \approx l/2$ on average) point additions. As the $H(k)$ determines the performance of double-and-add point multiplication algorithm, reducing it is always desired. A Non-Adjacent Form (NAF) representation of k is used to reduce $H(k)$. In this representation two consecutive digits are never nonzero, i.e., $k_i k_{i+1} = 0$ and $k_i \in \{0, \pm 1\}$ for all i. The NAF method reduces the Hamming weight to $H(k) \approx l/3$.

The double-and-add point multiplication is not secure against side channel attacks and an attacker can reveal k by tracing the power consumption for doubling and addition in each iteration. This method is suitable for the applications where point

addition and doubling have equal cost of computation, for example in binary Edwards [1] and generalized Hessian curves [2].

2.3.3.2 Montgomery Point Multiplication

Lopez and Dahab [3] generalized the Montgomery's idea [13] to binary generic curves (2.19) and obtained a very efficient algorithm for point multiplication. This method is known as Montgomery point multiplication or Montgomery's ladder and is widely used in the literature. It relies on the fact that the y-coordinate is not required during point multiplication because it can be recovered at the end. Then, the x-coordinate of point addition can be obtained as $P_3 = P_1 + P_2$ from

Algorithm 2.3 Lopez-Dahab Scalar Multiplication [12]

Inputs: An integer $k > 1$, $k := (k_{l-1} \cdots k_1 k_0)_2$, and $P = (x, y) \in E$
Output: $Q = kP$
Step 1: $X_1 := x$, $Z_1 := 1$, $X_2 := x^4 + b$, $Z_2 := x^2$
Step 2: For $i := l - 2$ down to 0
  if $k_i = 1$ then
Step 3: $(X_1, Z_1) =$ADD(X_1, Z_1, X_2, Z_2), $(X_2, Z_2) =$DBL(X_2, Z_2)
  else
Step 4: $(X_2, Z_2) =$ADD(X_1, Z_1, X_2, Z_2), $(X_1, Z_1) =$DBL(X_1, Z_1)
Step 5: return $Q = Mxy(X_1, Z_1, X_2, Z_2)$

$$Z_3 = (X_1 \cdot Z_2 + X_2 \cdot Z_1)^2, \quad X_3 = x \cdot Z_3 + (X_1 \cdot Z_2) \cdot (X_2 \cdot Z_1), \tag{2.23}$$

with the cost of $4\mathbf{M} + \mathbf{S} + 2\mathbf{A}$ and the x-coordinate of point doubling, $P_4 = 2P_1$ from

$$X_4 = X_1^4 + b \cdot Z_1^4, \quad Z_4 = Z_1^2 \cdot X_1^2 \tag{2.24}$$

with the cost of $2\mathbf{M} + 3\mathbf{S} + \mathbf{A}$. Then, the y-coordinate is recovered with the cost of $\mathbf{I} + 10\mathbf{M} + \mathbf{S} + 6\mathbf{A}$ [3]. In point multiplication using Montgomery algorithm in each step point addition and point doubling should be performed. Then, due to its uniform structure it reveals no information to distinguish it performs point addition point doubling an hence is resistive to simple power analysis attack. It also provides fast computations in comparison to the case where explicit addition and doubling formulation are employed. The cost of combined point addition and doubling based

on the x-coordinates only is $6\mathbf{M} + 4\mathbf{S} + 3\mathbf{A}$. Algorithm 2.3, presents Montgomery point multiplication for a given point $P \in E_{W,a,b}(GF(2^m))$. Also, Mxy, converts the Lopez-Dahab coordinates to affine ones and it is the only operation in this algorithm which requires inversion. The Montgomery point multiplication is fast, uniform, and secure against side channel attacks such as simple power analysis attacks. For detail information about elliptic curve cryptography and its mathematical computations one can refer to [11].

In the next chapter, we will present low-complexity hardware architectures for digit-level GNB multipliers.

Chapter 3

Low-Complexity Architectures for Digit-level and Bit-parallel GNB Multipliers over $GF(2^m)$

OUR objective in this chapter is to reduce the area complexity of digit-level GNB multiplier architectures presented in the previous chapter. The multiplication of two field elements in binary field of characteristic two, i.e., $GF(2^m)$, is more complicated than the other operations (e.g., addition and squaring) and plays an important role in determining the efficiency of cryptographic systems. Massey and Omura (MO) [35] invented a bit-level, parallel-in serial-out $GF(2^m)$ normal basis multiplier. Such a bit-level multiplier is slow as it generates the results of multiplication after m clock cycles. The fastest type of multipliers is the bit-parallel one whose results are available after the propagation delay through the gates in one clock cycle. We note that for type 2 GNB (which is type 2 optimal normal basis), there are several efficient multipliers available in the literature. For instance, in [56], Sunar and Koç proposed a bit-parallel multiplier based on a permuted normal basis. An efficient and systolic type of their multiplier has been proposed later by Kwon [57] for type 2 GNB which is highly regular. Also, sub-quadratic style multipliers have been proposed in [58], [59], and [60] which require smaller area but higher delays. A digit-level version of MO multiplier [35] is investigated for FPGA implementation of ECC in [10]. Also, Kwon *et al.* [44] proposed an improved digit-level GNB multiplier which has been employed in [6] for FPGA implementation of ECC over $GF(2^{163})$. In order to satisfy high speed and low complexity requirements of an ECC crypto-processor, one needs to design an efficient architecture for finite field multiplication using normal basis [10].

The contributions of this chapter can be summarized as follows. The result presented in this chapter can be found in [9] and partly in [61].

- We present a low complexity architecture for digit-level parallel in parallel out (DL-PIPO) GNB multiplier and propose a common subexpression elimination algorithm. We also reduce the complexity of digit-level parallel in serial out (DL-PISO) architecture presented in the previous chapter.
- We propose a new formulation and an improved architecture for digit-level serial in parallel out (DL-SIPO) GNB multiplier architecture and derive its time and area complexities. It is noted that the proposed architecture requires smaller area in comparison to the leading ones in the literature.
- We simulate the performance of the complexity reduction algorithm and for different digit sizes for the proposed digit-level multiplier architectures.
- A low complexity bit-parallel architecture has been obtained by extending the presented DL-PISO multiplier architecture and its time and area complexities compared with the counterparts in the literature.
- Finally, our proposed multiplier architectures are implemented on the Xilinx® Virtex™-4 FPGA and synthesized using 65-nm CMOS library (ASIC) technology for different digit sizes. The timing and required area is also reported.

The rest of this chapter is organized as follows. In Section 3.1, a low complexity digit-level parallel in parallel out multiplier architecture is presented. In Section 3.2, a new architecture for digit-level serial in parallel out multiplier proposed and its time and area complexities derived. In Section 3.3, the presented architecture for digit-level parallel in serial out architecture in the previous chapter is improved. In Section 3.4, a low-complexity bit-parallel architecture is proposed and its time and area complexities compared with its counterparts. In Section 3.5, the proposed multiplier architectures are implemented on FPGA and ASIC and the results are reported for different digit sizes. Finally, we conclude this chapter in Section 3.6.

3.1 An Improved Architecture for Digit-level PIPO GNB Multiplier

In this section, we propose an improved architecture for the digit-level PIPO multiplier presented in the previous chapter. The number of XOR gates of the DL-PIPO

multiplier can be reduced by reusing the common terms appeared at the outputs of the P blocks. The DL-PIPO GNB multiplier architecture, has several P blocks shown as p_0 to p_{d-1} in Fig. 2.3. As shown in this figure, P blocks use the shifted combination of the input operand B (preloaded in register $\langle Y \rangle$). Therefore, we first determine these combinations and after these combinations are computed, we use their results in different computations to optimize the area complexity by reducing the number of signals and consequently number of XOR gates. We propose a method to combine the computations of the P blocks into a ρ block as illustrated in the architecture of Fig. 3.1. As seen in this figure, the number of outputs of an unoptimized P block in this figure is $\frac{m+1}{2}$. These are based on the following signals [5]

$$\begin{aligned} P_k(Y) = (y_{1-k},\ & s_0^{'}(1, Y \ll k),\ s_0^{'}(2, Y \ll k), \cdots \\ & , \cdots , s_0^{'}(\tfrac{m-1}{2},\ Y \ll k)),\ 0 \leq k \leq d-1, \end{aligned} \tag{3.1}$$

for the P block that generates $P_k(Y)$. All signals in (3.1) are used to build the block ρ in Fig. 3.1. As shown in this figure, y_{1-k}s are removed from the block ρ. To reduce the complexity of the ρ block in Fig. 3.1, we divide the ρ block in two blocks ρ_1 and ρ_2, where ρ_1 includes all common pairs used to generate all signals in (3.1). In the following we explain the procedure to build the ρ block and propose a complexity reduction algorithm to obtain the optimized blocks of ρ_1 and ρ_2 having the time complexity to be the same as the original block ρ, i.e., the addition of gate delays of the two blocks ρ_1 and ρ_2.

Constructing the ρ Block

1. Corresponding to the output signals of the P block in Fig. 2.3, an $\frac{m-1}{2} \times T$ matrix denoted by $\mu = [\mu_k]_{k=1}^{\frac{m-1}{2}}$ is constructed, where μ_k is the row k, $1 \leq k \leq \frac{m-1}{2}$ of the matrix μ. The entries of μ_k are at most T integers in the range of $[0, m-1]$ and can be found from (2.18) which can be written as $s_0^{'}(k,\ Y) = \sum_{j \in \mu_k} y_j$, $1 \leq k \leq \frac{m-1}{2}$.

2. Based on the matrix μ and the given digit-size d, a matrix denoted by ρ is

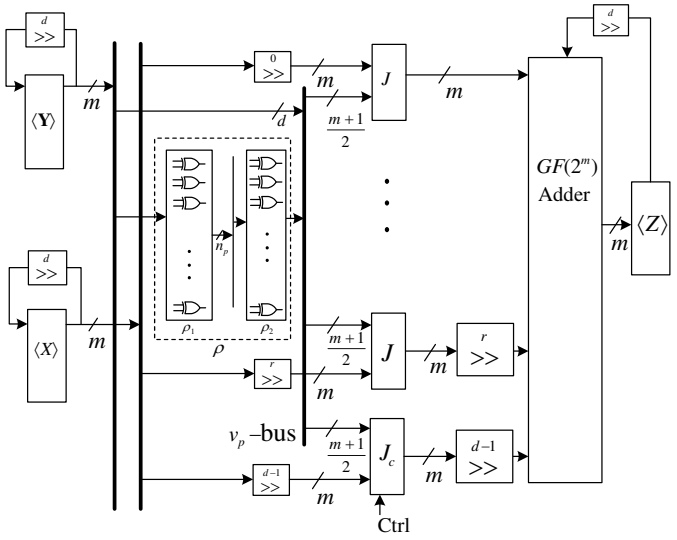


Figure 3.1: The proposed improved architecture for DL-PIPO GNB multiplier

obtained by appending the $d-1$ matrices of $\mu - [i] \mod m$ to μ as follows:

$$\rho = \begin{pmatrix} \mu & & \\ \mu & - & [1] \bmod m \\ \mu & - & [2] \bmod m \\ \vdots & \vdots & \vdots \\ \mu & - & [d-1] \bmod m \end{pmatrix}_{(d\times\frac{m-1}{2})\times T}, \tag{3.2}$$

where $[i]$, $1 \le i \le d$, denotes an $\frac{m-1}{2} \times T$ matrix whose all entries are i.

3. Let ρ_i be a set which contains the entries in row i of the matrix ρ. Then, all signals

$$s_j = \sum_{j\in\rho_i} y_j, \ 1 \le j \le d\frac{(m-1)}{2} \tag{3.3}$$

should be implemented by the block ρ shown in Figure 3.1.

Complexity Reduction Algorithm

We want to find the common addition pairs to realize (3.3) with the least number of XOR gates without changing the delay of the modified multiplier as compared with the original one.

1. Generate a pairset to form all pairs that should be implemented in the block ρ_1.

2. Initialize the pairset in Step 1 by all pairs with only two entries in the rows of the matrix ρ.

3. Based on the numbers of times that these pairs are repeated, update the ρ matrix by removing the pairs obtained in Step 2. Then, go to Step 1.

4. Repeat the above iteration until there is no rows with more than two entries in ρ matrix.

5. Generate the the ρ_1 inside the ρ block based on the common pairs stored in the pairset.

6. Reuse the output of the block ρ_1 and generate all signals from the block ρ_2 in Figure 3.1.

It is noted that unlike the complexity reduction schemes available in the literature, see for example [62], the proposed algorithm does not increase the gate delay of the proposed architecture as compared to the original one.

3.1.1 Complexities

In this subsection, the complexity of the proposed digit-level PIPO multiplier is given in terms of gate counts and critical-path delay.

Proposition 3.1. *The proposed improved architecture for DL-PIPO type T GNB multiplier over $GF(2^m)$ requires dm AND gates, 3 m-bit registers, and $n_p + v_p(\frac{T}{2} - 1) + dm$ XOR gates, where n_p, $n_p \leqslant \min\left\{\frac{v_pT}{2}, \binom{m}{2}\right\}$ is the number of XOR gates (pairs) required to construct the block ρ_1 in the proposed structure and the number of rows inside the matrix which builds ρ is $v_p = d \times \frac{m-1}{2}$. Also its critical path delay is*

$$T_{DL-PIPO} = T_A + (\lceil \log_2 T \rceil + \lceil \log_2(d+1) \rceil) T_X, \tag{3.4}$$

where T_A and T_X are the time delay of a two-input AND gate and an XOR gate, respectively.

Proof. The number of rows in the matrix which builds ρ is $v_p = d \times \frac{m-1}{2}$ and each row consists of at most $\frac{T}{2}$ pairs. Then, the number of pairs inside the ρ_1 block will be less than or equal to $v_p \times \frac{T}{2}$. In the case where $d = m$ (bit-parallel), one can find the upper bound of n_p as $\binom{m}{2} = \frac{m(m-1)}{2}$. Therefore, for the digit-level structure, i.e., $1 < d < m$, the upper bound for n_p is less than the minimum of $\left\{\frac{v_pT}{2}, \frac{m(m-1)}{2}\right\}$.

Moreover, at most $v_p \times (\frac{T}{2} - 1)$ XOR gates in the ρ_2 block are required to build all the signals of the ρ block. To construct the $GF(2^m)$ adders, one needs dm XOR gates. As a result, the complexity of the proposed multiplier is $n_p + v_p(\frac{T}{2} - 1) + dm$ XOR gates, dm AND gates, and $3m$ 1-bit registers.

The critical-path delay of the proposed architecture can be obtained by adding the delays of the three blocks of ρ_1, ρ_2, J, and the $GF(2^m)$ adder which are T_X, $\lceil \log_2 \frac{T}{2} \rceil T_X$, T_A, and $\lceil \log_2(d+1) \rceil T_X$, respectively. This results in the total delay of $T_X + \lceil \log_2 \frac{T}{2} \rceil T_X + T_A + \lceil \log_2(d+1) \rceil T_X = T_A + (\lceil \log_2 T \rceil + \lceil \log_2(d+1) \rceil) T_X$, which completes the proof.

□

In the following section, we present an illustrative example for the proposed complexity reduction algorithm.

3.1.2 An Example over $GF(2^7)$

To better understand the complexity reduction algorithm, we illustrate an example for the proposed algorithm for type 4 digit-level multiplier over $GF(2^7)$ when the digit-size is $d = m = 7$. The matrix $\mathbf{M}$ for type 4 GNB over $GF(2^7)$ is

$$\mathbf{M} = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 \end{pmatrix}_{7\times 7}.$$

The matrix μ can be generated according to the output of the P blocks in Fig. 2.3 as $s_0'(1, Y) = y_{1-1} + y_{3-1} + y_{4-1} + y_{5-1} = y_0 + y_2 + y_3 + y_4$, $s_0'(2, Y) = y_{2-2} + y_{6-2} = y_0 + y_4$, and $s_0'(3, Y) = y_{1-3} + y_{4-3} + y_{5-3} + y_{6-3} = y_5 + y_1 + y_2 + y_3$. Then μ can be written as

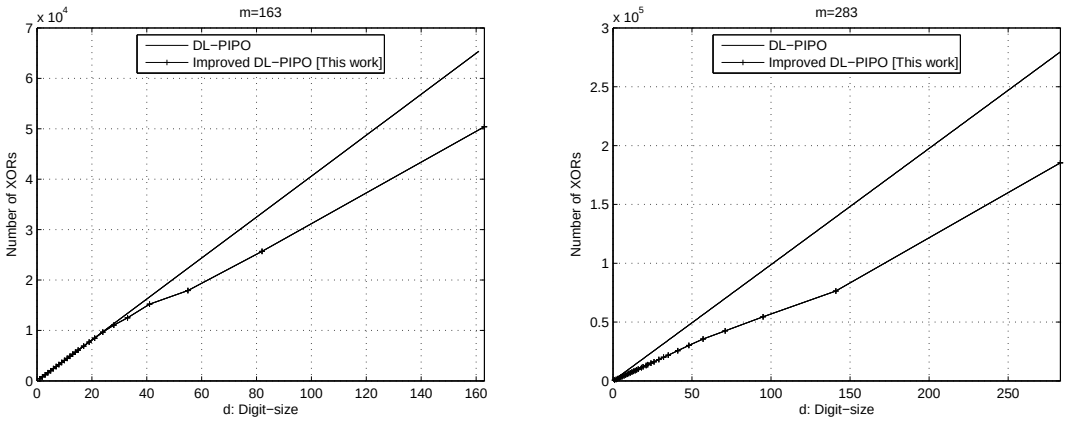
$$\mu = \begin{pmatrix} 0 & 2 & 3 & 4 \\ 0 & 4 & - & - \\ 5 & 1 & 2 & 3 \end{pmatrix}_{3\times 4}.$$

$$\rho=\begin{pmatrix}0&2&3&4\\0&4&-&-\\5&1&2&3\\6&1&2&3\\6&3&-&-\\4&0&1&2\\5&0&1&2\\5&2&-&-\\3&6&0&1\\4&6&0&1\\4&1&-&-\\2&5&6&0\\3&5&6&0\\3&0&-&-\\1&4&5&6\\2&4&5&6\\2&6&-&-\\0&3&4&5\\1&3&4&5\\1&5&-&-\\6&2&3&4\end{pmatrix}_{21\times4}\quad \text{Pairset1}=\begin{cases}y_{04}\\y_{63}\\y_{52}\\y_{41}\\y_{30}\\y_{26}\\y_{15}\end{cases}\quad \rho^{(1)}=\begin{pmatrix}0&2&3&4\\5&1&2&3\\6&1&2&3\\4&0&1&2\\5&0&1&2\\3&6&0&1\\4&6&0&1\\2&5&6&0\\3&5&6&0\\1&4&5&6\\2&4&5&6\\0&3&4&5\\1&3&4&5\\6&2&3&4\end{pmatrix}\quad \rho^{(2)}=\begin{pmatrix}&2&3&\\&1&&3\\&\underline{1}&\underline{2}&\\&&\underline{1}&\underline{2}\\&\underline{0}&\underline{1}&\\&&\underline{0}&\underline{1}\\&\underline{6}&\underline{0}&\\&&\underline{6}&\underline{0}\\&5&&0\\&&5&6\\&4&&6\\&&4&5\\&3&&5\\&2&&4\end{pmatrix}\quad \text{Pairset2}=\begin{cases}y_{23}\\y_{13}\\y_{12}\\y_{01}\\y_{60}\\y_{50}\\y_{56}\\y_{46}\\y_{45}\\y_{35}\\y_{24}\end{cases}$$

Based on the digit-size $d = 7$ and the matrix $\mu_{(3\times4)}$, the matrix $\rho_{(21\times4)}$ can be generated corresponding the complexity reduction algorithm. One can obtain from the matrix $\rho_{(21\times4)}$ in which 7 rows of the matrix have just two entries. Therefore, the pairs corresponding to these rows should be implemented as collected in the pairset1. The matrix ρ is updated to $\rho^{(1)}$ by deleting all the two entries mentioned in the pairset1. Then the elements of the pairset1 should be searched in $\rho^{(1)}$ and all common pairs are removed and $\rho^{(1)}$ is updated to $\rho^{(2)}$. This iteration is repeated until there is no rows with more than two entries. As a result, all the remaining pairs as mentioned in the pairset2 should be implemented and repeated pairs (which are underlined in the updated $\rho^{(2)}$ matrix) are removed. The union of pairset1 and pairset2 includes the total of 18 pairs that should be implemented for the block ρ_1 as follows:

$$\begin{aligned}\text{pairset}=\{&y_{04},y_{63},y_{52},y_{41},y_{30},y_{26},y_{15},y_{23},y_{13},y_{12},y_{01},y_{60},\\&y_{50},y_{56},y_{46},y_{45},y_{35},y_{24}\},\end{aligned}$$

where $y_{ij} = y_i + y_j$. In addition to the implementation of the ρ block which requires 18 XOR gates, one need $d\frac{m-1}{2} - d = 14$ (as, $d = m$) extra XOR gates for the block ρ_2 to construct its outputs. Therefore, the total number of XOR gates required to



(a) (b)

Figure 3.2: Comparison between the number of XOR gates required in the DL-PIPO and the improved DL-PIPO for (a): $m = 163$ ($T = 4$), (b): $m = 283$ ($T = 6$).

implement the ρ block will be $18 + 14 = 32$, whereas the unoptimized P blocks need 49 XOR gates and the scheme proposed in [5] requires 35 XOR gates.

It is noted that the other complexity reduction algorithms available in the literature may result in fewer number of gates at the expense of more delay. To compare our complexity reduction algorithm with the one proposed in [62], we have applied the complexity reduction algorithm proposed in [62] for the block ρ of this example. It decreases the number of XORs to 23 with the increase of critical path delay to $8T_X$ (eight level of XOR gates). Note that our scheme for this block results in the complexity of 32 XOR gates with the same critical path delay as the original one, i.e., $2T_X$.

3.1.3 Simulation Results for the DL-PIPO GNB Multiplier over $GF(2^{163})$ and $GF(2^{283})$

To evaluate the efficiency of proposed complexity reduction algorithm, a MATLAB code is written to generate common pairs and signals used in the blocks ρ_1 and ρ_2 of Fig. 3.1. It is noted that for type 2 GNB which is a type 2 optimal normal basis over $GF(2^m)$, there is no common terms to be reused in the block ρ. Therefore, the algorithm presented here cannot reduce the number of XOR gates for $T = 2$. The simulation results of the algorithm for the improved DL-PIPO GNB multipliers over $GF(2^{163})$ and $GF(2^{283})$ are obtained and plotted in Fig. 3.2a and Fig. 3.2b, respectively. In these figures, we plot the number of required XOR gates versus the

digit size for the fields $GF(2^{163})$ ($T = 4$) and $GF(2^{283})$ ($T = 6$) recommended by NIST for ECDSA [19] as compared to ones of the original DL-PIPO architecture. For a given number of clock cycle, q, $1 \leq q \leq m$, the least value of digit sizes in the form of $d = \left\lceil \frac{m}{q} \right\rceil$, $1 \leq d \leq m$, is implemented so that the area complexity is optimized for both multipliers.

From Fig. 3.2a and 3.2b, one can see that as the digit size increases, more common pairs will be found. As an example, in Fig. 3.2a for the digit size $d = m = 163$, the total number of XOR gates required in the original DL-PIPO is 66178 gates whereas, the improved one, requires 50400 XOR gates for $GF(2^{163})$. It means that the complexity of the proposed improved DL-PIPO is about 24% less than the original multiplier. More reduction can be found in Fig. 3.2b for the $GF(2^{283})$ with $d = m = 283$. As seen the number of XOR gates needed by the original DL-PIPO is 279,604, whereas the proposed DL-PIPO requires 185,375 XOR gates which is about 34% less than that of the original multiplier. The exact values of n_p, i.e., the number of pairs to construct ρ are given in Table 3.1 which are obtained by simulations.

Table 3.1: Comparison of number of XOR gates between bit-parallel GNB multipliers for $GF(2^{163})$ and $GF(2^{283})$.

m	T	n_p	Original DL-PIPO	This work
163	4	10,791	66,178	50,400
283	6	25,763	279,604	185,375

3.2 New Architecture for Digit-Level SIPO GNB Multiplier

In a digit-level SIPO multiplier, the bits of an operand are grouped into digits and in each clock cycle one digit is processed. We extend the architecture of the LSB-first bit-level GNB multiplier architecture presented in Chapter 2 and propose a low-complexity LSD-first digit-level SIPO GNB multiplier architecture. In the following, we present formulation, architecture, and complexity of the proposed multiplier architecture.

3.2.1 Formulation

Let us assume $A = \sum_{i=0}^{m-1} a_i \beta^{2^i} = (a_0, a_1, \cdots, a_{m-1})$, then one can group the bits into $q = \lceil \frac{m}{d} \rceil$ digits denoted by A_i, $0 \leq i \leq q-1$ as $(a_0, \cdots, a_{d-1})$ for the first digit followed by $(a_d, \cdots, a_{2d-1})$ for the second digit and finally $(a_{d(q-1)}, \cdots, a_{m-1})$ for the qth digit where d, $2 \leq d \leq m-1$, is denoted as the number of bits in each digit. Note that if the last digit does not have d bits, it will be appended by zeros as its most significant bit ends. Then, each digit can be represented as $A_i = (a_{id}, a_{id+1}, \cdots a_{id+d-2}, a_{id+d-1}) = \sum_{j=0}^{d-1} a_{j+id} \beta^{2^j}$, $A_i \in GF(2^m)$ with respect to the GNB and thus operand A can be written as

$$A = \sum_{i=0}^{q-1} A_i^{2^{id}} = (A_0, A_1, \cdots, A_{q-1}).$$

Therefore, one can write their product $AB = C \in GF(2^m)$ as

$$\begin{aligned} C &= AB \\ &= \sum_{i=0}^{q-1} A_i^{2^{id}} \cdot B = \sum_{i=0}^{q-1} \left(A_i \cdot B^{2^{-id}} \right)^{2^{id}} = \sum_{i=0}^{q-1} \left(C^{(i)} \right)^{2^{id}}, \end{aligned} \tag{3.5}$$

where

$$C^{(i)} = A_i B^{2^{-id}}. \tag{3.6}$$

In order to derive a formulation for multiplication whose implementation is more hardware-oriented we state the following.

Corollary 3.1. *Given the ith digit of A, i.e., A_i with d bits and a field element of $B^{2^{-id}} \in GF(2^m)$, their product $C^{(i)} \in GF(2^m)$ can be obtained as*

$$C^{(i)} = \sum_{j=0}^{d-1} J^{2^j} \left(a_{j+id}, B^{2^{-(id+j)}} \right),$$

where $J(x, Y) = x \cdot P(Y) \in GF(2^m)$.

Proof. Using (3.6), one has

$$C^{(i)} = \sum_{j=0}^{d-1} a_{j+id} \beta^{2^j} \cdot B^{2^{-id}} = \sum_{j=0}^{d-1} \left(a_{j+id} \cdot \beta B^{2^{-id-j}} \right)^{2^j}. \tag{3.7}$$

Now we define $J(x, Y)$ as a function of the product of a bit $x \in GF(2)$ and a field element $P(Y) \in GF(2^m)$ as

$$J(x, Y) = x \cdot P(Y). \tag{3.8}$$

Then, using (2.11) and Corollary 1, one can write $\beta B = P(B)$ to simplify $C^{(i)}$ in (3.7) as follows

$$\begin{aligned} C^{(i)} &= \sum_{j=0}^{d-1} \Big(a_{j+id} \cdot P\big(B \ll (id+j)\big) \Big)^{2^j}, \\ &= \sum_{j=0}^{d-1} J^{2^j} \left(a_{j+id}, B^{2^{-(id+j)}} \right) \end{aligned} \tag{3.9}$$

This completes the proof.

□

Then, the multiplication of A and B can be obtained from

$$C = AB = \sum_{i=0}^{q-1} \left(C^{(i)} \gg id \right). \tag{3.10}$$

In the following, we present the architecture of the proposed DL-SIPO GNB multiplier.

3.2.2 New Architecture

In order to map the formulation obtained in previous subsection to hardware, an architecture for the LSD-first DL-SIPO GNB multiplier is depicted in Fig. 3.3. Initially, the register $\langle Y \rangle$ is loaded by $B = (b_0, b_1, \cdots, b_{m-1})$ and the register $\langle Z \rangle$ is cleared to 0. The d-fold left cyclic shifts are realized by "$\overset{d}{\ll}$" as shown in Fig. 3.3. Also, as one can see in this figure, the last digit of operand A, i.e., A_{q-1}, is appended by $r = qd - m$, $0 \leq r \leq d-1$, zeros as its most significant bit ends. The remaining input bits are correspond to the terms appearing in A_{q-1} (as m is not always a multiple of digit-size d). This avoids redundant computations in the last clock cycle.

The DL-SIPO GNB multiplier architecture, has several P blocks shown as p_0 to p_{d-1} in Fig. 3.3b as a P array. As shown in this figure, P blocks use the shifted combination of $P(Y) \in GF(2^m)$ defined in (2.11) for the input operand B (preloaded in register $\langle Y \rangle$). Therefore, we first determine these combinations and after these

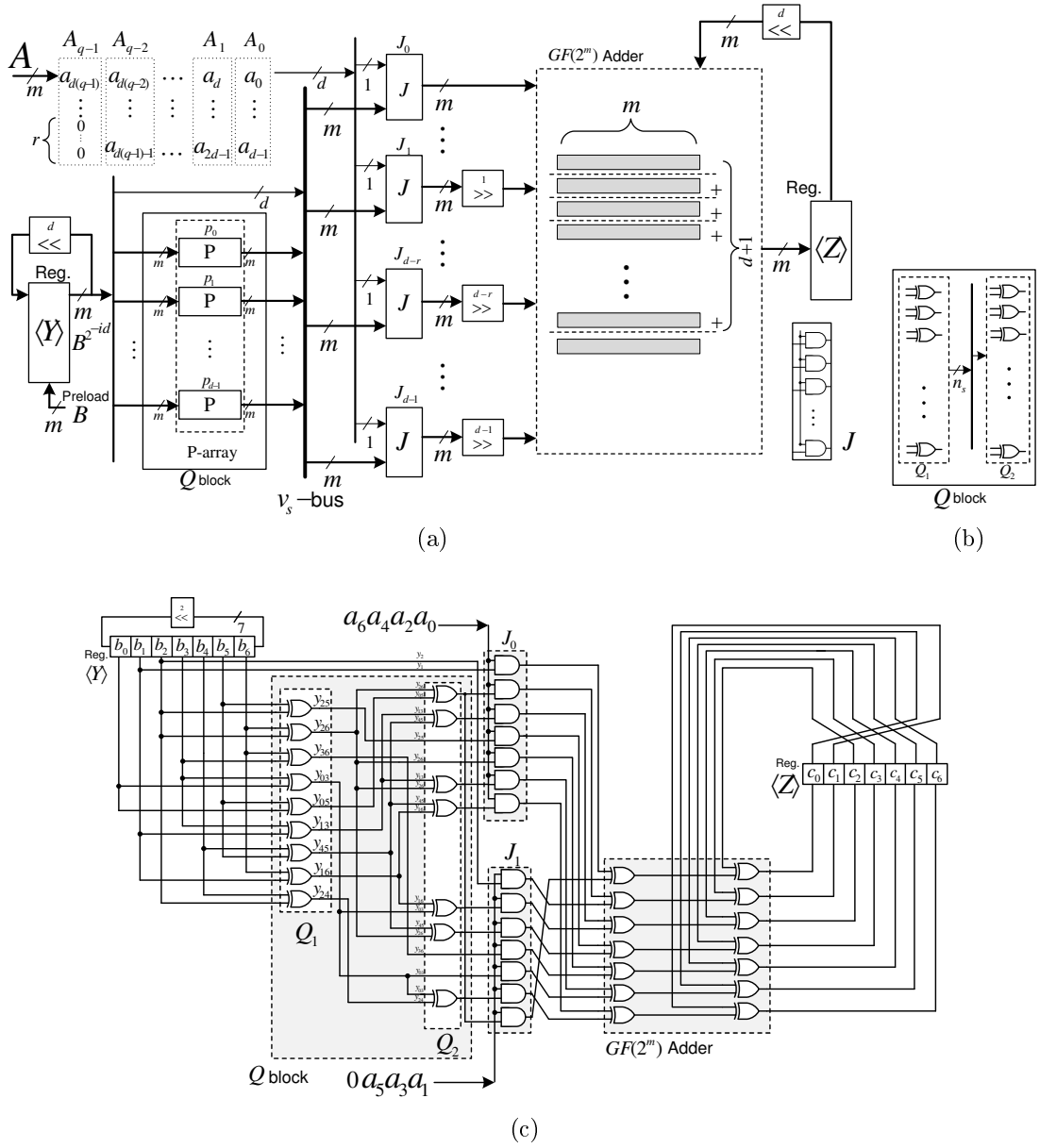


Figure 3.3: (a) The proposed architecture for LSD-first DL-SIPO multiplier. (b) an example of the proposed multiplier for type 4 GNB over $GF(2^7)$ and $d = 2$.

combinations are computed, we use their results in different computations to optimize the area complexity by reducing the number of signals and consequently number of XOR gates. We propose a method to combine the computations of the P blocks into a Q block as illustrated in the architecture of Fig. 3.3a.

The Q block is generated for the digit size d and type T GNB for operand B as $Q(Y) = (P(Y), P(Y) \gg 1, \cdots, P(Y) \gg d-1)$ as illustrated in Fig. 3.3 where $P(Y) \gg l$, $0 \leq l \leq d-1$ denotes l-fold right cyclic shift of $P(Y) \in GF(2^m)$. As shown in this figure, y_{l+1}, $0 \leq l \leq d-1$ are removed from the block Q as they are correspond to the lines on v_s-bus connected to register $\langle Y \rangle$. The Q block can also be represented by the $\mathbf{Q}$ matrix as

$$\mathbf{Q} = \begin{pmatrix} \mathbf{R}^{(0)} \\ \mathbf{R}^{(1)} \\ \mathbf{R}^{(2)} \\ \vdots \\ \mathbf{R}^{(l)} \end{pmatrix}_{v_s \times T}, \ 0 \leq l \leq d-1, \tag{3.11}$$

where using (2.16), $\mathbf{R}^{(l)}$ can be obtained by adding the (i,j)-th, $1 \leq i \leq m-1$,$1 \leq j \leq T$, entry of the matrix $\mathbf{R} = \mathbf{R}^{(0)}$, i.e., $R(i,j)$, $0 \leq R(i,j) \leq m-1$ with "l mod m", as $R(i,j)+l \bmod m$. Also, $v_s = d(m-1) - \frac{d(d-1)}{2}$ is the total number of rows inside the $\mathbf{Q}$ matrix. This is due to the fact that every two $\mathbf{R}^{(i')}$ and $\mathbf{R}^{(i'')}$, $0 \leq i', i'' \leq d-1$, have a common row with the total of $\binom{d}{2} = \frac{d(-1)}{2}$ in the $\mathbf{Q}$ matrix [5]. Then, as one can see, the multiplication of every bit of A_i in (3.9) by the outputs of the Q block which is connected to v_s-bus, is performed by J, (J_0 to J_{d-1}) blocks, using (3.8) where each J block includes m two-input AND gates as shown in Fig. 3.3a. After the first clock cycle, the content of register $\langle Y \rangle$ is $B^{2^{-d}}$ and in general it contains $B^{2^{-id}}$ after ith clock cycle. Let $Z(q) \in GF(2^m)$ denotes the field element after the q-th clock cycle whose its coordinates stored in the m-bit register $\langle Z \rangle$. Then, after one clock cycle, with the use of (3.9) the register $\langle Z \rangle$ contains

$$C^{(0)} = A_0 B = \sum_{j=0}^{d-1} J^{2^j} \left(a_j, B^{2^{-j}} \right). \tag{3.12}$$

Then, both registers $\langle Y \rangle$ and $\langle Z \rangle$ should be d-fold cyclically shifted to the left to obtain $C^{(1)}, C^{(2)}, \cdots, C^{(q-1)}$, accordingly. The sum of d m-bit intermediate results with one m-bit initial results in register $\langle Z \rangle$ is performed in the accumulator which is implemented using a $GF(2^m)$ adder (as shown in Fig. 3.3). Therefore, one can verify

that considering (3.10), after q-th clock cycle, the register $\langle Z \rangle$ contains

$$Z(q) = \Bigg(\cdots \Big(\big((C^{(0)})^{2^{-d}} + C^{(1)} \big)^{2^{-d}} + C^{(2)} \Big)^{2^{-d}} + \cdots \Bigg)^{2^{-d}} + C^{(q-1)}. \tag{3.13}$$

By comparing (3.10) with (3.13) one can write $Z(q) = C^{2^{-d(q-1)}} = C^{2^{m+(d-r)}} = C^{2^{d-r}}$. Thus, the coordinates of $C = AB$ can be obtained by $(d-r)$-fold left cyclic shift of the register $\langle Z \rangle$, i.e., $C = (Z(q) \ll d-r)$.

Remark 3.1. Using the above formulation, one can design similar architecture for the MSD-first digit-level SIPO GNB multiplier.

3.2.2.1 Complexities

In this section, the complexity of the proposed digit-level SIPO multiplier is given in terms of gate counts and critical-path delay.

The number of rows in the matrix which builds Q is $v_s = d(m-1) - \frac{d(d-1)}{2}$ and each row consists of at most $\frac{T}{2}$ pairs. We divide the Q block into two blocks Q_1 and Q_2 blocks. Block Q_1 contains at most n_s, $n_s \leq v_s \times \frac{T}{2}$, XOR gates with the delay of an XOR gate as shown in Fig. 3.3a. Block Q_2 consists of trees of XOR gates for the GNB, with $T > 2$. The Q_2 block connects its input bus to the v_s-bus having each of its output to be addition of at most T coordinates of $\langle Y \rangle$ which can be obtained by adding at most $\frac{T}{2}$ signals from the output of Q_1. Therefore, if no common subexpression in Q block are reused, the number of XOR gates in Q_1 block and Q_2 block of Fig. 3.3a are at most $v_s \frac{T}{2}$ and $v_s(\frac{T}{2}-1)$, respectively. It is noted that for the case where $d = m$ (i.e., bit-parallel architecture), the upper bound for n_s can be obtained as $\binom{m}{2} = \frac{m(m-1)}{2}$ and hence in general $n_s \leq \min\left\{\frac{v_s T}{2}, \binom{m}{2}\right\}$. Also, the number of XOR gates in the $GF(2^m)$ adder (which adds $d+1$ m-bit inputs together) is dm XOR gates. Moreover, the J blocks require dm two-input AND gates. Therefore, based on the above discussions, the followings can be stated to obtain the gate count and time complexity of the proposed multiplier architecture.

Proposition 3.2. *The gate complexities of the proposed LSD-first DL-SIPO multiplier architecture is*

$$\#AND = dm,$$
$$\#XOR \leq v_s(T-1) + dm.$$

Remark 3.2. The area complexity of proposed LSD-first DL-SIPO multiplier can be further reduced by incorporating a common subexpression elimination algorithm to $n_s + v_s \times \left(\frac{T}{2} - 1\right) + dm$ XOR gates which n_s is upper bounded by $n_s \leq \min\left\{\frac{v_s T}{2}, \binom{m}{2}\right\}$ and its exact number can be obtained by simulation.

To obtain the maximum clock frequency for the proposed multiplier, one can see that the critical-path delay of the proposed multiplier architecture includes those for the Q_1 and Q_2 blocks (i.e., T_X and $\lceil \log_2 \frac{T}{2} \rceil T_X$ respectively), the J blocks, (i.e., T_A) and the $GF(2^m)$ adder (i.e., $\lceil \log_2(d+1) \rceil T_X$). Then, the total critical-path delay due to delays through the above mentioned blocks is $T_A + (\lceil \log_2 T \rceil + \lceil \log_2(d+1) \rceil) T_X$.

3.2.2.2 Complexity Reduction

As explained in the previous subsection, the number of rows inside the $\mathbf{Q}$ matrix is $v_s = d(m-1) - \frac{d(d-1)}{2}$ to generate all signals at the output of $Q(Y)$. As mentioned in Conjecture 1, the matrix $\mathbf{R}$ contains rows with two equal entries (these entries cancel each other in the formulation). Then, the $\mathbf{Q}$ matrix has some rows with only two entries (i.e., one pair). Base on this fact and the number of times that these pairs are repeated, a subexpression sharing method presented in [9] is used here to obtain the optimized number of pairs in Q_1, i.e., n_s. In the following, we give an illustrative example for the proposed multiplier architecture.

3.2.3 An Illustrative Example

We consider the multiplication matrix $\mathbf{R}$ for type $T = 4$ GNB over $GF(2^7)$ as follows:

Table 3.2: Contents of variables in the proposed architecture for LSD-first DL-SIPO type 4 GNB multiplier over $GF(2^7)$.

Clock	LSD-First			
j	A	Y	Acc	Z
0	–	$B = 1100011$	–	0000000
1	11	1100011	0111010	0111010
2	00	0001111	0000000	1101001
3	01	0111100	1100111	1000000
4	10	1110001	1111010	$C^2 = 1111000$

$$\mathbf{R} = \begin{pmatrix} 0 & 2 & 5 & 6 \\ 1 & 3 & 4 & 5 \\ 2 & 5 & \underline{3} & \underline{3} \\ 2 & 6 & \underline{0} & \underline{0} \\ 1 & 2 & 3 & 6 \\ 1 & 4 & 5 & 6 \end{pmatrix}_{(6\times 4)} . \tag{3.14}$$

This matrix can be obtained from the location of non-zero entries (excluding the first row) of the multiplication matrix $\mathbf{M}$ as

$$\mathbf{M} = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 \end{pmatrix}_{7\times 7} .$$

Having the digit size to be $d = 2$, the matrix $\mathbf{Q}_{(11\times 4)}$ can be generated as

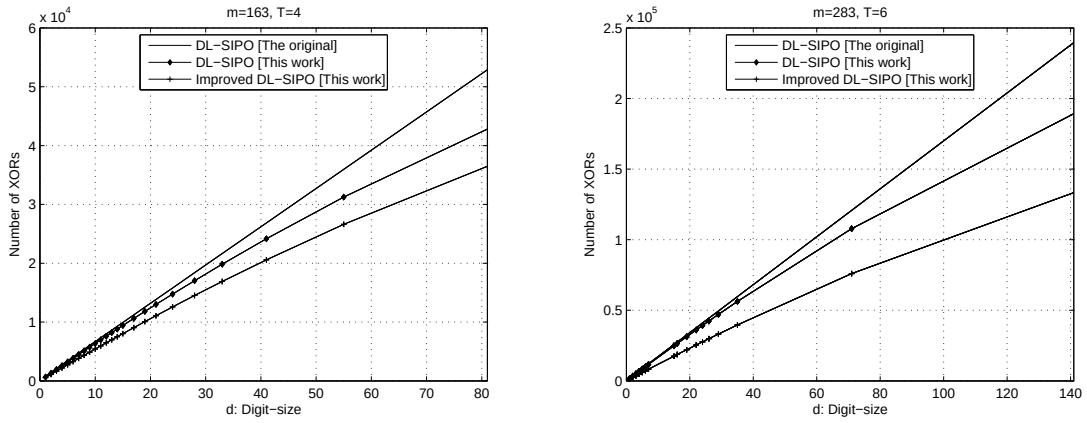
$$\mathbf{Q} = \begin{pmatrix} \mathbf{R}^{(0)} \\ \mathbf{R}^{(1)} \end{pmatrix} = \begin{pmatrix} \boxed{\underline{0} \quad 2 \quad \underline{5} \quad 6} \\ 1 \quad 3 \quad 4 \quad 5 \\ 2 \quad 5 \quad \underline{3} \quad \underline{3} \\ 2 \quad 6 \quad \underline{0} \quad \underline{0} \\ 1 \quad 2 \quad 3 \quad 6 \\ 1 \quad 4 \quad 5 \quad 6 \\ 1 \quad 3 \quad 6 \quad 0 \\ 2 \quad 4 \quad 5 \quad 6 \\ 3 \quad 6 \quad \underline{4} \quad \underline{4} \\ 3 \quad 0 \quad \underline{1} \quad \underline{1} \\ 2 \quad 3 \quad 4 \quad 0 \\ \boxed{2 \quad 5 \quad 6 \quad 0} \end{pmatrix} \begin{matrix} \} \mathbf{R}^{(0)} \\ \\ \\ \\ \\ \\ \} \mathbf{R}^{(1)} \\ \\ \\ \\ \\ \end{matrix}$$

Removed $\longrightarrow$ (the boxed row)

11×4

In this matrix, $\mathbf{R}^{(1)}$ is obtained by adding the (i,j)-th entry of $\mathbf{R} = \mathbf{R}^{(0)}$ by "1 mod 7". As one can see, the number of rows in this matrix is $v_s = 2\times(7-1) - \binom{2}{2} = 11$ (as $\mathbf{R}^{(0)}$ and $\mathbf{R}^{(1)}$ have a common row which is removed from this matrix) and it has $2d = 4$ rows with just two entries (as the equal underlined entries cancel each other in those four rows). Then, we first collect these pairs (in rows with two entries), i.e., (2,5), (2,6), (3,6), and (0,3) as a pairset to initialize $\mathbf{Q}_1$ matrix. The numbers of times that these pairs are repeated are 2,3,2, and 2, respectively. Then, applying the common subexpression elimination algorithm presented in [9], one can obtain the pairs inside the matrix $\mathbf{Q}_1$ as $\mathbf{Q}_1 = \{y_{25}, y_{26}, y_{36}, y_{03}, y_{05}, y_{13}, y_{45}, y_{16}, y_{24}\}$, where $y_{ij} = y_i + y_j$ and $n_s = 9$ is the number of pairs in Q_1. Also, as each row in $\mathbf{Q}$ needs $(\frac{T}{2} - 1)$ gates excluding the rows with only two entries (which is $2d$ here) and there are v_s rows in total, then $v_s(\frac{T}{2} - 1) - 2d = 7$ XOR gates in block Q_2 is required to produce the the outputs of $Q(Y)$. The architecture of the proposed multiplier over $GF(2^7)$ for $d = 2$ is depicted in Fig. 3.3c. Therefore, the complexity of the presented improved DL-SIPO multiplier is $n_s + v_s(\frac{T}{2} - 1) - 2d + dm = 30$ XOR gates. Note that the unoptimized structure (without common subexpression sharing) requires $\left(d(m-1) - \frac{d(d-1)}{2}\right)(T-1) - 2d + dm = 43$ XOR gates and the architecture proposed in [7] requires $m(dT+1) - d = 61$ XOR gates. Also, the critical-path delay is $T_A + 4T_X$.

For the multiplier operation, as one can see in Fig. 3.3c, operand A is grouped into four digits as $A_0 = (a_0, a_1)$, $A_1 = (a_2, a_3)$, $A_2 = (a_4, a_5)$, and $A_3 = (a_6, 0)$, each with the size of two bits, i.e., $d = 2$. Before starting the clock, the register $\langle Y \rangle$ is



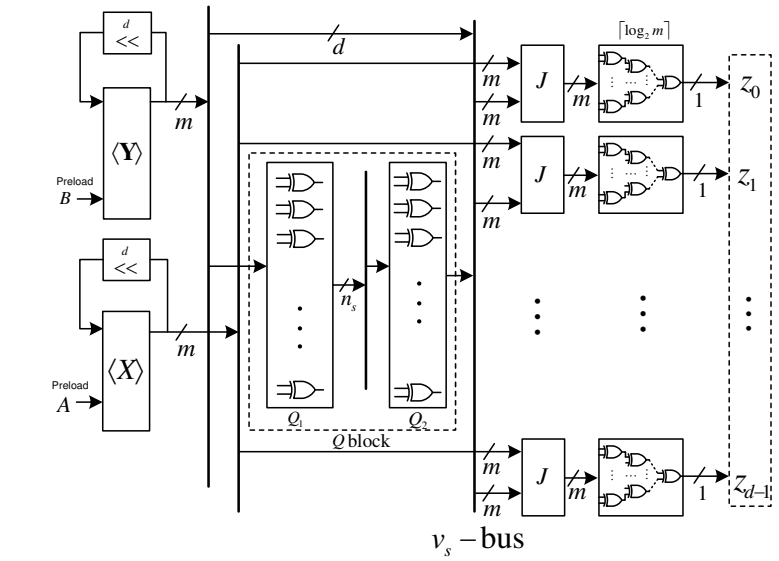
(a) (b) d

Figure 3.4: Comparison among the numbers of XOR gates required in the original and the improved digit-level SIPO multiplier architectures [7] for (a) type $T = 4$ GNB over $GF(2^{163})$ and (b) type $T = 6$ GNB over $GF(2^{283})$.

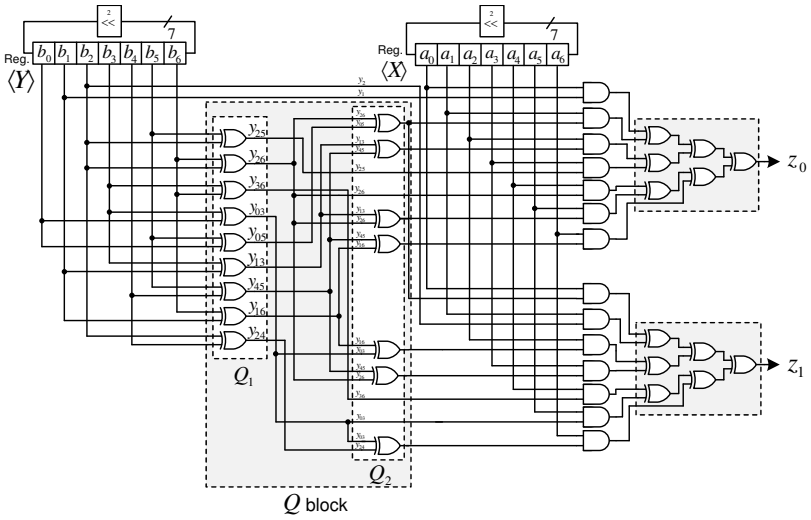
loaded with the coordinates of $B = (b_0, b_1, \cdots, b_6)$ and register $\langle Z \rangle$ is cleared to zero, i.e., $\langle Z \rangle = (0, 0, \cdots, 0)$. Then, in the first clock cycle, two LSD bits, i.e., a_0 and a_1 of operand A, are the inputs of the corresponding AND gates. One can realize that after $q = \lceil \frac{7}{2} \rceil = 4$ clock cycles, the result of $C^{2^{d-r}}$ is available in parallel at register $\langle Z \rangle$. The contents of registers are given in Table 3.2 for $A = B = (11000011)$. Note that as mentioned before, the result of multiplication $C = AB$ is obtained after one $(d - r = 1)$ left cyclic shift of the content of register $\langle Z \rangle$ at the last clock cycle, i.e., $C = (Z(q) \ll 1) = 1110001$.

3.2.4 Simulations

To compare the complexity of the proposed improved DL-SIPO GNB multiplier to the counterpart a MATLAB code is written to generate common pairs and signals used in the blocks Q_1 and Q_2 of the proposed architectures in Fig. 3.3a. The simulation results of the algorithm for the improved DL-SIPO GNB multiplier for $T = 4$ over $GF(2^{163})$ and $T = 6$ over $GF(2^{283})$ are obtained and plotted in terms of different digit sizes in Fig. 3.4a and 3.4b, respectively.

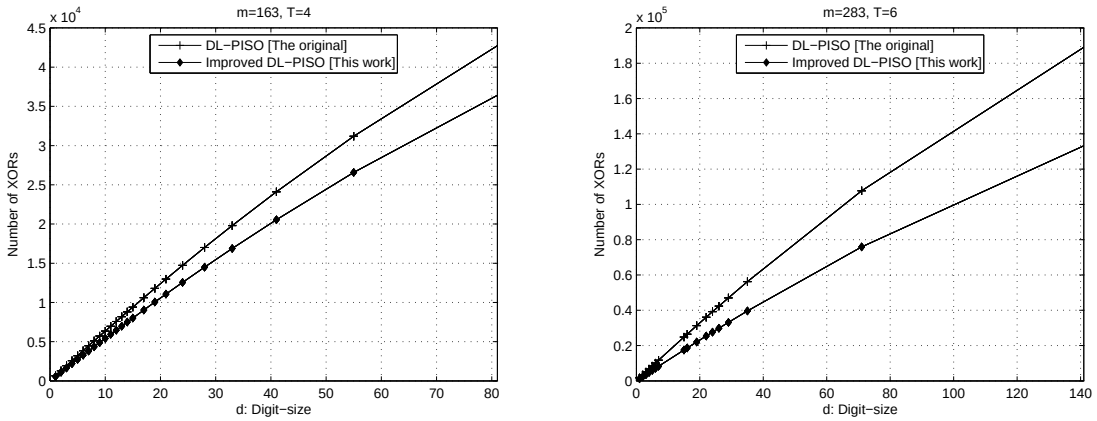


(a)



(b)

Figure 3.5: (a) The architecture of the improved digit-level PISO GNB multiplier architecture with the LSD-first output. (b) The improved architecture of type 4 GNB multiplier over $GF(2^7)$ and $d = 2$.



(a) (b)

Figure 3.6: Comparison among the numbers of XOR gates required in the original and improved digit-level PISO multiplier architectures for (a) type $T = 4$ GNB over $GF(2^{163})$ and (b) type $T = 6$ GNB over $GF(2^{283})$.

3.3 New Architecture for Digit-Level PISO GNB multiplier

3.3.1 Low-Complexity Digit-Level PISO GNB Multiplier

In this subsection, we present a low-complexity architecture for the digit-level PISO GNB multiplier presented in Chapter 2. The improvement of the new architecture is based on a formulation of the multiplication operation, which is given in the following.

3.3.1.1 Improved Architecture

In this section, similar to the previous section, we present an improved architecture for DL-PISO GNB multiplier and reduce its area complexity. As shown in Fig. 2.2, the digit-level PISO multiplier architecture has several BTX blocks that use the same combination of the input operand B (preloaded in the register $\langle Y \rangle$). We combine the computations of the parallel computed functions into a Q block (which is the same as the one presented in previous section for DL-SIPO architecture) as illustrated in the architecture in Fig. 3.5. As shown in this figure, y_{1+d}s are removed from the block Q as they are corresponding to the lines on v_s-bus connected to the register $\langle Y \rangle$. The v_s-bus contains all signals to generate all different terms required in (2.14). These signals are implemented by the blocks of Q_1 and Q_2 inside the Q block. We first use the block Q_1 to implement all pairs required for all signals in (2.14). In

this architecture, each J block consists of m 2-input AND gates to implement (2.15). Then, a level of XOR trees are utilized to implement all $z_0, z_1, \cdots, z_{d-1}$ coordinates in (2.15). The proposed improved architecture provides the LSD of multiplication at the first clock cycle (LSD-first).

For the purpose of illustration, the improved architecture of DL-PISO ($d = 2$) for type 4 GNB over $GF(2^7)$ is shown in Fig. 3.5b. As shown in this figure, the Q_1 and Q_2 blocks are generated for the given matrix $\mathbf{R}$ in (3.14). The registers $\langle X \rangle$ and $\langle Y \rangle$ should be initialized with the coordinates of A and B and then after each clock cycle two bits of $C = AB$ become available at the output.

In the following, we derive the complexity of the improved LSD-first DL-PISO GNB multiplier.

3.3.1.2 Complexities

To determine the area and time complexities of the presented architecture, the following is stated.

Proposition 3.3. For type T GNB over $GF(2^m)$, the improved digit-level PISO GNB multiplier requires dm AND gates and $n_s + v_s \times \left(\frac{T}{2} - 1\right) + d(m-1)$ XOR gates. Also, the critical-path delay of the improved architecture is the same as the original structure, i.e., $T_A + (\lceil \log_2 T \rceil + \lceil \log_2 m \rceil) T_X$.

Proof. The proof is similar to the one presented in Subsection 3.2.2.1.

□

We further optimize the number of XOR gates required for the improved LSD-first DL-PISO GNB multiplier similar to the one proposed for DL-SIPO multiplier. The results of simulations obtained for different digit-size and are plotted in Figs. 3.6a and 3.6b for $m = 163$ and $m = 283$, respectively. As one can see, the improved architecture requires fewer number of XOR gates.

3.3.2 Complexity Comparison

In Table 3.3, the time and area complexities of the presented DL-SIPO multiplier (before applying common subexpression elimination algorithm) are compared with the ones, namely, DL-SIPO [7], DL-PISO [5], and DL-PIPO [45] multipliers as they appear to be the most recently proposed works available in the literature. It is noted that our presented multiplier architecture (Fig. 3.3) requires fewer number of gates

Table 3.3: Comparison of the most recently proposed type T digit-level GNB multipliers over $GF(2^m)$ with parallel outputs.

Multiplier Architecture	# AND gates	# XOR [a] gates	# Reg.	Critical-Path delay	Output	Input	
						A	B
WL-PIPO [45]	dm	$2v_p.T + d$	$3m$	$T_A + (\lceil \log_2 T \rceil + \lceil \log_2 (d+1) \rceil) T_X$	Parallel	Parallel	Parallel
DL-PIPO [5]	dm	$\leq v_p.T + \frac{d}{2}(m+1)$	$3m$	$T_A + (\lceil \log_2 T \rceil + \lceil \log_2 (d+1) \rceil) T_X$	Parallel	Parallel	Parallel
DL-PIPO [9]	dm	$\leq v_p(T-1) + dm$	$3m$	$T_A + (\lceil \log_2 T \rceil + \lceil \log_2 (d+1) \rceil) T_X$	Parallel	Parallel	Parallel
DL-SIPO [7]	dm	$2v_p.T + d(T-1) + m$	$2m$	$T_A + (\lceil \log_2 T \rceil + \lceil \log_2 d \rceil + 1) T_X$	Parallel	Serial	Parallel
DL-SIPO (Fig. 3.3)[2]	dm	$\leq v_s(T-1) + dm$	$2m$	$T_A + (\lceil \log_2 T \rceil + \lceil \log_2 (d+1) \rceil) T_X$	Parallel	Serial	Parallel

1. $v_p = \frac{d(m-1)}{2}$ and $v_s = d(m-1) - \frac{d(d-1)}{2}$.

2. Without applying common subexpression elimination algorithm.

[a]

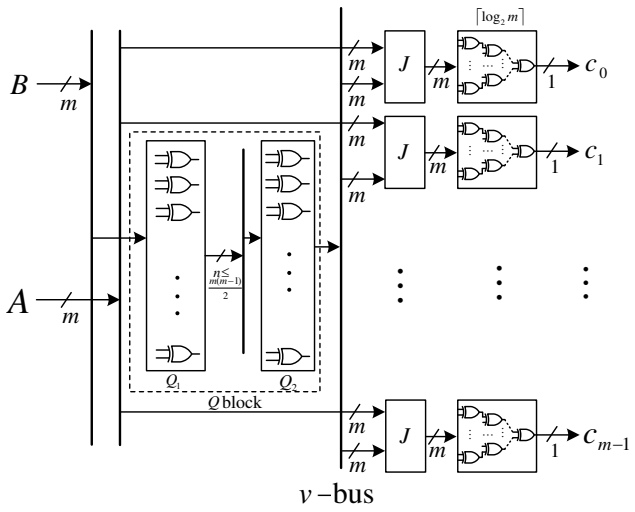


Figure 3.7: The architecture of proposed bit-parallel GNB multiplier

than the previously proposed ones DL-SIPO [7] and DL-PIPO [45]. Also, as seen in this table, in terms of time complexity our presented multiplier (Fig. 3.3) is favorably comparable with the DL-SIPO [7]. Moreover, in Fig. 3.4, the area complexity of the improved architecture over $GF(2^{163})$ and $GF(2^{283})$ after applying the common subexpression elimination algorithm is illustrated in terms of different digit sizes and compared with the ones of its counterpart [7]. As illustrated in Figs. 3.4 and 3.6, the presented improved architectures require fewer XOR gates than the one proposed in [7] and the original one proposed in [5], respectively.

In the following section, we propose a new bit-parallel multiplier.

3.4 An Extension to Bit-Parallel GNB Multiplier

Based on the formulation used in the previous sections, we present a new bit-parallel GNB multiplier over $GF(2^m)$ in this section. The proposed digit-level GNB multiplier architectures can be easily scaled up to the bit-parallel type. To obtain the bit-parallel multiplier, one can implement (2.4) in hardware for all c_l, $0 \leq l \leq m-1$. Thus, the hardware architecture of a bit-parallel multiplier is obtained by implementing m copies of identical structures used for c_0 with cyclic shifts of their inputs.

The architecture of the proposed bit-parallel GNB multiplier is depicted in Fig. 3.7. In Propositions 3.1 and 3.3 for DL-PIPO and DL-PISO multiplier architectures we defined n_p and n_s as the number of pairs (inside the blocks ρ_1 and Q_1) to build the ρ and Q blocks, respectively. For a bit-parallel architecture the upper bound for the number of pairs in these blocks are the equal to the the all combinations of

two coordinates of A, i.e., $n = \binom{m}{2} = \frac{m(m-1)}{2}$ combinations. Note that for $T = 2$, $n = \frac{m(m-1)}{2}$ and the block ρ_2 connects its input bus to the next bus without using any XOR gates. Note that the exact complexities of Q_1 and Q_2 depend on the GNB. However, one can find the upper bound for the number of XOR gates and time delay of this structure as follows.

Proposition 3.4. *For Type T GNB over $GF(2^m)$, the proposed bit-parallel GNB multiplier architecture requires m^2 AND gates and at most $(T+4)(\frac{m(m-1)}{4})$ XOR gates with the critical path delay of*

$$T_C = T_A + (\lceil \log_2 T \rceil + \lceil \log_2 m \rceil) T_X, \tag{3.15}$$

where T_A and T_X are the time delay of a two-input AND gate and an XOR gate, respectively.

Proof. The proof can be obtained by equating $n = n_s = n_p = \frac{m(m-1)}{2}$ in Propositions 3.1 and 3.3. Then, one can obtain the upper bound for the total number of XOR gates as $\frac{m(m-1)}{2} + \frac{m(m-1)(T-2)}{4} + m(m-1) = (T+4)(\frac{m(m-1)}{4})$.

The critical-path delay of the proposed architecture can be obtained by adding the delays of the three blocks of Q_1, Q_2, J, and the $GF(2^m)$ adders which are T_X, $\lceil \log_2 \frac{T}{2} \rceil T_X$, T_A, and $\lceil \log_2 m \rceil T_X$, respectively. This results in the total delay of $T_X + \lceil \log_2 \frac{T}{2} \rceil T_X + T_A + \lceil \log_2 m \rceil T_X = T_A + (\lceil \log_2 T \rceil + \lceil \log_2 m \rceil) T_X$, which completes the proof.

□

3.4.1 Comparison

The time and area complexities of the proposed bit-parallel GNB multiplier and the previous schemes are compared in Table 3.4 for general and special values of T. As shown in this table, the critical path delay of the proposed multiplier matches the fastest results available in the literature. For type $T = 2$ GNB, the number of XOR gates also matches the fastest result available in the open literature, i.e., $1.5m(m-1)$. However, it is much greater than the sub-quadratic results proposed in [63] and [59] which require much higher delay as compared to the one proposed here. It is interesting to note that for $T > 2$, the proposed multiplier requires smaller area in comparison to its counterparts which are proposed most recently with the same delay as shown in this table.

It should be noted that, to obtain the exact number of XOR gates for a given GNB, the exact value of n should be obtained by simulations. Using the complexity reduction algorithm proposed in Section 3.1, a comparison between the number of XOR gates of bit-parallel GNB multipliers is illustrated in Table 2 for $GF(2^{163})$ and $GF(2^{283})$ fields recommended by NIST for ECDSA.

Table 3.4: Area and time complexity comparison of bit-parallel GNB multipliers over $GF(2^m)$. Note that for Type T GNB: $C_N \leq Tm - T + 1$.

Multiplier	type$T \geq 2$		
	#AND	#XOR	Critical path
Massey & Omura [35]	m^2	$m(C_N - 1)$	$T_A + \lceil \log_2 C_N \rceil T_X$
Gao & Sobelman[51]	m^2	$m(C_N - 1)$	$T_A + (\lceil \log_2 T \rceil + \lceil \log_2 m \rceil) T_X$
[50]	m^2	$\leq \frac{m}{2}(C_N + m - 2)$	$T_A + (\lceil \log_2 (C_N + 1) \rceil) T_X$
DLGMp [5], [6] $(d = m)$	m^2	$\leq \frac{m}{2}(C_N + m)$	$T_A + (\lceil \log_2 T \rceil + \lceil \log_2 (m) \rceil) T_X$
DLGMs [5] $(d = m)$	m^2	$\leq \frac{m(m-1)}{2}(T + 1)$	$T_A + (\lceil \log_2 T \rceil + \lceil \log_2 (m) \rceil) T_X$
DL-PIPO [45] $(d = m)$	m^2	$\leq Tm(m - 1) + m$	$T_A + (\lceil \log_2 T \rceil + \lceil \log_2 (m) \rceil) T_X$
DL-SIPO [7] $(d = m)$	m^2	$\leq (T - 1)m^2 + m(m - 1)$	$T_A + (\lceil \log_2 T \rceil + \lceil \log_2 (m) \rceil) T_X$
This work	m^2	$\leq (\frac{m(m-1)}{4})(T + 4)$	$T_A + (\lceil \log_2 T \rceil + \lceil \log_2 (m) \rceil) T_X$
	$T = 2$		
[35, 51]	m^2	$2m(m - 1)$	$T_A + \lceil \log_2 (2m - 1) \rceil T_X$
Koc & Sunar [48]	m^2	$1.5m(m - 1)$	$T_A + (1 + \lceil \log_2 m \rceil) T_X$
Fan & Hasan [59]	$2m^{1.6}$	$11m^{1.6} - 12m + 1$	$T_A + (2 \log_2 m + 1) T_X$
Gathen et. al [63]	$2m^{1.6}$	$7.6m^{1.6} + \mathcal{O}(m \log m)$	$T_A + (2 \log_2 m + 1) T_X$
[50, 5, 6], This work	m^2	$1.5m(m - 1)$	$T_A + (1 + \lceil \log_2 m \rceil) T_X$
	$T = 4$		
[35], [51]	m^2	$4m^2 - 4m$	$T_A + (2 + \lceil \log_2 (m) \rceil) T_X$
[50]	m^2	$2.5m^2 - 4.5m$	$T_A + \lceil 1 + \log_2 (2m - 1) \rceil T_X$
DLGMp [5], [6] $(d = m)$	m^2	$2.5m^2 - 1.5m$	$T_A + (2 + \lceil \log_2 (m) \rceil) T_X$
DLGMs [5] $(d = m)$	m^2	$2.5m^2 - 2.5m$	$T_A + (2 + \lceil \log_2 (m) \rceil) T_X$
DL-PIPO [45] $(d = m)$	m^2	$4m^2 - 3m$	$T_A + (2 + \lceil \log_2 (m) \rceil) T_X$
DL-SIPO [7] $(d = m)$	m^2	$4m^2 - m$	$T_A + (\lceil \log_2 T \rceil + \lceil \log_2 (m) \rceil) T_X$
This work	m^2	$\leq 2m^2 - 2m$	$T_A + (2 + \lceil \log_2 (m) \rceil) T_X$
	$T = 6$		
[35], [51]	m^2	$6m^2 - 6m$	$T_A + (3 + \lceil \log_2 (m) \rceil) T_X$
[50]	m^2	$3.5m^2 - 3.5m$	$T_A + (\lceil \log_2 (6m - 4) \rceil) T_X$
DLGMp [5], [6] $(d = m)$	m^2	$3.5m^2 - 2.5m$	$T_A + (3 + \lceil \log_2 (m) \rceil) T_X$
DLGMs [5] $(d = m)$	m^2	$3.5m^2 - 3.5m$	$T_A + (3 + \lceil \log_2 (m) \rceil) T_X$
DL-PIPO [45] $(d = m)$	m^2	$6m^2 - 5m$	$T_A + (3 + \lceil \log_2 (m) \rceil) T_X$
DL-SIPO [7] $(d = m)$	m^2	$6m^2 - m$	$T_A + (3 + \lceil \log_2 (m) \rceil) T_X$
This work	m^2	$\leq 2.5m^2 - 2.5m$	$T_A + (3 + \lceil \log_2 (m) \rceil) T_X$

Table 3.5: FPGA implementation of BL-SIPO (Fig. 2.1) multiplier for type 4 over $GF(2^{163})$ on xc4vlx100-ff1148 device.

Multiplier	CPD [ns]	FF	LUT	Slice	Time [ns]
BL-SIPO	1.9	326	486	323	309.7

Table 3.6: ASIC synthesis results for BL-SIPO (Fig. 2.1) multiplier for type 4 over $GF(2^{163})$.

Multiplier	CPD [ns]	Area [μm^2]	Time [ns]
BL-SIPO	0.34	6817.2	55.42

3.5 FPGA and ASIC Implementations

In this section, we implement the presented architectures in the previous sections to evaluate their area and time requirements. We have selected the Xilinx® Virtex™-4 xc4vlx100-ff1148 device as the target FPGA. In terms of available resources, xc4vlx100-ff1148 contains 49,152 slices (98,304 LUTs and 98,304 registers). Each slice contains two flip-flops (FFs) and two 4-input look-up tables (LUTs) [64].

The proposed multiplier architectures are modeled in VHDL and synthesized for different digit sizes using XST™ of Xilinx® ISE™ version 12.1 design software. Also, 65-nm Complementary Metal-Oxide-Semiconductor (CMOS) library has been chosen for the synthesis on application-specific integrated circuit (ASIC) technology. The proposed architectures synthesized using Synopsys® Design Vision® which is a GUI for Synopsys® Design Compiler® tools. The correctness of the multiplier architectures is verified by Xilinx® ISE™ Simulator (ISim) and m-bit 2-to-1 multiplexers are used to preload operands to the registers in each architecture. For the FPGA implementations, the optimization goal is set to the speed (i.e., default) and optimization effort is set to normal and the area (Slices, LUTs, and FFs) and timing (ns) for the critical-path delays (CPD) are obtained for different digit sizes. It is noted that the results of the implementations on FPGA, are all after post place and route results. For the ASIC implementations, the map effort is set to medium with a target clock period of 5 ns and the area (μm^2) and timing (ns) are obtained for each of the designs.

We first implemented the LSB-first BL-SIPO (Fig. 2.1) multiplier and the results are tabulated in Table 3.5 and 3.6, for FPGA (after post place and route) and ASIC (after synthesis), respectively. Then, we have implemented the proposed architectures for LSD-first SIPO, digit-level PISO, and digit-level PIPO, multipliers for different

Table 3.7: FPGA (Xilinx® Virtex™-4 xc4vlx100-ff1148 device) and ASIC (65-nm CMOS library) synthesis results for the improved DL-SIPO (Fig. 3.3) multiplier architectures for type 4 GNB over $GF(2^{163})$ for different digit sizes.

digit size	$q = \lceil \frac{m}{d} \rceil$	FPGA Implementation					ASIC Synthesis		
		Slices	FF	LUT	CPD $[ns]$	T $[ns]$	Area $[\mu m^2]$	CPD $[ns]$	T $[ns]$
11	15	1,691	326	3,365	4.8	72.0	34,278.4	0.93	13.95
21	8	3,099	326	6,185	5.8	46.4	63,283	1.56	12.48
33	5	5,739	326	10,281	6.3	31.5	97,420.4	2.16	10.80
41	4	7,229	326	12,783	6.5	26.0	120,295	2.57	10.28
55	3	9,323	326	16,715	6.7	20.1	160,298.3	3.25	9.75

Table 3.8: FPGA (Xilinx® Virtex™-4 xc4vlx100-ff1148 device) and ASIC (65-nm CMOS library) synthesis results for the improved DL-PISO (Fig. 3.5) multiplier architecture for type 4 GNB over $GF(2^{163})$ for different digit sizes.

digit size	$q = \lceil \frac{m}{d} \rceil$	FPGA Implementation					ASIC Synthesis		
		Slices	FF	LUT	CPD $[ns]$	T $[ns]$	Area $[\mu m^2]$	CPD $[ns]$	T $[ns]$
11	15	1,899	444	3,912	5.7	85.5	34,837.4	1.38	20.70
21	8	3,754	408	6,995	6.1	48.8	63,397.2	1.85	14.80
33	5	5,908	365	10,735	6.8	34.0	97,804.2	2.37	11.85
41	4	7,385	378	13,218	6.9	27.6	121,356	2.94	10.96
55	3	9,678	419	17,348	7.3	21.9	161,494.8	3.85	10.65

digit sizes. The results of the implementations for different digit sizes are reported in Tables 3.7, 3.8, and 3.9. As one can see the digit-level PIPO multiplier architecture requires smallest area for both FPGA and ASIC implementations. Moreover, it is faster than the other multiplier architectures. We note that one can reduce the critical-path delay of the proposed multiplier architectures by pipelining the multiplier architectures and maintain high-throughput performance. It should be noted that for any particular application the digit-size should be chosen in such a way to achieve highest performance considering the time-area trade-offs.

3.6 Conclusion

In this chapter, we have proposed three improved multiplier architectures, namely DL-PIPO, DL-PISO, and DL-SIPO, for digit-level GNB multiplication. We have proposed a complexity reduction algorithm to reduce the complexity of each multiplier. Then, we have derived the area and time complexities of the proposed architectures and compared them with the counterparts in the literature. It has been shown that

Table 3.9: FPGA (Xilinx® Virtex™-4 xc4vlx100-ff1148 device) and ASIC (65-nm CMOS library) synthesis results for the improved DL-PIPO (Fig. 3.1) multiplier architecture for type 4 GNB over $GF(2^{163})$ for different digit sizes.

digit size	$q = \lceil \frac{m}{d} \rceil$	FPGA Implementation					ASIC Synthesis		
		Slices	FF	LUT	CPD $[ns]$	T $[ns]$	Area $[\mu m^2]$	CPD $[ns]$	T $[ns]$
11	15	1,563	495	2,399	4.7	70.5	28,667	0.91	13.65
21	8	2,545	532	4,261	4.9	39.2	52,663	1.48	11.84
33	5	4,033	554	7,194	5.4	27.0	80,566	2.16	10.8
41	4	4,628	502	8,503	5.6	22.4	99,546	2.59	10.36
55	3	6,484	500	11,412	5.8	17.4	132,225	3.39	10.17

the proposed architectures require smaller area in comparison to the leading ones in the literature in terms of area and time complexities. For studying the application of the proposed multiplier architectures, we have implemented them on FPGA and ASIC and the results are compared. We also extended the DL-PISO multiplier architecture to a bit-parallel architecture and its time and area complexities also compared with the counterparts. As seen from the FPGA and ASIC implementation results, the DL-PIPO multiplier architecture requires the smallest area and runs in highest clock frequencies in comparison to the DL-SIPO and DL-PISO architectures. These multiplier architectures are suitable for the applications such as exponentiation and point multiplication on binary elliptic curves where GNB multiplication is desired. In the next chapter, we employ the DL-PIPO multiplier architecture to design a ECC-based crypto-processor. We also provide an efficient pipelined architecture for this multiplier as well.

Chapter 4

Efficient FPGA Implementation of Point Multiplication over Binary Edwards and Generalized Hessian Curves Using Gaussian Normal Basis

IN the previous chapter, we presented a low complexity digit-level parallel-in parallel-out (DL-PIPO) architecture for Gaussian normal basis multiplier. In this chapter, we efficiently pipeline the DL-PIPO proposed architecture and study its time-area trade-offs. Then, we choose efficient values for the digit-size and compare the results with the non-pipelined architecture. We employ the proposed multiplier architecture for efficient implementation of point multiplication over binary elliptic curves, including binary generic, Edwards, and generalized Hessian curves. We demonstrate how parallelization in higher levels can be performed by full resource utilization of computing point addition and point doubling formulas for the binary Edwards and generalized Hessian curves. We employ the w-coordinate differential formulations for computing point multiplication. Using a look-up table (LUT) based pipelining and efficient digit-level GNB multiplier, we evaluate the LUT complexity and time-area trade-offs of the proposed crypto-processor on FPGA. We compare the implementation results of point multiplication on these curves with the ones on the traditional binary generic curve. We note that, this is the first FPGA implementation of point multiplication on binary Edwards and generalized Hessian curves represented by w-coordinates.

The main contributions of this chapter are as follows. It is noted that these contributions have been also presented in [65] and can be can be summarized as

follows:

- We propose an efficient hardware architecture for point multiplication on binary Edwards and generalized Hessian curves incorporating higher level parallelization and optimum lower level scheduling. This increases the overall performance considering maximum utilization of available resources.
- We incorporate w-coordinate version of Montgomery's ladder for point multiplication in binary Edwards and generalized Hessian curves using mixed differential representation.
- For the proposed crypto-processor architecture over $GF(2^m)$, we obtain the optimum digit sizes in terms of time-area trade-offs for the proposed fast and low-complexity digit-level Gaussian normal basis multiplier.
- Finally, we perform efficient FPGA implementations of point multiplication on binary Edwards and generalized Hessian curves over $GF(2^{163})$ on a Xilinx® Virtex™-5 device and investigate the LUT-based time-area efficiency for different digit sizes. We have also implemented ECC on binary generic curve and compared its FPGA implementation results with the ones obtained for binary Edwards and generalized Hessian curves.

The rest of the chapter is organized as follows. In Section 4.1, preliminaries of arithmetic on binary Edwards and generalized Hessian curves are presented. In Section 4.2, point multiplication and parallelization of point addition and doubling are explained. The proposed hardware architecture for elliptic curve crypto-processor is presented in Section 4.3. In this section, a pipelined version of digit-level PIPO GNB multiplier architecture proposed in the previous chapter is also presented and analyzed in terms of time-area trade-offs for different digit sizes. Section 4.4 presents the results of FPGA implementations for the proposed ECC crypto-processor. Finally, we conclude this chapter in Section 4.5.

4.1 Preliminaries

4.1.1 Arithmetic over Binary Edwards and Generalized Hessian Curves

It is well known that a non-supersingular binary generic (short Weierstraß) elliptic curve can be defined by a set of points (x, y) and a special point at infinity $\mathcal{O}$ (group

identity) that satisfy the following equation

$$E_{a,b}/GF(2^m): \ y^2 + xy = x^3 + ax^2 + b, \tag{4.1}$$

where $a, b \in GF(2^m)$ and $b \neq 0$ [11]. These curves are also called anomalous binary curves or Koblitz curves if $a \in \{0, 1\}$ and $b = 1$, i.e., defined over $GF(2)$ [66].

Binary Edwards curves belong to a special class of generic elliptic curves defined over binary field when $m \geq 3$ [1]. The merit of binary Edwards curves over generic curves is that they have two special properties of being unified and complete [1]. The former is that the point addition formulations can be used for point doubling while the latter means that point addition formulations can be used for all pairs of inputs on the curve.

Definition 4.1. [1] Let $\mathbb{K}$ be a finite field of characteristic two, i.e., $\text{char}(\mathbb{K}) = 2$ and d_1 and d_2 be the elements of $\mathbb{K}$ with $d_1 \neq 0$ and $d_2 \neq d_1^2 + d_1$. The binary Edwards curve with coefficients d_1 and d_2 is the affine curve

$$\begin{aligned} &E_{B,d_1,d_2}/GF(2^m): \\ &d_1(x + y) + d_2(x^2 + y^2) = xy + xy(x + y) + x^2y^2, \end{aligned} \tag{4.2}$$

where $d_1, d_2 \in GF(2^m)$.

Given a point $P = (x, y)$, its negation, $-P$, is obtained as (y, x) which has no cost [1]. The point $(0, 0)$ is the neutral element and $(1, 1)$ has order 2 [1]. The binary Edwards curves are complete if $\text{Tr}(d_2) = 1$, i.e., d_2 cannot be written as $e^2 + e$ for any e in $\mathbb{K}$, where Tr is the absolute trace of $GF(2^m)$ over $GF(2)$ [1].

Definition 4.2. [2] Let c and d to be elements of $\mathbb{K}$ such that $c \neq 0$ and $d^3 \neq 27c$. The generalized Hessian curve $H_{c,d}$ over $\mathbb{K}$ is defined by the equation

$$H_{c,d}/GF(2^m): \ x^3 + y^3 + c = dxy, \tag{4.3}$$

where $c = 1$ results in a Hessian curve, i.e., H_d.

Note that the generalized Hessian curves are complete if and only if c is not a cube in $\mathbb{K}$.

The standard formulas on generic curves [3] fail in computing addition of two points on curves if one of the points or their addition is at infinity. These possibilities should be tested before designing an elliptic curve cryptosystem. Note that point

Table 4.1: Cost of point operations on binary Edwards curves (BECs), generalized Hessian curves (GHCs), and binary generic curves (BGCs) over $GF(2^m)$ [1], [2], and [3].

Curve	Curve Parameter	Combined Addition and Doubling[1]	
		Projective Diff	Mixed Diff
BEC [1]	$d_1 \neq d_2$	$8\mathbf{M} + 4\mathbf{S} + 2\mathbf{D}$	$6\mathbf{M} + 4\mathbf{S} + 4\mathbf{D}$
	$d_1 = d_2$	$7\mathbf{M} + 4\mathbf{S} + 2\mathbf{D}$	$5\mathbf{M} + 4\mathbf{S} + 2\mathbf{D}$
GHC [2]	$c \neq 1$	$7\mathbf{M} + 4\mathbf{S} + 3\mathbf{D}$	$5\mathbf{M} + 4\mathbf{S} + 3\mathbf{D}$
	$c = 1$	$7\mathbf{M} + 4\mathbf{S} + 2\mathbf{D}$	$5\mathbf{M} + 4\mathbf{S} + 2\mathbf{D}$
BGC [3]	$b \neq 0$	$7\mathbf{M} + 5\mathbf{S} + 1\mathbf{D}$	$5\mathbf{M} + 5\mathbf{S} + 1\mathbf{D}$

1. **M**, **S**, and **D**, are the costs of multiplication of two field elements, a squaring, and a multiplication by a constant, respectively.

addition and doubling formulas on binary Edwards and generalized Hessian curves work for all input pairs. This characteristic is called completeness. In what follows, we discuss the point addition and doubling using w-coordinates for binary Edwards and generalized Hessian curves.

4.1.2 Point Addition and Doubling Using Differential Formulations in w-coordinates

Differential addition [13] is the computation of $Q + P$, given points of Q, P, and $Q - P$. In [1] and [2], the idea of Montgomery's ladder [13] is used to present fast formulas for w-coordinate differential addition on binary Edwards and generalized Hessian curves, respectively. Let us assume w to be a linear and symmetric function in terms of the coordinates x and y of the point P and is defined as $w_i = x_i + y_i$, where $w(P) = w(-P)$. Bernstein *et al.* [1] have defined w-coordinate differential addition for computing $w(Q+P)$ given $w(Q)$, $w(P)$, and $w(Q-P)$. Similarly, the w-coordinates differential doubling is the computation of $w(2P)$ given $w(P)$. Therefore, using w-coordinates of differential addition and doubling formulas, $w((2n + 1)P)$ and $w(2nP)$ can be computed given $w(nP)$ and $w((n + 1)P)$, recursively [1]. In the following, we revisit the differential addition and doubling formulas for binary Edwards and generalized Hessian curves using w-coordinates [1] and [2].

Let $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$ be two affine points on the binary Edwards curve E_{B,d_1,d_2}. Let us define $P_3 = P_1 + P_2 = (x_3, y_3)$, $P_4 = 2P_2 = (x_4, y_4) = (x_2, y_2) + (x_2, y_2)$, and $P_0 = P_2 - P_1 = (x_0, y_0) = (x_2, y_2) - (x_1, y_1)$. Then, one can write $w_3 = w(P_1 + P_2)$, $w_4 = w(2P_2)$, and $w_0 = w(P_2 - P_1)$ as defined above. In the mixed coordinate representation of w_i can be written as the fractions W_i/Z_i in

projective, as $w_1 = w(P_1) = W_1/Z_1$ and $w_2 = w(P_2) = W_2/Z_2$, and w_0 is given as an affine field element. Then, the mixed w-coordinate addition (`MDiffADD`) of these two points can be obtained from [1] as

$$\begin{aligned} &C = W_1 \cdot (Z_1 + W_1), D = W_2 \cdot (Z_2 + W_2), \\ &E = Z_1 \cdot Z_2, F = W_1 \cdot W_2, V = C \cdot D, W_3 = V + w_0 Z_3, \\ &Z_3 = V + (\sqrt{d_1} \cdot E + \sqrt{d_2/d_1 + 1} \cdot F)^2, \end{aligned} \tag{4.4}$$

and the formulas for w-coordinate doubling (`DiffDBL`) [1] are

$$\begin{aligned} &C = W_2 \cdot (Z_2 + W_2), W_4 = C^2, \\ &Z_4 = W_4 + ((\sqrt[4]{d_1} \cdot Z_2 + \sqrt[4]{d_2/d_1 + 1} \cdot W_2)^2)^2. \end{aligned} \tag{4.5}$$

For the generalized Hessian curves, the w-coordinate differential addition formulas can be written as follows [2]

$$\begin{aligned} &A = W_1 \cdot Z_2, B = W_2 \cdot Z_1, C = AB, \\ &U = d^3 \cdot C, Z_3 = (A + B)^2, \\ &W_3 = U + w_0 \cdot Z_3, \end{aligned} \tag{4.6}$$

and similarly for doubling, those are presented as follows [2]:

$$\begin{aligned} &A = W_2^2, B = Z_2^2, C = A + \sqrt{c^3(d^3 + c)} \cdot B, \\ &D = d^3 \cdot B, W_4 = C^2, Z_4 = AD. \end{aligned} \tag{4.7}$$

The costs of different coordinates to compute differential addition and doubling are given in Table 4.1 for binary Edwards [1], generalized Hessian [2], and generic curves [3]. Let **M**, **S**, and **D** be the costs of multiplication of two field elements, a squaring, and a multiplication by a constant curve parameter, respectively. As illustrated in this table, the mixed w-coordinate offers fast and comparable PA and PD formulas. Therefore, we use the mixed w-coordinate differential addition and doubling formulas [1]. Note that the difference of two points for differential addition is given in affine, i.e.,

$w_0 = w(P_2 - P_1)$. Moreover, the mixed w-coordinate addition and doubling formulas are complete which means there is no need to check for the exceptional cases [1]. In order to have efficient computation of point operations, i.e., PAs and PDs, one needs to employ an efficient point multiplication algorithm. In the following section, we give an explanation of using Montgomery's ladder for point multiplication.

4.2 Point Multiplication on Binary Edwards and Generalized Hessian Curves

In this section, we consider Montgomery's ladder [13] and its modified version [3] to present point multiplication algorithm over w-coordinates for binary Edwards, generalized Hessian, and binary generic curves. Using combined PA and PD formulations, we explain how parallelization can increase the performance of point multiplication. At the end, the cost of recovering final coordinates of point multiplication is derived.

4.2.1 Point Multiplication

The elliptic curve point multiplication is defined in the Abelian group as $Q = k \cdot P = P + P + \cdots + P$, ($k$ times), where k is a positive integer, and Q and P are two points on the elliptic curve $Q, P \in E(GF(2^m))$ [3]. The efficiency of point multiplication depends on finding the minimum number of steps to reach kP from a given point $P = (x_0, y_0)$. In binary Edwards and generalized Hessian curves, point multiplication can be defined similar to the one on generic curves [3]. Let P be a point on a binary Edwards curve E_{B,d_1,d_2} and let us assume $w(nP)$ and $w((n+1)P)$, $0 < n < k$ are known. Therefore, one can use the w-coordinate differential addition and doubling formulas to compute their sum as $w((2n+1)P)$ and double of $w(nP)$ as $w(2nP)$.

Among different algorithms to perform point multiplication on elliptic curves, the Montgomery's ladder [13] is widely used in the literature. It has a uniform double-and-add structure which makes it secure against non-differential (simple) side-channel attacks [1], [53]. In [3], an efficient version of Montgomery's algorithm is proposed over $GF(2^m)$. The Montgomery's ladder algorithm for point multiplication using mixed w-coordinates is provided in Algorithm 4.1. As shown in in Step 1 of this algorithm, the point $P = (x_0, y_0)$ is converted to the mixed w-coordinates by computing $w_0 = w(P) = x_0 + y_0$ and setting $W_1 = w_0$ and $Z_1 = 1$. Assume the scalar k is represented in binary, i.e., $k = \sum_{i=0}^{l-1} k_i 2^i$, $k_i \in GF(2)$. Then, the initialization steps, i.e., Steps 1a and 1b of Algorithm 4.1, produce $w(P) = (W_1, Z_1)$

Algorithm 4.1 Montgomery's algorithm [13] for point multiplication using w-coordinates.

Inputs: A point $P = (x_0, y_0) \in E(GF(2^m))$ on a binary curve and an integer $k = (k_{l-1}, \cdots, k_1, k_0)_2$.
Output: $w(Q) = w(kP) \in E(GF(2^m))$.
1: `set` : $w_0 \leftarrow x_0 + y_0$ and initialize
  a: $W_1 \leftarrow w_0$ and $Z_1 \leftarrow 1$
  b: $(W_2, Z_2) =$ `DiffDBL`(W_1, Z_1)
2: **for** i **from** $l-2$ down to 0 **do**
  a: **if** $k_i = 1$ **then**
    i): $(W_1, Z_1) =$ `MDiffADD`$(W_1, Z_1, W_2, Z_2, w_0)$
    ii): $(W_2, Z_2) =$ `DiffDBL`(W_2, Z_2)
  b: **else**
    i): $(W_1, Z_1) =$ `DiffDBL`(W_1, Z_1)
    ii): $(W_2, Z_2) =$ `MDiffADD`$(W_1, Z_1, W_2, Z_2, w_0)$
  end if
  end for
3: **return** $w(kP) \leftarrow (W_1, Z_1)$ and $w((k+1)P) \leftarrow (W_2, Z_2)$

and $w(2P) = (W_2, Z_2)$ using (4.5) [67]. For binary Edward curves, the formulations of (4.4) and (4.5) are implemented in `MDiffADD` and `DiffDBL` functions of Algorithm 4.1, respectively. Therefore, after $l-1$ iterations as presented in Steps 2a and 2b of Algorithm 4.1, the w-coordinates of kP and $(k+1)P$, i.e., $w(kP) = (W_1, Z_1)$ and $w((k+1)P) = (W_2, Z_2)$, will be available. Similarly, for generalized Hessian curves $w_0 = w(P) = 1 + dx_0y_0$, $d \neq 0$ is computed in Step 1 and $(W_1, Z_1) = (w_0, 1)$ is initialized in Step 1a for point multiplication [2]. For this curve, the formulations of (4.6) and (4.7) are implemented in `MDiffADD` and `DiffDBL` functions of Algorithm 4.1, respectively.

4.2.2 Parallelism in Point Multiplication Algorithm

Parallelism is an approach to reduce the number of field arithmetic operations, mainly multiplications, in the critical-path by using multiple multipliers concurrently [10]. In addition, merging point operations, i.e., the PA and PD, can result in less data dependency and can reduce the latency of the point multiplication over binary Edwards and generalized Hessian curves. Computing the w-coordinates of PA and PD for binary Edwards curves together in one step of the Montgomery's algorithm requires six general finite field multiplications and four field multiplications by constants as

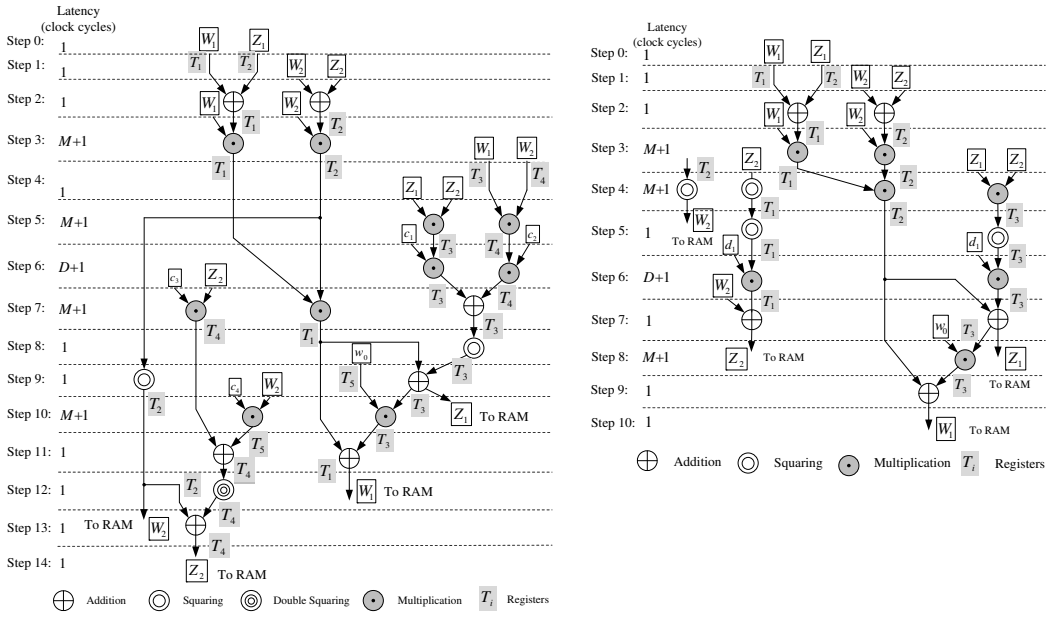


Figure 4.1: Data dependency graphs for parallel computing of the combined PA and PD operations on binary Edwards curves (a): $d_1 \neq d_2$ and (b): $d_1 = d_2$ assuming $\mathcal{M} = 2$. It requires five registers of T_1, T_2, T_3, T_4, and T_5. The constant parameters, $c_1 = \sqrt{d_1}$, $c_2 = \sqrt{d_2/d_1 + 1}$, $c_3 = \sqrt{c_1}$, and $c_4 = \sqrt{c_2}$ are assumed to be precomputed and stored in the memory.

reported in Table 4.1. As summarized in this table, for generalized Hessian curves, the cost of combined PA and PD is five field multiplications and two multiplications by constants [2]. In the following, we explain how parallel field operations can be utilized to reduce the latency of the point multiplication operation.

4.2.2.1 Scheduling Field Operations for PA and PD

We have obtained the data dependency graphs for the combined PA and PD on binary Edwards and generalized Hessian curves as illustrated in Fig. 4.1 (Fig. 4.1a for $d_1 \neq d_2$ and Fig. 4.1b for $d_1 = d_2$) and Fig. 4.2a, respectively. As shown in these figures, the latency (in terms of number of clock cycles) of each step is the latency of an operation with the longest latency. As one can see in Fig. 4.1a and 4.1b, the first four operations of PA, i.e., Step 0 to Step 3, on binary Edwards curve should be performed before any PD operation. This is because computation of PD depends on the PA. For generalized binary Hessian curve (Fig. 4.2a), operations of PA and PD can be performed in parallel at any time. Note that the latency of field additions and field squarings are negligible in comparison to the latency of the

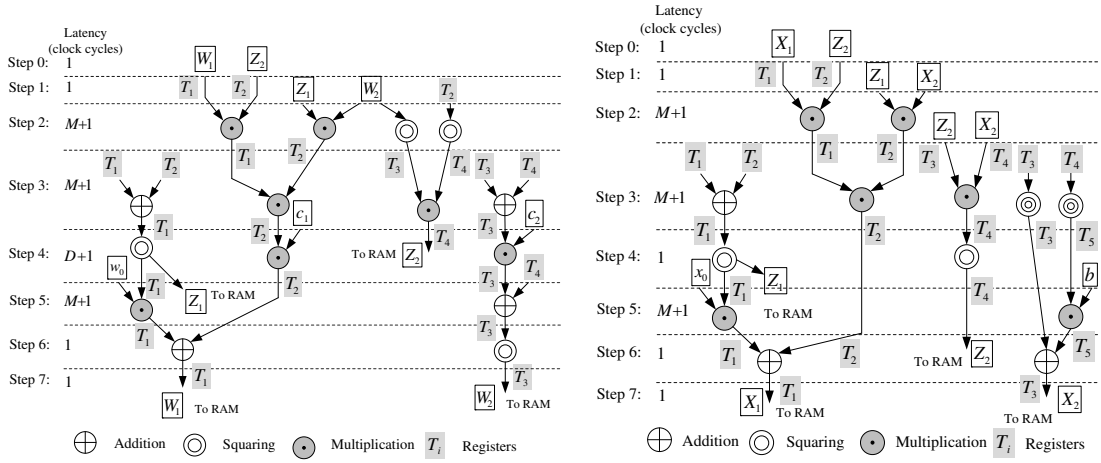


Figure 4.2: Data dependency graph for parallel computing of the combined PA and PD operations for $\mathcal{M} = 2$ available multipliers on (a) generalized Hessian curves, assuming $c_1 = d^3$, and $c_2 = \frac{1}{\sqrt{d^3}}$ and (b) binary generic curves (BGCs) [8].

field multipliers. Therefore, we calculate the latency of the critical-path in terms of number of field multiplications. Let M be the latency (in terms of number of clock cycles) for multiplying two field elements and D be the latency of multiplication of a field element by a constant (e.g., curve parameters, d_1 or d_2). Let us denote $\mathcal{M}$ as the number of parallel finite field multipliers. In the following, we investigate the parallelization using different number of multipliers $\mathcal{M} = 1, 2$ and 3.

4.2.2.2 Parallelization for Binary Edwards Curve (BEC)

For binary Edwards curves with $d_1 \neq d_2$ and one available multiplier ($\mathcal{M} = 1$), the latency of the combined PA and PD is $6M + 4D$ as reported in Table 4.1. Utilizing two multipliers, i.e., $\mathcal{M} = 2$, reduces the latency to $4M + 1D$ and $3M + 1D$ for $d_1 \neq d_2$ (Fig. 4.1a) and $d_1 = d_2$ (Fig. 4.1b), respectively. As one can see in Steps 3, 5, 6, 7, and 10 of Fig. 4.1a, two independent multipliers are fully utilized. Thus, the utilization factor of two multipliers in Fig. 4.1a is 100%. Similarly, in Steps 3, 4, and 6 of Fig. 4.1b, two multipliers are fully utilized. However, in Step 8 of Fig. 4.1b, only one of the two multipliers is utilized (shown in Fig. 4.1b) and the other one is idle (not shown in Fig. 4.1b). Therefore, the utilization factor of two multipliers in Fig. 4.1b is $7/8 \times 100 = 87.5\%$.

If three parallel multipliers, i.e., $\mathcal{M} = 3$, are employed, the latency will become $4M$ and $3M$ for $d_1 \neq d_2$ and $d_1 = d_2$, respectively. Therefore, adding one multiplier only

reduces the latency by one multiplication by a constant. Moreover, one can figure out, the utilization factors for $d_1 \neq d_2$ and $d_1 = d_2$ will reduce to $10/12 \times 100 = 83.34\%$ and $7/9 \times 100 = 77.78\%$, respectively. In addition, employing four multipliers reduces the latency to $3M$ for $d_1 \neq d_2$ and has no impact for the case where $d_1 = d_2$. Note that employing more multipliers, i.e., $\mathcal{M} > 4$, does not decrease the latency. As a result, one can see the maximum utilization of the multipliers with low latency for the combined PA and PD operations is achieved only by choosing $\mathcal{M} = 2$. Multiplier utilization factors for data dependency graph of different curves are summarized in Table 4.2. It is also worth noting that employing two multipliers for the case where $d_1 \neq d_2$, reduces the latency nearly 50% as compared to the case where only one multiplier is utilized.

4.2.2.3 Parallelization for Generalized Hessian Curve (GHC)

For generalized Hessian curve with $\mathcal{M} = 1$, the latency of combined PA and PD algorithm is $5M+2D$. In such a case, the multiplier is always performed the operation and hence the utilization of multiplication for $\mathcal{M} = 1$ is 100%. The data dependency graph for GHC is illustrated in Fig. 4.2a using the combined PA and PD. In this figure, two multipliers, are available, i.e., $\mathcal{M} = 2$. As shown in Steps 2, 3, and 4 of Fig. 4.2a, two multipliers operate in parallel, whereas, in Step 5 only one multiplier performs the multiplication. Therefore, the utilization for $\mathcal{M} = 2$ is $7/8 \times 100 = 87.5\%$. Also, the latency of computing the combined PA and PD operations in parallel is $3M + 1D$. Note that employing three parallel multipliers ($\mathcal{M} = 3$) reduces the latency to $2M + 1D$. However, one can figure out that only in a new step (including combination of Steps 2 and 3 in Fig. 4.2a) all three multipliers will be utilized and in Step 4, i.e., multiplication by constant, only one multiplier will perform the operation and the other two multipliers are idle. As a result, the utilization factor will reduce to $7/9 \times 100 = 77.78\%$. As one can figure out, increasing the number of multipliers from two to three reduces latency only 14% while increasing the required area about 33%.

4.2.2.4 Parallelization for Binary Generic Curve (BGC)

For the sake of comparison, we have included data dependency graph for binary generic curves employing two multipliers $\mathcal{M} = 2$ in Fig. 4.2b [8]. As seen from this figure, the latency of the combined PA and PD operations in parallel is $3M$. Incorporating three multipliers $\mathcal{M} = 3$ reduces the latency to $2M$ with multiplier utilization

Table 4.2: Multiplier Utilization factors for data dependency graph of different curves.

Curve	Utilization factor	
	$\mathcal{M}=2$	$\mathcal{M}=3$
BEC $d_1 \neq d_2$ (Fig. 4.1a)	100%	83.34%
BEC $d_1 = d_2$ (Fig. 4.1b)	87.5%	77.78%
GHC (Fig. 4.2a)	87.5%	77.78%
BGC (Fig. 4.2b)	100%	100%

of 100% [6]. It is worth mentioning that employing more than three multipliers, i.e., $\mathcal{M} \geq 4$, will not reduce the latency of point multiplication. This has been investigated in a different way with $\mathcal{M} = 4$ to parallelize PA and PD operations as well as parallelizing finite field operations in [8]. We note that parallel computation of point multiplication over binary generic curves has been widely studied in the literature, for instance one can refer to [20], [21], [10], [6], [25], and [8].

In the proposed architecture, multiplication by a constant is performed using one of the available multipliers. As a result, its cost is calculated the same as one of a multiplier.

As illustrated in Figs. 4.1 and 4.2, in each step, two words (e.g., W_1 and Z_1 in Step 0 of Figs. 4.1a and 4.1b) are read from the memory as the inputs (it is discussed in details in Section 4.3.3). Consequently, this reduces the memory requirements. Scheduling has been made by two multipliers ($\mathcal{M} = 2$), two adders, and two squarers for efficient implementations. Also, addition and squaring can be performed in one clock cycle and multiplication using digit-level multiplier requires several $M = \lceil \frac{m}{d} \rceil$ clock cycles with an additional clock cycles for loading the inputs. Note that the order of operations are scheduled to achieve optimum number of clock cycles as illustrated in each step of data dependency graphs. At the end of point multiplication (the bottoms of data dependency graphs), the results of PAs and PDs for point multiplication are written to the memory. In what follows, we explain how to recover $Q = kP$ from P, $w(kP)$, and $w((k+1)P)$ at the end of the proposed Montgomery's point multiplication.

4.2.3 Recovering the Final Coordinates of x and y

In this thesis, having w-coordinates in the last step of point multiplication, one can obtain $w(kP) = w_1 = W_1 \cdot Z_1^{-1}$ and $w((k+1)P) = w_2 = W_2 \cdot Z_2^{-1}$. The procedure of recovering the final point from w-coordinates is presented in [1]. At the end of differential addition, one has $w(kP)$, $w((k+1)P)$, and (x, y) for the base point P.

Table 4.3: Latency of the operations in the point multiplication with $\mathcal{M} = 1, 2, 3$, where M is the number of clock cycles required for multiplication of two arbitrary field elements.

Operation	Latency of Point Multiplication Operations			
Curve	BEC [1]		GHC [2]	BGC [3]
Parameter	$d_1 \neq d_2$	$d_1 = d_2$	$c = 1$	
Initialization	$1M + 5$	$1M + 5$	$1M + 3$	5
PA & PD, $\mathcal{M} = 1$	$10M + 21$	$7M + 16$	$7M + 10$	$6M + 10$
PA & PD, $\mathcal{M} = 2$	$5M + 15$	$4M + 11$	$4M + 8$	$3M + 8$
PA & PD, $\mathcal{M} = 3$	$4M + 9$	$3M + 7$	$3M + 9$	$2M + 5$
w-coord/aff $\mathcal{M} = 1$	$22M + 109$	$21M + 104$	$20M + 98$	$19M + 75$
w-coord/aff $\mathcal{M} = 2, 3$	$15M + 105$	$15M + 105$	$15M + 98$	$15M + 74$

First, one needs to check if $w_1^2 + w_1 \neq 0$ and then obtain $x_2^2 + x_2 = A'$ from the equation given in [1]. Since $\mathrm{Tr}(A') = 0$ [1], then employing linear half-trace H: $GF(2^m) \rightarrow GF(2)$ computation over $GF(2^{163})$, one has x_2 or $x_2 + 1$ as the output for polynomial basis. With solving the curve equation for x_2 (or $x_2 + 1$), one can get y_2 (or $y_2 + 1$) whose cost is $I + 13M + 167S + 81A$ for $m = 163$. Note that using normal basis solving the quadratic equation and computing inversion can be performed very efficiently as explained in Chapter 1. Inversion requires $\lfloor \log_2(m-1) \rfloor + HW(m-1) - 1$ multiplications and $m - 1$ squarings, where $HW(m-1)$ is the hamming weight (number of ones) of the binary representation of $m - 1$. Thus, for $m = 163$, the cost of an inversion is $9M + 162S$, where M and S are the costs (in terms of number of clock cycles in our analysis) to perform a finite field multiplication and squaring, respectively. Then, the total cost of recovering (x, y) coordinates of kP as a final point is $22M + 109$ clock cycles.

4.2.4 Latency of Point Multiplication Operations

The latency of point multiplication operations are summarized in Table 4.3 for $\mathcal{M} = 1, 2, 3$. The total latency consists of latencies of initialization ($L_{initial}$), computing PA and PD in the main loop (L_{loop}), and recovering the final point (L_R) for binary Edwards and generalized Hessian curves as follows

$$L_{Total} = L_{initial} + (l - 1) \times L_{loop} + L_R. \tag{4.8}$$

As shown in Table 4.3, M is the number of clock cycles to multiply two field elements as well as a multiplication of a field element by a constant curve parameter. As an

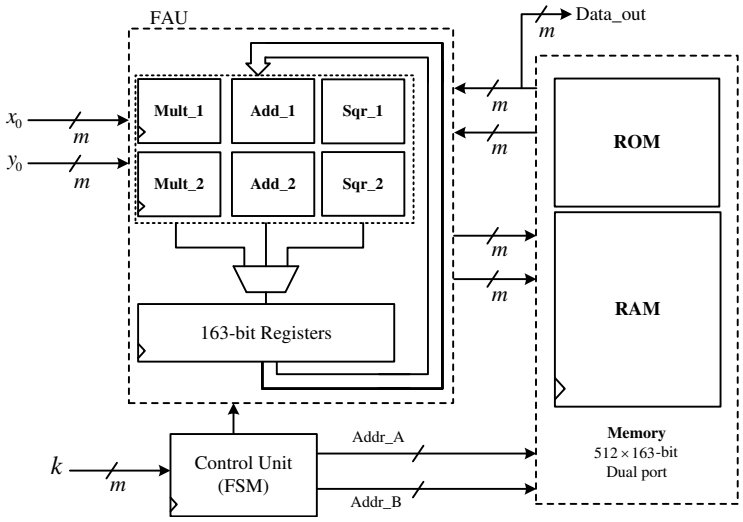


Figure 4.3: Architecture of the proposed elliptic curve crypto-processor for binary Edwards, generalized Hessian, and binary generic curves.

example, the latency of combined PA and PD with $\mathcal{M} = 2$ is calculated from Fig. 4.1a as $5M + 15$, by adding all clock cycles in 15 steps shown in Fig. 4.1a, with an assumption of $D = M$.

4.3 Architecture of the Proposed Elliptic Curve Crypto-Processor

In this section, we propose a hardware architecture for point multiplication over binary Edwards, generalized Hessian, and binary generic curves. A generic structure for the implementation of the point multiplication on FPGA platform is depicted in Fig. 4.3. The architecture is comprised of several blocks: a finite field arithmetic unit (FAU), a control unit and memory. The FAU includes two field multipliers, two adders, and two squarers, as well as five 163-bit registers to store intermediate results. The controller uses program instructions and implements finite state machine (FSM). The memory includes Block RAMs (BRAMs) and ROM to store the intermediate/final results and program instructions. The lower level (finite field) arithmetics are implemented in FAU and higher levels, i.e., PA and PD, are implemented in control logic as a FSM. In the following, we explain these blocks in details.

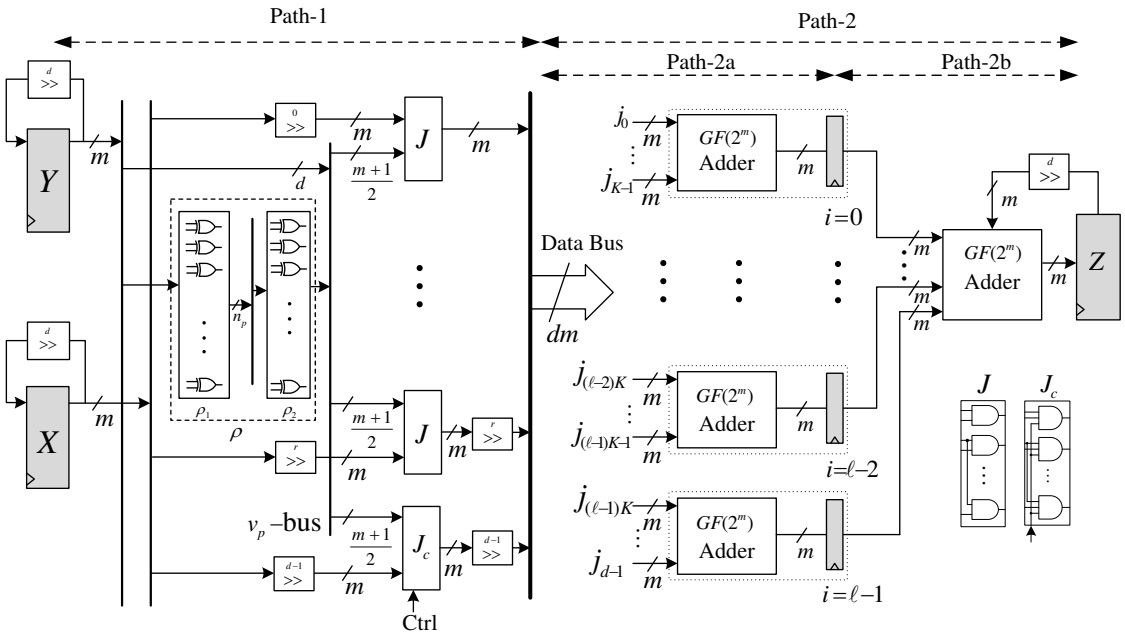


Figure 4.4: The pipelined architecture of the low-complexity type T digit-level GNB multiplier with parallel-output [9].

4.3.1 Field Arithmetic Unit (FAU)

In the binary field with characteristic two, $GF(2^m)$, addition is a bit-wise XOR and can be computed in one clock cycle. In normal basis, squaring of a field element is almost free (in hardware) in terms of both timing and area as it is equivalent to rewiring. The finite field multiplier plays the main role in determining the performance as it dominates the costs of point operations. Therefore, it is essential to design an efficient multiplier.

Bit-parallel multipliers can perform the finite field multiplication in one clock cycle. These multipliers are fast but require a large area complexity. Bit-serial multipliers require m clock cycles for the entire multiplication operation and they are efficient in terms of area but they are slow. Digit-level multipliers are the most suitable ones because the digit-size can be chosen for specific cryptographic applications based on the available resources. In this work, we use a digit-level multiplier which is explained in the following.

4.3.2 A Fast and Low-Complexity Digit-Level GNB Multiplier over $GF(2^m)$

In this subsection, we first present a pipelined low-complexity hardware architecture for digit-level GNB multiplier over $GF(2^m)$. Then, we evaluate the practical time-area efficiency of the presented multiplier by implementing it on a Xilinx® Virtex™-5 FPGA device.

4.3.2.1 Hardware Architecture

Let $A = (a_0, a_1, \cdots, a_{m-1})$ and $B = (b_0, b_1, \cdots, b_{m-1})$ be the field elements represented by type T GNB over $GF(2^m)$. Let $C = (c_0, c_1, \cdots, c_{m-1})$ denote their multiplication, i.e., $C = AB$. Reyhani-Masoleh in [5] has proposed a digit-level GNB multiplier with parallel output and digit-size d, $1 \leq d \leq m$. It requires $M = \lceil \frac{m}{d} \rceil$, $1 \leq M \leq m$, clock cycles to generate all the m coordinates of $C = AB$ simultaneously at the end of the final clock cycle. In [9], a modified and low-complexity version of the digit-level GNB multiplier proposed in [5] is presented. In this section, we pipeline this architecture to make a faster VLSI architecture which operates at very high clock frequencies.

The used pipelined multiplier is depicted in Fig. 4.4. It consists of a ρ block, J blocks in Path-1, and the pipelined $GF(2^m)$ adder in Path-2. The ρ block includes two sub-blocks ρ_1 and ρ_2 and its structure depends on type T, $T \geq 2$, of GNB and multiplication matrix. Each J block consists of m two-input AND gates and each $GF(2^m)$ adder consists of binary trees of XOR gates. As illustrated in Fig. 4.4, the multiplier is pipelined by adding a stage of pipelined registers inside the $GF(2^m)$ adder in order to allow the multiplier to operate at very high clock frequencies. Therefore, instead of performing $GF(2^m)$ addition of dm inputs (as shown in Fig. 4.4), which are connected to the outputs of AND gates in J blocks, we perform the additions in two stages, i.e., over $\lceil \frac{dm}{\ell} \rceil$-inputs. The first stage contains ℓ $GF(2^m)$ adders, each of which has at most $K = \lceil \frac{d}{\ell} \rceil$ m-bit inputs and are depicted by j_0 to j_{d-1} in the architecture. The outputs of the first adders are added with the output of the Z register using another $GF(2^m)$ adder in the second stage. Choosing the optimum value of ℓ plays an important role in designing the fast multiplier. This will be considered later in this section. It is shown in [5] and [9] that the critical-path delay of the non-pipelined multiplier is composed of the delays of the components located in Path-1 and path-2, i.e., $(\lceil \log_2 T \rceil T_X + T_A)$ and $(\lceil \log_2 (d+1) \rceil T_X)$ for $1 \leq d \leq m$, respectively. Note that these are functions of the type of the multiplier T and the digit size d. As shown in

Fig. 4.4, Path-2 is divided into Path-2a and Path-2b by inserting a stage of pipelined registers in between (hereafter we call it ℓ-level of accumulation). This technique reduces the number of logic gates in the critical-path and simplifies the routing.

4.3.2.2 Complexities

In this subsection, we give the number of registers and time complexities of the pipelined digit-level GNB multiplier over $GF(2^m)$. The gate counts of the pipelined multiplier remains the same as the ones of the non-pipelined modified architecture presented in [9]. It requires dm AND gates and $n_p + v_p(\frac{T}{2} - 1) + dm$ XOR gates, where n_p, $n_p \leqslant \min\left\{\frac{v_p T}{2}, \binom{m}{2}\right\}$ [9].

Proposition 4.1. *The pipelined multiplier structure of Fig. 4.4 requires* $(3 + \ell)m$ *registers and its critical-path delay is*

$$\max\left\{\left(T_A + \left(\lceil \log_2 T \rceil + \lceil \log_2 K \rceil\right) T_X\right), \left(\lceil \log_2(\ell + 1) \rceil\, T_X\right)\right\}, \tag{4.9}$$

where ℓ *is the level of accumulation and* $K = \left\lceil \frac{d}{\ell} \right\rceil$.

Proof. As one can see from Fig. 4.4, ℓm registers are required between Path-2a and Path-2b for the pipeline purposes. As a result, the $(\ell + 3)m$ 1-bit registers required in the presented multiplier. The critical-path delay of Path-1, $D_{\text{Path-1}}$ is composed of the delays of the components in Path-1, i.e., T_X, $\left\lceil \log_2 \frac{T}{2} \right\rceil T_X$, and T_A. The delay of Path-2a, $D_{\text{Path-2a}}$ is the delay of an m-bit $GF(2^m)$ adder with at most $K = \left\lceil \frac{d}{\ell} \right\rceil$ m-bit inputs, i.e., $\lceil \log_2 K \rceil T_X$, and the delay of Path-2b, $D_{\text{Path-2b}}$ is $\lceil \log_2(\ell + 1) \rceil T_X$. Therefore, the critical-path delay of the presented architecture is $\max\left\{\left(D_{\text{Path-1}} + D_{\text{Path-2a}}\right), \left(D_{\text{Path-2b}}\right)\right\}$ which completes the proof.

□

The critical-path delay of the pipelined and non-pipelined architecture of the presented multiplier in terms of number of levels of accumulation, ℓ and digit-size, d are illustrated in Table 4.4. It is noted that employing the proposed ℓ-level of accumulation using one stage of pipelined registers increases the latency of the multiplication by one clock cycle to $\left\lceil \frac{m}{d} \right\rceil + 1$.

Lemma 4.1. *The number of feasible accumulators is upper bounded by* $l \leq \left\lceil \frac{d}{2} \right\rceil$ *and is lower bounded by* $l \geq 2$.

Table 4.4: Critical-path delay of the pipelined and non-pipelined architecture of presented digit-level type 4 GNB multiplier over $GF(2^{163})$.

Non-Pipelined [5], [9]		Pipelined			
d	$D_{\text{Path-1}} + D_{\text{Path-2}}$:	K	$D_{\text{Path-2}a}$: $\lceil \log_2 K \rceil T_X$	ℓ	$D_{\text{Path-2b}}$: $\lceil \log_2(\ell+1) \rceil T_X$
$2 \leq d \leq 3$	$T_A + 4T_X$	$2 < K \leq 4$	$2T_X$	$2 \leq \ell \leq 3$	$2T_X$
$3 < d \leq 7$	$T_A + 5T_X$	$4 < K \leq 8$	$3T_X$	$3 < \ell \leq 7$	$3T_X$
$7 < d \leq 15$	$T_A + 6T_X$	$8 < K \leq 16$	$4T_X$	$7 < \ell \leq 15$	$4T_X$
$15 < d \leq 31$	$T_A + 7T_X$	$16 < K \leq 32$	$5T_X$	$15 < \ell \leq 31$	$5T_X$
$31 < d \leq 63$	$T_A + 8T_X$	$32 < K \leq 64$	$6T_X$	$31 < \ell \leq 63$	$6T_X$

Proof. It is clear that from (4.9), the followings should be true in order to achieve the goal of pipelining:

$$\begin{cases} \textit{(a):} \ \lceil \log_2(l+1) \rceil < D_{Path\text{-}1} + \lceil \log_2(d+1) \rceil, & l \geq 1 \\ \textit{(b):} \ \lceil \log_2 k \rceil < \lceil \log_2(d+1) \rceil, & k \geq 2 \end{cases}, \tag{4.10}$$

where k is defined before. From 4.10(a), one can realize that $\lceil \log_2(l+1) \rceil < \lceil \log_2(d+1) \rceil$ and the level of accumulation should be less than the digit-size, i.e., $l < d$, and from 4.10(b) one can get a tighter upper bound for l as $k \geq 2$ and $k < d+1$. The former requires the number of accumulators to be $1 < l < d$ and the latter requires the number of accumulators to be about less than half of the digit-size, i.e., $1 < l \leq \lceil \frac{d}{2} \rceil$. This completes the proof.

□

4.3.2.3 LUT-based Critical-path Delay Analysis

In this subsection, we investigate the critical-path delay of the presented pipelined scheme based on the 6-input programmable look-up tables (LUTs) available in Xilinx® Virtex™-5 FPGA device. To estimate resource consumption and critical-path delay we need to convert the gate-oriented schematics to LUT-based schematics. Then, when the tree of XOR gates are converted into Γ-input ($\Gamma = 6$ in this case) LUT-oriented schematics the $\Gamma - 1$ XOR gates can be replaced by one LUT in the best case. For type $T \leq 4$, each output of the ρ block is obtained by adding (XORing) of T inputs and considering the J block which includes an additional input for the AND operation. Therefore, such outputs can be implemented using 6-input LUTs in $1T_{LUT}$ delay. Then, the LUT-based critical-path delay of the Path-1 is $1T_{LUT}$ for

Table 4.5: LUT-based critical-path delay (CPD) (T_{LUT}) of the presented pipelined multiplier for different digit sizes (d) and levels of accumulation (ℓ) for type 4 GNB multiplier over $GF(2^{163})$ where $K = \lceil \frac{d}{\ell} \rceil$.

d	$D_{\text{Path-1}}$	$D_{\text{Path-2a}}$: $\lceil \log_6 K \rceil$	ℓ	$D_{\text{Path-2b}}$: $\lceil \log_6(\ell+1) \rceil$
$11 \leq d \leq 28$	$1T_{LUT}$	$1T_{LUT}$	$2 \leq \ell \leq 5$	$1T_{LUT}$
$33 \leq d \leq 163$	$1T_{LUT}$	$1T_{LUT}$	$6 \leq \ell \leq 28$	$2T_{LUT}$

type $T \leq 4$. The critical-path delay of Path-2 is summarized in Table 4.5 in terms of different levels of accumulation, ℓ and digit-size d. The critical-path delay of Path-2a and Path-2b are $\lceil \log_6 K \rceil T_{LUT}$ and $\lceil \log_6(\ell+1) \rceil T_{LUT}$, respectively. Therefore, K and ℓ should be chosen in such a way to have a balance for the LUT-based critical-path delay. For example, assume digit-size, $d = 55$ then the critical-path delay of the non-pipelined multiplier is $1T_{LUT} + \lceil \log_6 56 \rceil T_{LUT} = 4T_{LUT}$. Employing $\ell = 10$ levels of accumulation results to have at most $K = \lceil \frac{55}{10} \rceil = 6$ inputs for each $GF(2^{163})$ adders in Path-2a. Then, the critical-path delay of the presented multiplier is $\max\{(1T_{LUT} + \lceil \log_6 6 \rceil T_{LUT}), (\lceil \log_6 11 \rceil) T_{LUT}\} = 2T_{LUT}$. Therefore, for practical implementations one needs to obtain optimum level of pipelining considering number of inputs of LUTs.

In this work, we have proposed an LUT-based pipelining scheme. We have tried several different pipelining techniques including the re-timing scheme of ISE tools but none of them was as efficient as the LUT-based analysis. Therefore, inserting pipelined registers in appropriate locations has a significant impact on the critical path delay of the proposed structure as the $GF(2^m)$ adder of the multiplier has the major critical path delay. In the following subsection, we implement the presented multiplier on FPGA.

4.3.2.4 Implementation

To evaluate the practical performance, the presented pipelined digit-level type 4 GNB PIPO multiplier over $GF(2^{163})$ is implemented on a Xilinx® Virtex™-5 FPGA device. First, feasible values for digit size d are chosen in such a way to decrease the critical-path delay while increasing the area (as a result of upper ceiling). Then, a careful LUT-based with floor-planing design is performed based on the given number of accumulators ℓ and digit-size d. The efficiency of the multiplier is measured in terms of reciprocal of the time-area products, i.e., (time×area)$^{-1}$ and is plotted for different digit sizes d, $11 \leq d \leq 82$, in Fig. 4.5. As shown in this figure, the local optimum

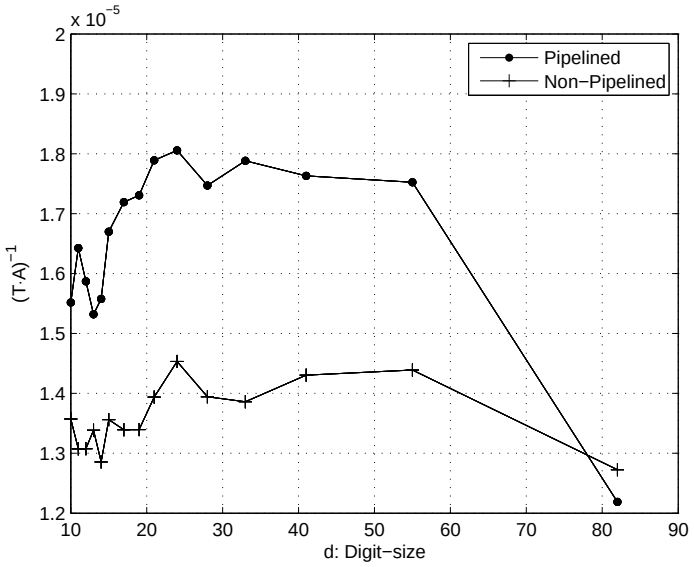


Figure 4.5: Time-Area ratio of the presented pipelined low-complexity digit-level GNB multiplier for type 4 over $GF(2^{163})$ for different digit sizes d.

(for time-area efficiency) in terms of digit sizes for the presented multiplier can be chosen as $d \in \{21, 24, 28, 33, 41, 55\}$. It is noted that two largest digit sizes of $d = 82$ and $d = 163$ degrade the maximum clock frequencies as the place and route (PAR) operation becomes complicated. Therefore, we exclude $d = 163$ from our analysis and keep $d = 82$ for comparison purposes. The presented multiplier is faster (i.e., operates at high clock frequencies) and is smaller than the digit-level MO multiplier employed in [10] for FPGA implementations of ECC over $GF(2^{163})$ [5].

4.3.3 Memory and Control Unit

4.3.3.1 Memory

The proposed architecture requires RAM to store intermediate and variables output as from the FAU and registers and ROM to store program instructions and constant values. As illustrated in Figs. 4.1 and 4.2, in each cycle two words (163-bit) from memory are accessed. Then, dual port BRAMs are configured as two single port BRAMs with independent data access [64]. One can perform two read operations per cycle by using a dual port BRAM. This feature allows us to reduce the number of required BRAMs and achieve greater utilization of this resource. In the utilized Xilinx® Virtex™-5 FPGA device, 36-Kbit (1024, 36-bit words) dual port BRAM blocks are available with a combined 72-bit bus width (36-bit per port). The dual

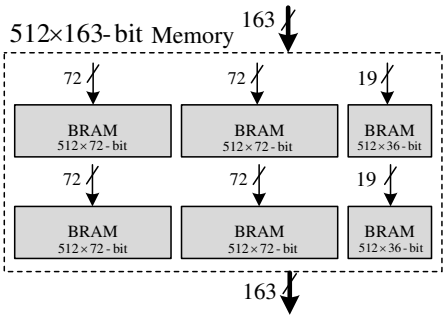


Figure 4.6: Configuration of BRAMs for the proposed architecture.

port RAM is assigned through Xilinx® Synthesis Tool (XST™). In Fig. 4.3, the storage RAM has been designed to allow the reading and writing of the 163-bit words for $m = 163$. This results in minimizing the number of accesses to the memory. Therefore, as shown in Fig. 4.6, the storage RAM is constructed with $\lceil \frac{163\times 2}{72} \rceil = 5$ BRAMs resulting in the storage of 512×163-bit words to store the intermediate inputs as illustrated in the data flow diagrams of Figs. 4.1 and 4.2.

The basic field arithmetic operations, i.e., multiplication, addition, and squaring, are implemented in the FAU. The constants d_1, d_2, c_1, c_2, c_3, and c_4 are stored in the ROM. The ROM to store constants, is implemented with the same BRAM explained above by reserving a few addresses. A register file of 5×163-bit registers (shown by T_i in Figs. 4.1 and 4.2) is incorporated in the FAU to reduce the overhead of the communication between the FAU and the RAM. It is noted that the load and store between the FAU and the memory storage require a single clock cycle. We count all of these clock cycles when calculating the total latency of the point multiplication. The ROM is also generated using Xilinx® BRAMs as illustrated in Fig. 4.3. In Table 4.3, the latency of the operations required to perform arithmetic operations are reported.

4.3.3.2 Control Unit

The control unit of the ECC crypto-processor controls the FAU and memory and it is implemented as a FSM. As shown in Fig. 4.3, the control unit has two address signals, Addr_A and Addr_B, which control the interface between the FAU and the memory. The program instructions are stored in ROM and the control unit fetches and decodes instructions and sends appropriate control signals to the other units based on the presented data dependency graphs of Figs. 4.1 and 4.2. Note that the ROM that stores the program instructions is instantiated using BRAMs as 1024×36-bit words. Therefore, to store program instructions one extra BRAM is required. It

is noted that the control unit decides where to store and conditionally swamp (based on k_i) the results of the combined PA and PD operations.

4.4 Comparisons and Implementations

In this section, we discuss the results obtained in the previous sections and compare them with the counterparts in terms of side-channel analysis and implementation results.

4.4.1 Side-Channel Analysis

As mentioned before, Montgomery's Ladder is highly regular and suitable choice to protect scalar k against simple power analysis attacks [68]. Newly introduced binary Edwards and generalized Hessian curves have two special properties of being unified and complete [1]. The former is that the point addition formulations can be used for point doubling while the latter means that point addition formulations can be used for all pairs of inputs on the curve. Then, the point multiplication algorithm based on unified addition and doubling operations, will not cause side-channel leakage and hence it is protected against side-channel attack (SCA). Baldwin *et al.* [69], have investigated resistivity against simple power analysis (SPA) attacks of the unified operations for twisted Edwards curves over prime fields $GF(p)$. Also, this fact has been investigated in [53] using the unified addition formula of binary Edwards curves. They have also taken advantage of incorporating a simple random order execution (i.e., randomly changing the storage location of the results) in the Montgomery's ladder that makes the differential power analysis (DPA) attack difficult [53]. In this work, we take advantage of completeness of w-coordinates differential PA and PD formulas on Montgomery's ladder which is also SPA resistant.

The cost of explicit point addition is $8\mathbf{M}+5\mathbf{S}+1\mathbf{D}$ for generic curves [55], $13\mathbf{M}+3\mathbf{S}+3\mathbf{D}$ for binary Edwards curves [1], and $8\mathbf{M}+3\mathbf{S}$ for Hessian curves [2]. Therefore, the generalized Hessian curves offer the fastest addition formulas for binary elliptic curves. Although the explicit addition formulas for generic curves are faster than binary Edwards curves, they are not complete and unified. Therefore, one can realize that the cost of one step of point multiplication on binary Edwards curves using explicit addition formulas in [53] is higher than employing Montgomery's differential addition algorithm, i.e., combined differential PA and PD. It is interesting to note that one can reduce this cost by employing explicit addition formulas for generalized

Table 4.6: FPGA implementation results for BECs over $GF(2^{163})$ and $\mathcal{M} = 2$.

d	$M+1$	**Latency** (L_{Total})		$f_{\max}$ (MHz)	**Area**						**P.M. Time** [μs]	
					LUTs		FFs		Slices			
		$d_1 \neq d_2$	$d_1 = d_2$		$d_1 \neq d_2$	$d_1 = d_2$	$d_1 \neq d_2$	$d_1 = d_2$	$d_1 \neq d_2$	$d_1 = d_2$	$d_1 \neq d_2$	$d_1 = d_2$
21	9	10041	7915	269.3	8158	8158	3097	2771	3181	3181	37.2	29.3
24	8	9208	7247	268.8	8750	8750	3423	3097	3371	3371	34.2	26.9
28	7	8375	6579	267.5	10309	10309	3423	3097	4078	4078	31.3	24.6
33	6	7542	5911	265.8	11139	11139	3249	3423	4681	4681	28.3	22.2
41	5	6709	5243	264.5	14235	14235	4075	3749	5788	5788	25.3	19.8
55	4	5876	4575	263.3	17432	17432	5053	4727	6536	6536	22.3	17.3
82	3	5043	3907	196.1	23301	23301	6357	6031	8872	8872	25.7	19.9

Table 4.7: FPGA implementation results for GHC over $GF(2^{163})$ and $\mathcal{M} = 2$.

d	$M+1$	**Total Latency** Clock Cycles (L_{Total})	$f_{\max}$ (MHz)	**Area**			**P.M. Time** [μs]
				LUTs	FFs	Slices	
21	9	7419	272.3	8158	2934	3181	27.2
24	8	6751	271.8	8750	3260	3371	24.8
28	7	6083	269.3	10309	3260	4078	22.5
33	6	5415	268.2	11139	3586	4681	20.1
41	5	4747	267.1	14235	3912	5788	17.7
55	4	4079	266.2	17432	4890	6536	15.3
82	3	3411	196.1	23301	6194	8872	17.3

Table 4.8: FPGA implementation results for BGC over $GF(2^{163})$ and $\mathcal{M} = 2$.

d	$M+1$	**Total Latency** Clock Cycles (L_{Total})	$f_{\max}$ (MHz)	**Area**			**P.M. Time** [μs]
				LUTs	FFs	Slices	
21	9	5884	272.3	8158	3097	3181	21.6
24	8	5383	271.8	8750	3423	3371	19.8
28	7	4882	269.3	10309	3423	4078	18.1
33	6	4381	268.2	11139	3249	4681	16.3
41	5	3880	267.1	14235	4075	5788	14.5
55	4	3379	266.2	17432	5053	6536	12.7
82	3	2878	196.1	23301	6357	8872	14.7

Hessian curves.

4.4.2 Implementation Results and Discussion

We have selected the Xilinx® Virtex™-5 xc5vlx110-2ff1760 device as the target FPGA. In terms of available resources, xc5vlx110-2ff1760 contains 17,280 slices (69,120 LUTs and 69,120 registers), 128 BlockRAMs (BRAMs), and 800 input/output (I/O) pins. Each slice contains 4 flip-flops (FFs) and 4 look-up tables (LUTs) [64].

Choosing Xilinx® Virtex™-5 FPGA would increase the performance and speed of our design. This is mainly due to the availability of 6-input LUTs and large word size in its high 36-Kbit BRAMs. Having 6-input LUTs helps the design to be implemented with fewer logic levels and availability of large word size makes it easier to build large memory arrays (for storing large-bit field elements over $GF(2^m)$) with less routing delay. As a result, using Xilinx® Virtex™-5 FPGAs increases the speed by reducing both the critical-path delay and number of clock cycles (latency). Note that for the comparison purpose, we also implement the proposed design on a Xilinx® Virtex™-4 xc4vlx100 device (which offers four input LUTs) and compared it to the counterparts.

The presented architecture for elliptic curve crypto-processor of Section 4.3 is coded in VHDL and synthesized for different digit sizes d, $d \in \{21, 24, 28, 33, 41, 55, 82\}$ using XST™ of Xilinx® ISE™ version 12.1 design software. The optimization goal for synthesize is set to the default value (i.e., speed). The results of the timing analysis of the implementations after the post place and route are reported in Tables 4.6 and 4.7 for binary Edwards and generalized Hessian curves, respectively. The number of required clock cycles for computing the point multiplication is also presented in these tables for the different digit sizes and different curve parameters, i.e., $d_1 = d_2$, $d_1 \neq d_2$, and $c = 1$. Moreover, the total latencies are found from (4.8) using $l = 163$ as the summation of the required clock cycles for the initialization, the total PA and PD in of the point multiplication, and the conversion as obtained from Table 4.3.

The area requirements are stated in terms of the number of occupied slices (including LUTs and FFs) as reported in Tables 4.6 and 4.7. Note that the proposed architecture for the FAU is the same for binary Edwards (with $d_1 = d_2$ and $d_1 \neq d_2$) and generalized Hessian curves, but they only differ in the control logic provided by instruction program (in ROM) and the number of required registers. Therefore, the area is equal for theses curves as presented in Tables 4.6 and 4.7. The fastest point multiplications are computed for digit size $d = 55$ at approximately 17.3 μs and 15.3 μs for binary Edwards and generalized Hessian curves, respectively. The

proposed architecture requires almost $6,536$ occupied slices ($17,432$ LUTs and $5,053$ FFs) and 6 BRAM blocks for $d = 55$. Similar implementation results are found for binary generic curve as illustrated in Table 4.8.

It is noted that from our implementations results (Tables 4.6, 4.7, and 4.8), one can see that the slices occupation is usually larger than the number of LUTs divided by four ($\frac{\#LUT}{4}$) for Virtex$^{\text{TM}}$-5. It is because the ISE design software starts the unrelated logic packing after the CLB pack factor (100% for the default value) is reached [64]. A higher percentage number will result in lower density packing and a lower pack factor results in a denser design with a difficult place and route and consequently higher delays.

Several implementations of ECC have been published in the literature targeting various applications with different requirements in terms of time-area trade-offs. The implementation results of this work are reported in Table 4.9 and are compared with the results for generic and Koblitz curves available in the literature. We note that because different curves and different FPGA technologies are used to implement different crypto-processors, meaningful quantitative comparisons of the area and time results are difficult. Therefore, as mentioned above we have implemented the crypto-processor for $d = 55$ on Virtex$^{\text{TM}}$-4 device and its area and timing results are reported in Table 4.9. Moreover, as the finite field multiplier plays an important role in determining the performance of an ECC crypto-processor, we discuss the performance results in terms of efficiency of the finite field multiplier and fairly compared them with the counterparts.

It is worth mentioning that in these implementations, we have chosen normal basis as it offers free repeated squarings. Also, we could have taken more advantages of normal basis as it is utilized for Koblitz curves in [10] and [23]. However, by using normal basis, we have eliminated the extra hardware for squarings for the proposed ECC crypto-processor over binary Edwards and generalized Hessian curves. Moreover, recovering final coordinates (x, y) of $Q = kP$ (represented in w-coordinates) requires several repeated squarings and Half-trace computation, that their costs are reduced by using normal basis.

In [10], Järvinen *et al.* have presented the use of parallelization on different levels of point multiplication and have extensively studied the speed and area requirements for NIST B-163 and K-163 curves. For generic curves, the time-area performances are investigated using one, two, and four digit-level MO [35] multipliers over $GF(2^{163})$. As discussed in [5], the area complexity of a digit-level MO multiplier and its improved version is larger than the one presented in this work. Also, as one can realize,

Table 4.9: Comparison of ECC implementations on FPGA over $GF(2^{163})$.

Work[1]	Device	Basis	d	$\mathcal{M}$	Area	Time [μs]
BGC [10]	Stratix II	NB	14	4	11,800 ALMs	48.88
BKC [10]	Stratix II	NB	11	4	13,472 ALMs	25.81
BKC [26]	Stratix II	NB	17	4	23,580 ALMs (26,647 ALUTs, 20575 FFs)	9.48
BGC [10]	Stratix II	NB	41	2	18,489 ALMs	51.56
BKC [10]	Stratix II	NB	41	2	19,498 ALMs	35.1
BGC	Virtex-5	NB	41	2	5,788 Slices (14,235 LUTs, 4,075 FFs)	14.4
BEC ($d_1 \neq d_2$)	Virtex-5	NB	41	2	5,788 Slices (14,235 LUTs, 4,075 FFs)	24.9
BEC ($d_1 = d_2$)	Virtex-5	NB	41	2	5,788 Slices (14,235 LUTs, 3,749 FFs)	19.5
GHC ($c = 1$)	Virtex-5	NB	41	2	5,788 Slices (14,235 LUTs, 3,912 FFs)	17.4
BGC [6]	Virtex-4	NB	55	3	24,363 Slices	10.11
BGC	Virtex-4	NB	55	2	12,834 Slices (22,815 LUTs, 6,683 FFs)	17.2
BEC ($d_1 \neq d_2$)	Virtex-4	NB	55	2	12,834 Slices (22,815 LUTs, 6,683 FFs)	23.3
BEC ($d_1 = d_2$)	Virtex-4	NB	55	2	12,834 Slices (22,815 LUTs, 6,520 FFs)	22.9
GHC ($c = 1$)	Virtex-4	NB	55	2	12,834 Slices (22,815 LUTs, 6,520 FFs)	20.8
BGC	Virtex-5	NB	55	2	6,536 Slices (17,305 LUTs, 4,075 FFs)	12.9
BEC ($d_1 \neq d_2$)	Virtex-5	NB	55	2	6,536 Slices (17,432 LUTs, 5,053 FFs)	22.3
BEC ($d_1 = d_2$)	Virtex-5	NB	55	2	6,536 Slices (17,432 LUTs, 4,727 FFs)	17.3
GHC ($c = 1$)	Virtex-5	NB	55	2	6,536 Slices (17,305 LUTs, 4,890 FFs)	15.3

1. BGC: binary generic curve, BKC: binary Koblitz curve, BEC: binary Edwards curve, GHC: generalized Hessian curve.

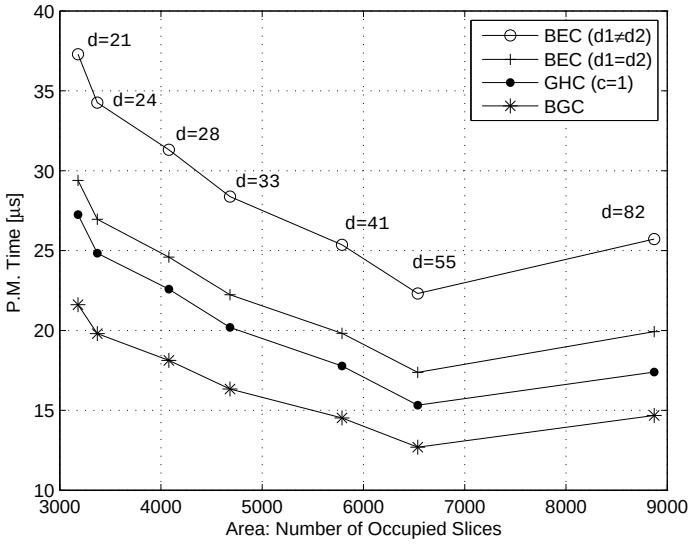


Figure 4.7: Implementation results of point multiplication for binary Edwards, generalized Hessian, and binary generic curves reported in Tables 4.6, 4.7, and 4.8 on Xilinx® Virtex™-5 xc5vlx110-2ff1760 FPGA device. The points are related to digit sizes of $d = 21, 24, 28, 33, 41, 55, 82$.

time complexity of our presented multiplier is less than digit-level MO multiplier as compared in [5]. In addition, we have reached higher clock frequencies with LUT-based pipelining techniques as well. Further, the implementations in ([10], Table VII) for generic curves over $GF(2^{163})$ require higher latency and subsequently larger computation time.

In [26], the same digit-level MO multiplier, has been used for point multiplication on Koblitz curves and has been compared with the results of using polynomial basis. The authors indicated that implementation results using polynomial basis is faster than the ones using normal basis having the same area ([26], Table 4). They have also taken advantage of operation interleaving in their implementations on Koblitz curves. However, it is worth mentioning that the large area consumption of the implementations results of using normal basis in [26] might be as a result of large number of pipelined registers and the implementations results of [26] can be improved using our proposed scheme. Therefore, if one employs our presented multiplier architecture incorporating the techniques proposed in [26], the results of point multiplication using normal basis would be comparable with the ones using polynomial basis. We further note that our implementations are not claimed to be the best possible and faster than counterparts using polynomial basis.

The point multiplication scheme proposed in [6] by Kim *et al.* has been per-

formed on NIST B-163 generic curve employing $\mathcal{M} = 3$ digit-serial GNB multipliers (proposed by Kwon *et al.* in [44]) with Montgomery's ladder on a 4-input Xilinx® Virtex™-4 FPGA. The maximum clock frequency that is reported for the ECC crypto-processor is $f_{\max} = 143$ MHz achieved with digit-size $d = 55$. Therefore, as the multiplier determines the upper bound for critical-path delay, one can estimate that the maximum operating frequency for the multiplier is 143 MHz. However, our presented multiplier operates at $f_{\max} = 196.5$ MHz on Virtex™-4 FPGA with only one level of pipelining. We further note that the proposed LUT-based pipelining technique has significant increase on $f_{\max}$. Moreover, the latency of point multiplication (i.e., the number of clock cycles) in [6] is $L_{Total} = 1 + 162 \times (2M + 2) + 149 = 1446$ employing three multipliers and hence the total time achieved for point multiplication is $T_{kP} = \frac{L_{kP}}{f_{\max}} = \frac{1446}{143} = 10.11$ μs with occupying 24,363 slices. Our implementation on Virtex™-4 FPGA uses only two GNB multiplier and computes a point multiplication in 17.2 μs with using only 12,834 slices as reported in Table 4.9.

Table 4.9 shows a number of related designs (on NIST B-163 and K-163) which are implemented on different FPGA platforms using different types and number of multipliers. To have a fair comparison, we have implemented the ECC crypto-processor based on NIST B-163 generic curve using the presented GNB multiplier for different digit sizes. Data dependency graph of point multiplication of this curve has been illustrated in Fig. 4.2b as its latencies are summarized in Table 4.3. Their implemented results are tabulated in Table 4.8.

In Fig. 4.7, the implementation results are illustrated and point multiplication time is plotted versus area (number of occupied slices). As shown in this figure, increasing the area, as a result of increasing digit-size d, results in faster point multiplications. It is noted that larger digit sizes than 55, i.e., $d > 55$, are not efficient for the proposed architecture as it is seen from Fig. 4.7. Therefore, incorporating multiple smaller multipliers is more efficient than using of a large multiplier. As illustrated in Table 4.8 and Fig. 4.7, our results indicate that the point multiplication over binary generic curve is faster than binary Edwards and generalized Hessian curves. This is because it has smaller latency which requires fewer number of clock cycles.

We further note that the implementations of point multiplication over binary generic curves (short Weierstraß) require special hardware to handle point at infinity. Then, during each point operation, a check should be performed to ensure that the resulting point is not at infinity. It should be noted that the proposed ECC crypto-processor for binary Edwards and generalized Hessian curves works for all the input pairs without any changes (i.e., it is complete). However, exceptional cases should

be tested separately for the case employing NIST generic and Koblitz curves which requires extra hardware and time.

4.5 Conclusions

In this chapter, we have investigated the hardware implementation of point multiplication on binary Edwards and generalized Hessian curve over $GF(2^{163})$ using GNB. We have presented a pipelined version of digit-level GNB PIPO multiplier which operates in higher clock frequencies and studied its time-area trade-offs for different digit sizes. The effect of parallelization using two multipliers for computing the point addition and point doubling on binary Edwards and generalized Hessian curves has been investigated. For point multiplication, the widely-used Montgomery's ladder has been incorporated for differential w-coordinates. The proposed architecture has been implemented on FPGA to obtain the optimum digit-size. Also, we have examined the completeness of the point operations. For binary Edwards and generalized Hessian curves, the fastest point multiplication achieved with choosing $d = 55$. The proposed architecture requires $6,536$ occupied slices ($17,432$ LUTs and $5,053$ FFs), and computes a single point multiplication in 17.3 μs and 15.3 μs for binary Edwards and generalized Hessian curves, respectively. Our implementation results also indicate that the point multiplication over binary generic curve is faster than binary Edwards and generalized Hessian curves. On the other hand, the point multiplication over binary Edwards and generalized Hessian curves is complete. In the next chapter, we propose a new method to reduce the latency of point multiplication on binary Edwards and generalized Hessian curves.

Chapter 5

New Architecture for Double-Multiplication Using GNB and Its Applications for Exponentiation and Elliptic Curve Cryptography

IN this chapter, based on the two low-complexity multiplier architectures proposed in Chapter 3, we present a new digit-level hybrid multiplier which performs two multiplications together with the same number of clock cycles required as the one for one multiplication. It has advantages for high speed finite field arithmetic operations such as exponentiation and elliptic curves point multiplication. The hybrid structure is developed by connecting the output of the proposed digit-level PISO GNB multiplier into the input of a new digit-level SIPO multiplier.

To the best of our knowledge, this is the first digit-level hybrid GNB multiplier which performs two multiplications with the same latency as the one for one multiplier. In order to investigate the applicability of the proposed hybrid multiplier architecture, we employ it for double-exponentiation which is the key operation for Schnorr [70] and ElGamal-type signature verification algorithms [71]. We further note that this scheme can be incorporated to reduce the latency of point multiplication for ECC-based cryptosystems when other schemes (such as parallelization and interleaving) fail due to data dependencies. To obtain the actual implementation results, the proposed hybrid multiplier architecture is coded using VHDL and then implemented

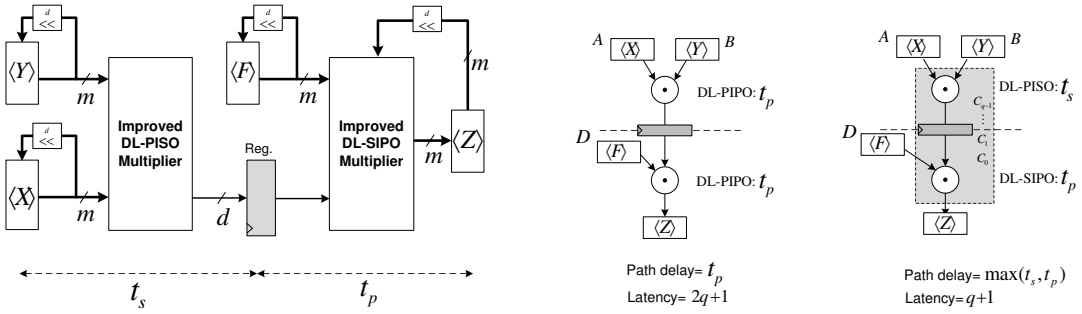


Figure 5.1: (a) Proposed structure for the hybrid multiplier. (b) Two digit-level multipliers with parallel output operating in two separate steps. (c) A hybrid multiplier operating in one step and composed of an improved DL-PISO and an improved LSD-first DL-SIPO multipliers.

on Xilinx® Virtex™-4 field-programmable gate array (FPGA) and synthesized using 65-nm CMOS library of application-specific integrated circuit (ASIC) technology for different digit sizes.

The rest of this chapter is organized as follows. In Section 5.1, the architecture of the proposed hybrid multiplier is presented and its complexities studied for different digit sizes. In Section 5.2, the application of proposed hybrid multiplier are investigated. In Section 5.3, the proposed hybrid multiplier is implemented on FPGA and ASIC and the timing and area requirements are reported. In Section 5.4, we concludes this chapter.

5.1 Hybrid Multiplication

The discussion of the previous chapters dealt with low-complexity and improved DL-PISO and DL-SIPO GNB multipliers. Based on the information provided there, we here present a new hybrid structure by connecting the output of the DL-PISO multiplier to the serial input of the DL-SIPO multiplier. This entire hybrid multiplier performs two multiplications simultaneously, where the results are available in parallel after $\lceil \frac{m}{d} \rceil + 1$ clock cycles assuming that one clock cycle is required to load the output of the first multiplier (stored in the register) to the input of the second multiplier. The structure of the proposed hybrid multiplier is illustrated in Fig. 5.1a. It computes $E = A \times B \times D$ over $GF(2^m)$.

5.1.1 Traditional Multiplication Scheme

The traditional method requires two separate multiplications, one to multiply $A \times B$ and the other one to multiply its result by D. Thus, the latency of computing E is two multiplications if a traditional multiplication scheme is used and its latency can be obtained as follows. In Fig. 5.1b, two digit-level multipliers with parallel output (DL-PIPO) are employed to compute $E = A \times B \times D$, $E \in GF(2^m)$. Let us assume that registers $\langle X \rangle$, $\langle Y \rangle$, and $\langle F \rangle$ are preloaded with the operands A, B, and D, respectively. Also, the register $\langle Z \rangle$ should be initialized with $0 \in GF(2^m)$. The top multiplier (of Fig. 5.1b) requires q clock cycle to compute $C = A \times B$ and store the results to the m-bit register. Also, the bottom multiplier requires q clock cycles to perform $(AB) \times D$ and store it to the register $\langle Z \rangle$. Therefore, to obtain the results in register $\langle Z \rangle$, $2q + 1$ clock cycles are required. It should be noted that the critical-path delay is equal to t_p which is the delay of a digit-level GNB multiplier with parallel output. Then, the required time to compute E is $T = t_p \times (2q + 1)$.

5.1.2 Hybrid Multiplication Scheme

Now, we consider Fig. 5.1c, which depicts the use of a hybrid multiplier which is composed of a digit-level PISO GNB multiplier and a LSD-first digit-level SIPO multiplier. This multiplier performs two dependent multiplications to reduce the latency to the one of one multiplication. Let us assume that $C \in GF(2^m)$ be the product of A and B, i.e., $C = AB$. Based on the output of digit-level PISO multiplier, C will be available from its LSD as $C_0, C_1, \cdots, C_{q-1}$ in each clock cycle. In the first clock cycle it provides the first digit of C, in the order of c_0, followed by $c_1, \cdots$, and c_{d-1}, i.e., $C_0 = (c_0, c_1, \cdots, c_{d-1})$. In the second clock cycle, the bottom multiplier (i.e., DL-SIPO) multiplies the first digit of C, i.e., C_0 by D (stored in register $\langle F \rangle$) and the top multiplier computes the second digit of C, i.e., $C_1 = (c_d, c_{d+1}, \cdots, c_{2d-1})$. Then, one can realize that after $q + 1$ clock cycles, register $\langle Z \rangle$ contains the result of multiplication of $E = A \times B \times D$. The critical-path delay of the hybrid multiplier is equal to the maximum of the delays for the DL-PISO and DL-SIPO multipliers i.e., $t_s = \max\{t_p, t_s\}$, and consequently one can obtain the time of multiplication as $T = t_s \times (q + 1)$.

Based on the information provided above, one can state the following to obtain the complexities of the presented hybrid multiplier.

Proposition 5.1. *The proposed hybrid multiplier architecture requires* $\leq 2v_s(T-1)+2dm-d$ *XOR gates,* $2dm$ *AND gates, four* m*-bit registers and one* d*-bit register. Also,*

its critical-path delay is equal to $T_A + (\lceil \log_2 T \rceil + \lceil \log_2 m \rceil) T_X$ which is due to the delays through logic gates in the path with longer critical-path delay (i.e., DL-PISO architecture).

5.1.2.1 Analysis

In Table 5.1, the latency and time delay of the proposed hybrid multiplier is investigated in terms of different digit sizes for type 4 GNB over $GF(2^{163})$. As shown in this table, the latency, critical-path delay, and time to perform the entire multiplication are given for different digit sizes d, $7 < d < 128$. For the traditional method, i.e., the structure of Fig. 5.1b, the latency is $2q + 1$ while for the hybrid structure, i.e., Fig. 5.1c, the latency is $q + 1$. The time of multiplication for the proposed hybrid structure is $T = (q + 1)T_A + (10q + 10)T_X$ which is about 17% less than the general method for smaller digit-sizes, e.g., $7 < d \leq 15$ and is 38% less while choosing larger digit sizes, e.g., $31 < d \leq 63$. Therefore, the proposed hybrid structure in Fig. 5.1c reduces the latency and consequently the total time of multiplication and is faster than the one depicted in Fig. 5.1b.

5.2 Applications of the Proposed Hybrid Multiplier

The proposed hybrid architecture is particularly applicable for reducing the latency whenever there are repeated multiplications. In this subsection, we provide some of the applications of the proposed hybrid multiplier architecture whenever high speed double-multiplications are required.

5.2.1 Double-Exponentiation

The exponentiation on an Abelian group (e.g., finite fields) is one of the most important arithmetic operations for public key cryptography such as Diffie-Hellman [14] key agreement, RSA, and encoding the Reed Solomon codes [72], [73], and [74]. The exponentiation is usually accomplished by performing repeated field multiplications and squarings [72]. Let A and B be two field elements and K and H be two integers. Then, the computation of $A^K B^H$ (denoted by Double-exponentiation) is a crucial operation for cryptographic applications such as Schnorr- and ElGamal-like signature verifications [70] and [71]. Computing double-exponentiation is presented in [74] by multiplying the result of single exponentiations. Such an scheme is not the most efficient method and efficient computation of double-exponentiation is required.

Table 5.1: Time delay evaluation of the proposed structure for type 4 GNB over $GF(2^{163})$.

digit-size	Structure of Fig. 5.1b			Hybrid Structure in Fig. 5.1c		
	Latency	CPD: t_p	Time	Latency	CPD: t_s	Time
$7 < d \leq 15$	$2q+1$	$T_A + 6T_X$	$(2q+1)T_A + (12q+6)T_X$	$q+1$	$T_A + 10T_X$	$(q+1)\times (T_A + 10T_X)$
$15 < d \leq 31$		$T_A + 7T_X$	$(2q+1)T_A + (14q+7)T_X$			
$31 < d \leq 63$		$T_A + 8T_X$	$(2q+1)T_A + (16q+8)T_X$			
$63 < d \leq 127$		$T_A + 9T_X$	$(2q+1)T_A + (18q+9)T_X$			

Note that $t_p = T_A + (\lceil \log_2 T \rceil + \lceil \log_2 (d+1) \rceil) T_X$ and $t_s = T_A + (\lceil \log_2 T \rceil + \lceil \log_2 m \rceil) T_X$

As explained before, under normal basis representation of field elements squarings are free. Thus, to speed up double-exponentiation one requires to reduce the total number of field multiplications as well as the complexity of each multiplication. The former reduces the latency (in terms of number of clock cycles) while the latter improves the execution time of the multiplier (in terms of propagation delay through logic gates). Based on the discussion regarding low-complexity multipliers presented in the previous sections, we reduce the latency of double-exponentiation using the proposed hybrid multiplier architecture. The following is used in *[73]* to compute the double-exponentiation operation.

Lemma 5.1. [73] *Let A and B be two field elements on $GF(2^m)$ and represented by normal basis and assume K and H be the two positive integers represented by $K = (k_{m-1}, \cdots, k_1, k_0)_2$ and $H = (h_{m-1}, \cdots, h_1, h_0)_2$, respectively. Double-exponentiation of the form $A^K B^H$ is computed by*

$$\begin{aligned} A^K B^H &= A^{k_0 + k_1 2 + \cdots + k_{m-1} 2^{m-1}} B^{h_0 + h_1 2 + \cdots + h_{m-1} 2^{m-1}} \\ &= (A^{k_0} B^{h_0})(A^{k_1} B^{h_1})^2 \cdots (A^{k_{m-1}} B^{h_{m-1}})^{2^{m-1}} \\ &= \left(...(A^{k_{m-1}} B^{h_{m-1}})^2 A^{k_{m-2}} B^{h_{m-2}})^2 ...\right)^2 A^{k_0} B^{h_0}. \end{aligned}$$

The architecture of a multiplexer based double-exponentiation using one multiplier is given in Fig. 5.2a. It is assumed that AB is precomputed *[73]*. As seen in this figure, the result of double-exponentiation is available after $m-1$ iterations, i.e., $(m-1) \times q$, $q = \lceil \frac{m}{d} \rceil$ clock cycles. In Fig. 5.2b, we have proposed a new architecture by employing our proposed hybrid multiplier architecture. This hybrid multiplier performs two multiplications with the latency of one multiplication and as seen the double-exponentiation results will be in the register $\langle Z \rangle$ available after $\lceil \frac{m-1}{2} \rceil$ iterations, i.e., $\lceil \frac{m-1}{2} \rceil \times (q+1)$ clock cycles. This is due to the fact that in each iteration two bits of K, $k_i k_{i+1}$ and H, $h_i h_{i+1}$ are processed from their LSB in parallel. One should note that as the representation of field elements are under normal basis, thus computation of repeated squarings are free. Therefore, our proposed scheme reduces the latency of the double-exponentiation based on choosing efficient values for digit-size d. It is noted that the fast operation is achieved at the expense of extra area. More importantly, one can obtain a trade-off between time and area by choosing suitable values for d. The presented architectures for double-exponentiation can be

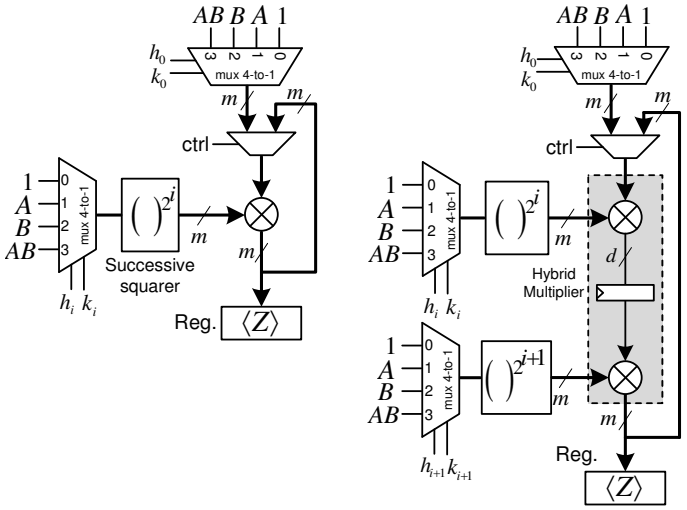


Figure 5.2: Architectures for multiplexer based double-exponentiation. (a) with one multiplier (b) with incorporating the proposed hybrid multiplier.

easily modified to eliminate the multiplication bye 1, i.e., $(1, \cdots, 1, 1)$ in normal basis, whenever h_i and k_i are both zero. However, for the sake of simplicity we do not investigate it here. In [74], a new exponentiation algorithm based on split exponents is proposed. Using normal basis representation and the proposed hybrid multiplier, it can be improved.

5.2.2 Reducing the Latency of Point Multiplication on Binary Curves

In this Section, we employ the proposed hybrid multiplier to perform double-multiplication and reduce the overall latency of point multiplication on binary elliptic curves.

5.2.2.1 Binary Edwards Curves

In Chapter 4, we have proposed a parallel processor for computing point multiplication on binary Edwards curves employing two digit-level multipliers. In binary Edwards curves, mixed w-coordinate has been incorporated to compute mixed differential PA and PD for Montgomery point multiplication with $d_1 \neq d_2$ as given in [1] as:

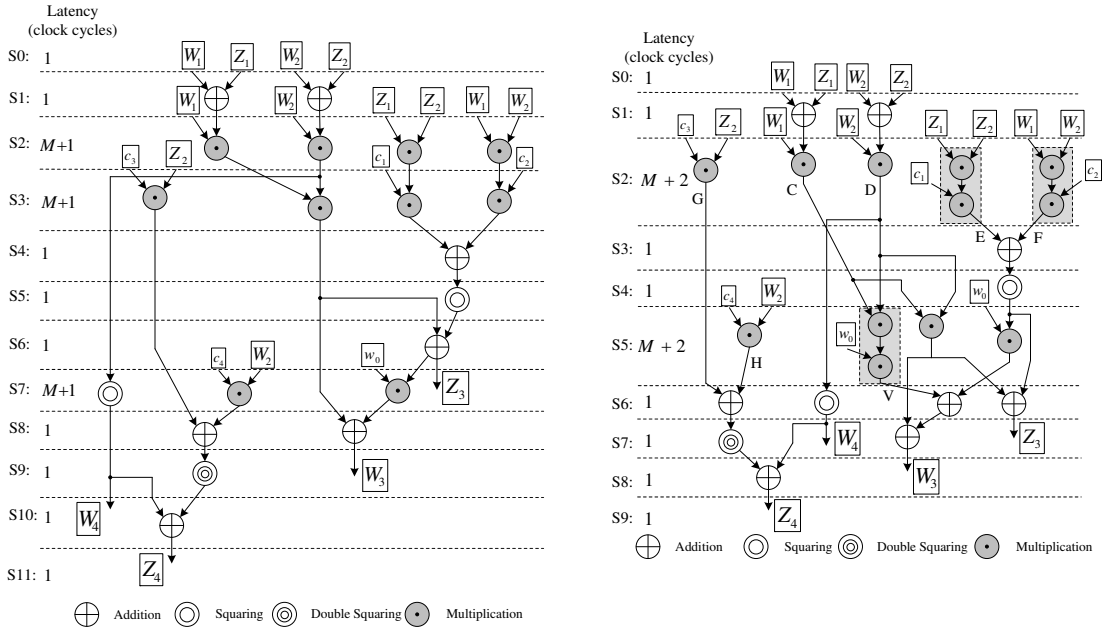


Figure 5.3: Data dependency graph for fast computation of combined PA and PD for binary Edwards curves (a): employing four different PIPO multipliers. (b): employing proposed hybrid multiplier. $c_1 = \sqrt{d_1}$, $c_2 = \sqrt{d_2/d_1 + 1}$, $c_3 = \sqrt{c_1}$, and $c_4 = \sqrt{c_2}$.

$$
\begin{aligned}
&C = W_1 \cdot (Z_1 + W_1), D = W_2 \cdot (Z_2 + W_2), E = Z_1 \cdot Z_2, \\
&F = W_1 \cdot W_2, V = C \cdot D, Z_3 = V + (c_1 \cdot E + c_2 \cdot F)^2, \\
&W_3 = V + w_0 \cdot Z_3, W_4 = D^2, \\
&Z_4 = W_4 + ((c_3 \cdot Z_2 + c_4 \cdot W_2)^2)^2, \qquad (5.1)
\end{aligned}
$$

where $c_1 = \sqrt{d_1}$, $c_2 = \sqrt{d_2/d_1 + 1}$, $c_3 = \sqrt{c_1}$, and $c_4 = \sqrt{c_2}$. As seen from the above formulations, the cost of combined PA and PD operations is $10M$, where M is the cost of a multiplication. For achieving highest degree of parallelization, we employ maximum number of parallel multipliers. The data dependency graph is depicted in Fig. 5.3a employing four DL-PIPO multipliers. In Steps S2 and S3 of Fig. 5.3a four DL-PIPO multipliers are operating in parallel and in Step 7 only two multipliers performed the operation. Therefore, the multiplier utilization is 84%. As one can see, the smallest latency for the combined PA and PD is achieved by employing four multipliers as $3M + 12$. Note that employing more than four multipliers dose not reduce the latency due to data dependencies.

We modify the combined PA and PD formulations in (5.1) in such a way to

incorporate the proposed hybrid multiplier and remove the data dependencies and further reduce the number of multipliers in the data path (i.e., reduce the latency). The modified formulations are as follows

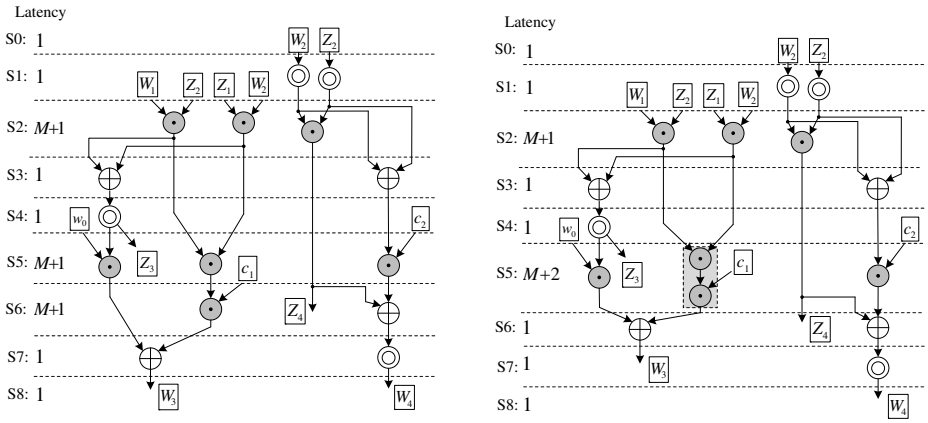
$$
\begin{aligned}
&C = W_1 \cdot (Z_1 + W_1), D = W_2 \cdot (Z_2 + W_2), \\
&E = Z_1 \cdot Z_2 \cdot c_1, F = W_1 \cdot W_2 \cdot c_2, G = c_3 \cdot Z_2 \\
&V = C \cdot D \cdot w_0, Z_3 = C \cdot D + (E + F)^2, H = c_4 \cdot W_2 \\
&W_3 = V + (E + F)^2 \cdot w_0 + CD, W_4 = D^2, \\
&Z_4 = W_4 + ((G + H)^2)^2.
\end{aligned} \tag{5.2}
$$

The corresponding data dependency graph for the modified formulations for combined PA and PD is illustrated in Fig. 5.3b. As shown in this figure, we employed the proposed hybrid multiplier in Steps S2 and S5. In Step S2, we combined computation of field multiplications by constants (c_1 and c_2) and performed them in one step with the latency of $M + 2$ using two hybrid multipliers. Three multipliers regular multipliers are also operating in this step. In Step S5, we modified formulation of the PA operation in computing (W_3 and Z_3) to take the advantage of the hybrid multiplier as much as possible. As one can see, in this step the computation of $V = C \cdot D \cdot w_0$ is done using one hybrid multiplier with the latency of $M + 2$. As a result, the latency of the overall point multiplication over binary Edwards curves is reduced to $2M + 12$. Therefore, applying the proposed technique reduces the latency of computation of combined PA and PD to about 34%. We further note that the proposed approach is a new method to reduce the latency of point multiplication while parallelization fails due to data dependency. Therefore, one can achieves higher speeds in computing of point multiplication for high speed applications mentioned before.

The proposed hybrid structure is also applicable for explicit addition formulas for generic, Hessian, and Koblitz elliptic curves, wherever there is data dependency that limit incorporating parallelization to reduce latency and achieve higher speeds.

5.2.2.2 Generalized Hessian Curves

Similar to binary Edwards curves, mixed w-coordinate has been incorporated to compute mixed differential PA and PD for Montgomery point multiplication as follows [2]:



(a) (b)

Figure 5.4: generalized Hessian curves with $c_1 = d^3$, and $c_2 = \frac{1}{\sqrt{d^3}}$, employing the proposed hybrid multiplier.Generalized Hessian curves

$$
\begin{aligned}
&A = W_1 \cdot Z_2, B = W_2 \cdot Z_1, Z_4 = W_2^2 \cdot Z_2^2 \\
&Z_3 = (A+B)^2, D = W_2^2 + Z_2^2 \\
&E = w_0 \cdot Z_3, F = (A \cdot B), G = D \cdot c_2 \\
&H = F \cdot c_1, W_3 = E + H, W_4 = (Z_4 + G)^2
\end{aligned} \tag{5.3}
$$

where $c_1 = d^3$, and $c_2 = \frac{1}{\sqrt{d^3}}$. As one can figure out the cost of combined PA and PD is $7M$. In Fig. 5.4a, the data dependency graph for combined PA and PD is depicted employing three parallel multipliers. As illustrated in this figure the latency is $3M+9$ and employing more than three multipliers will not reduce the latency. This is the maximum possible number of parallel multipliers that can be used to accelerate the computation of combined PA and PD. However, by employing hybrid multiplier we can reduce the latency to $2M + 10$ as shown in Fig. 5.4b. As one can see, the computation of $A \cdot B \cdot c_1$ is done in one step (Step 5) with the latency of $M + 2$ clock cycles.

5.2.2.3 Binary Koblitz Curves

Jacobian Projective Coordinates

In Jacobian projective coordinates [11], the projective point $(X : Y : Z)$, $Z \neq 0$, corresponds to the affine point $(X/Z^2, Y/Z^3)$ with the projective equation of the curve being $Y^2 + XYZ = X^3 + aX^2Z^2 + bZ^6$. The addition formulas for computing

$P_3 = (X_3, Y_3, Z_3) = (X_1, Y_1, Z_1) + (x_2 + y_2)$ in mixed coordinate cost $10M + 3S + 7A$ with $Z_2 = 1$ as

$$
\begin{aligned}
&B = x_2 Z_1^2, D = y_2 Z_1^2 Z_1, E = X_1 + B, F = Y_1 + D,\\
&Z_3 = EZ_1, H = x_2 F + y_2 Z_3, I = F + Z_3, G = Z_3^2,\\
&X_3 = aG + FI + EE^2, Y_3 = IX_3 + GH,
\end{aligned}
$$

where $a \in \{0, 1\}$. In Fig. 5.5, the data dependency graph for computing point addition on Koblitz curves with mixed coordinates is depicted. In Fig. 5.5a, we have employed three parallel field multipliers to reduce the latency as much as data dependency allows. As one can see, in Steps S5 and S8 three multipliers are operating while in Steps S2 and S11 only two multipliers are operating. Thus, the latency of the point addition is $4M + 13$. As one can realize, employing four or more multipliers does not reduce the latency due to the data dependencies in Steps S5, S8, and S11. In Fig. 5.5b, we have slightly modified the computation of point addition and employed a hybrid architecture to reduce the latency. As seen in this figure, in Step S2 a hybrid multiplier is employed to perform a double-multiplication. Also, in Step S5 hybrid multiplier is used to perform two double-multiplications. Note that in Step S5 we recompute $Z_3 = E \cdot Z_1$ employing another parallel multiplier. However, one eliminate this multiplier and obtain it from the first output of the hybrid multiplier, i.e., DL-PISO. Through employing hybrid technique the latency of mixed point addition on Koblitz curves with Jacobian coordinates reduced to $3M + 14$ which is the smallest one that has been achieved in the literature.

5.2.2.4 Attacking ECC2K-130

In [75], Fan *et al.* have performed an extensive investigation to solve one of the Certicom elliptic curve discrete logarithm problem (ECDLP) challenges, ECC2K-130 using Pollard's rho method [76]. They have focused on Koblitz curves over $GF(2^{131})$ and because of performing several squarings, normal basis is incorporated as the Hamming weight of x-coordinate is also represented with this basis [75]. Each iteration of their method requires five multiplications that can not be reduced by employing parallel multipliers due to data dependencies. However, our proposed hybrid multiplier for GNB (for type 2) can be incorporated to reduce the latency of each iteration to four multiplications and improve the overall speed of the attack.

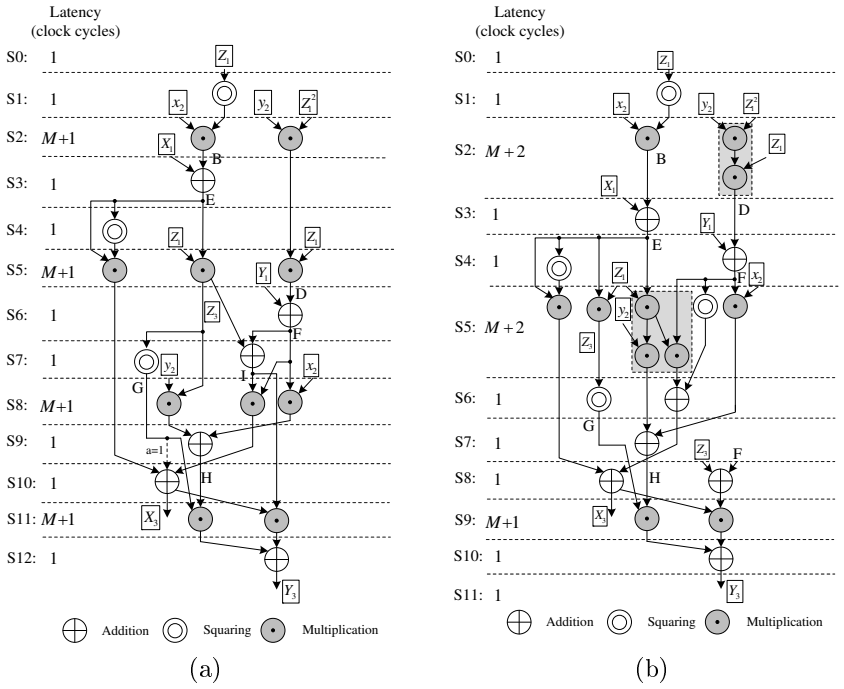


Figure 5.5: Parallel computation of point addition on Koblitz curves using Jacobian coordinates (a): with three finite field multipliers and (b): employing hybrid multiplier and three parallel multipliers.

5.3 Implementations

In this section, to study the time and area requirements of the proposed hybrid multiplier we implemented it on Xilinx® Virtex™-4 xc4vlx100-ff1148 FPGA and 65-nm Complementary Metal-Oxide-Semiconductor (CMOS) library for the synthesis on application-specific integrated circuit (ASIC) technology. The proposed hybrid architecture for double-multiplication is modeled in VHDL and synthesized for different digit sizes using XST™ of Xilinx® ISE™ version 12.1 design software and Synopsys® Design Vision® which is a GUI for Synopsys® Design Compiler® tools. The implementation results are reported in Table 5.2 for different digit sizes over $GF(2^{163})$. The correctness of the multiplier architectures is verified by Xilinx® ISE™ Simulator (ISim). For the FPGA implementations, the optimization goal is set to the speed (i.e., default) and optimization effort is set to normal and the area (Slices, LUTs, and FFs) and timing (ns) for the critical-path delays (CPD) are obtained for different digit sizes. It is noted that the results of the implementations on FPGA, are all after post place and route results. For the ASIC implementations, the map effort is set to medium with a target clock period of 5 ns and the area (μm^2) and timing (ns) are obtained for each of the designs.it on ASIC the proposed hybrid multiplier

architecture

5.4 Conclusion

In this chapter, for the first time we proposed a digit-level hybrid multiplier over GNB which performs two multiplications with the same latency as the one for one multiplier proposed in the literature. We employed the proposed hybrid architecture to reduce the latency of double-exponentiation. The analyzes results indicate that the proposed hybrid multiplier architecture reduces the latency of double-exponentiation about 50%. Moreover, we employed the hybrid multiplier architecture to reduce the latency of point multiplication on binary Edwards, generalized Hessian, and Koblitz curves. It is shown that the proposed scheme reduces the latency of point multiplication about 33% for both binary Edwards and generalized Hessian curves and 25% for Koblitz curves using Jacobean coordinates. Therefore, the point multiplication on binary Edwards and generalized Hessian curves are competitive with the binary generic curves using our hybrid multiplier and provide completeness for input points. It is worth mentioning that the proposed architecture is suitable for the applications when fast computations of point multiplication is desired.

Table 5.2: ASIC and FPGA implementation results for the proposed low-complexity hybrid multiplier architecture (Fig. 5.1) over $GF(2^{163})$ for different digit sizes.

digit size	Latency $q+1$	ASIC			FPGA				
		Area [μm^2]	CPD [ns]	Time [ns]	# Slice	# FF	# LUT	CPD [ns]	Time [ns]
11	16	69,048	1.38	22.08	3753	677	7321	5.7	91.2
21	9	127,053	1.85	16.65	6753	705	13,258	6.5	61.2
33	6	195,170	2.37	14.22	11,306	811	21,023	6.8	40.8
41	5	242,096	2.87	14.35	14,589	724	25,928	6.9	34.5
55	4	321,692	3.65	14.60	19,030	763	34,118	7.4	29.6

Chapter 6

Highly Parallel and Fast Crypto-Processor for Point Multiplication on Koblitz Curves

IN this chapter, based on the DL-PIPO GNB multiplier architecture proposed in chapter 3, we propose a highly parallel an fast crypto-processor for point multiplication on Koblitz curves. Binary Koblitz (or anomalous) curves, are special class of binary generic curves that point multiplication can be efficiently computed using special properties for these curves. These curves employ Frobenius map (instead of doubling) and point addition operation using projective mixed coordinates for computing point multiplication. The binary Koblitz curves are specified in NIST [19], IEEE [18], and SEC2 [77] as the mostly standardized and specified binary curves for different levels of security depending on the availability of the resources. In the recent past, considerable efforts have been made to accelerate the computation of point multiplication over binary elliptic curves. Those include parallelization [78], [6], and [10], interleaving [79], [26], and pipelining [80]. The two former techniques are used to reduce the latency of the computation, whereas the latter is used to increase the maximum operating clock frequency. In this chapter, we employ parallelization and efficient pipelining in our implementations for high speed applications.

Parallelization is a well-known approach to accelerate the ECC computations, employing multiple parallel field arithmetic units (mainly multipliers) in the lower level, i.e., finite field computations, for instance one can refer to [81], [78], and [82]. It is worth mentioning that in case of dependencies amongst lower level computations, achieving parallelization is a challenging task and employing more than certain number of parallel arithmetic units will not increase the speed of ECC computations.

Recently, several methods to perform parallel computations for point addition on Koblitz curves have been proposed [83], [10], and [26]. It has been claimed that the maximum number of the finite field multipliers to achieve the highest parallelization in computing point multiplication on Koblitz curves is three parallel finite field multipliers. However, here we modify the point addition formulation in such a way to employ four multipliers to reduce the latency of point addition. This techniques will increase the overall speed of point multiplication on Koblitz curves. To do so, we first perform data-flow analysis for ECC computations to understand how data has to move between the different logic and computational elements such as field multipliers, adders, and squarers. Then, we perform a latency analysis to determine where potential bottlenecks may occur and then find a balance between desired performance and the cost of implementing the design. In this effect, we modify the point addition formulation to employ four parallel finite field multipliers to reduce the latency of point multiplication about 25%.

For investigating the practical performance of the proposed architecture, we implement it on FPGA for different digit sizes over $GF(2^{163})$ targeting the applications where high speed is required and area usage should be considered as well. It is noted that our method can be applied to any finite field representation and for the sake of efficient implementation and comparison, we use GNB in this chapter.

The rest of this chapter is organized as follows. In Section 6.1, properties of Koblitz curves and arithmetic on these curves are presented. In Section 6.2, parallel computation of point multiplication is investigated. In Section 6.3, the hardware architecture of proposed crypto-processor on Koblitz curves is presented. In Sections 6.4, the implementation results for proposed architecture on FPGA are presented. Finally, we conclude this chapter in Section **??**.

6.1 Properties of Koblitz Curves

In finite field of characteristic two, Frobenius map ϕ is an endomorphism that raises every element to its power of two, i.e., $\phi : x \rightarrow x^2$. The squaring over $GF(2^m)$ using GNB is a free operation in hardware. Then, Frobenius endomorphism can be carried out efficiently (cost free) if the elements of finite field are represented in normal basis [11]. Koblitz [84] showed that point doublings can be performed efficiently by utilizing the Frobenius endomorphism if the binary curve is defined over $GF(2)$ as

$$E_{K,a}/GF(2^m): \ y^2 + xy = x^3 + ax^2 + 1, \tag{6.1}$$

and $a \in \{0, 1\}$. Then, the Frobenius map can be defined as

$$\begin{aligned} \phi : & E(GF(2^m)) \rightarrow E(GF(2^m)) \\ & (x, y) \rightarrow (x^2, y^2), \end{aligned}$$

and one can show that

$$\phi^2(P) - \mu\phi(P) + 2P = 0 \text{ for every } P \in E_{K,a}(GF(2^m)).$$

Let τ be the complex root of $P(T) = T^2 - \mu T + 2$ which is the characteristic polynomial of the Frobenius endomorphism. Then, if one represent the scalar k in τ-adic NAF (τNAF), i.e., $k = \sum_{i=0}^{l-1} k_i \tau^i$ for $k_i \in \{0, 1, -1\}$ and $k_i k_{i+1} = 0$, then point multiplication can be computed as $kP = \sum_{i=0}^{l-1} k_i \tau^i(P)$ [11]. It results in the hamming weight of τNAF to be the same as the one of the binary NAF, i.e., $\approx (\log_2 k)/3$, and its length to be approximately $2m$ which is twice the length of the binary NAF. Since $(\phi^m - 1)P = \phi^m P - P = P - P = \mathcal{O}$ stands for all $P \in E_{K,a}(GF(2^m))$, Solinas [85] proposed a method that if $k' \equiv k (\text{mod } \delta)$, $\delta = (\tau^m - 1)/(\tau - 1)$, then $k'P = kP$ and the length of the τNAF over remainder of k can be reduced to m. Recently, efficient hardware architectures for τNAF conversion have been proposed in [86], [87], and [88].

In normal basis when $P = (x, y)$ is known, $\tau^i(P)$ can be computed by i-fold right cyclic shifts of the x and y coordinates representing P, i.e., $\tau^i(P) = (x^{2^i}, y^{2^i}) = (x \gg i, y \gg i)$. As $2P = -\tau^2(P) + \mu\tau(P)$, then the point doubling operation requires two squarings and a point addition. The faster computation of $\tau(P) = (x \gg 1, y \gg 1)$ in normal basis results in a faster point multiplication of $Q = kp = \sum_{i=0}^{m-1} k_i \tau^i(P)$ than the traditional methods [89].

6.1.1 Point Addition on Koblitz Curves

Point addition on Koblitz curve can be performed using different coordinate systems such as, Jacobian, standard projective, and Lopez-Dahab projective coordinates. Among them Lopez-Dahab coordinate system provides efficient point addition formulation as coming in the following.

6.1.1.1 Lopez-Dahab Projective Coordinates

For Lopez-Dahab coordinates, [3] the triple coordinates (X, Y, Z) is used to represent $(X/Z, Y/Z^2)$ in affine when $Z \neq 0$ and $\mathcal{O} = (1, 0, 0)$. The curve equation in this coordinate is

$$Y^2 + XYZ = X^3Z + aX^2Z^2 + bZ^4,\ a, b \in GF(2^m),$$

and the cost of point addition and doubling is $13\mathbf{M} + 4\mathbf{S} + 9\mathbf{A}$ and $5\mathbf{M} + 4\mathbf{S} + 5\mathbf{A}$, respectively. Note that $\mathbf{M}$, $\mathbf{S}$, and $\mathbf{A}$, are the costs of multiplication, squaring, and addition, respectively. In Lopez-Dahap coordinates where one of the points represented in affine, the cost of mixed projective point addition, i.e., $(X_3, Y_3, Z_3) = (X_1, Y_1, Z_1) + (x_2, y_2)$, reduces to $9\mathbf{M} + 5\mathbf{S} + 9\mathbf{A}$ [55].

The explicit formulation are given as follows [55]:

$$\begin{aligned}
Z: &\begin{cases} A = Y_1 + y_2Z_1^2, B = X_1 + x_2Z_1; \\ C = BZ_1, \\ Z_3 = C^2, \end{cases} \\
X: &\begin{cases} D = x_2Z_3, \\ X_3 = A^2 + C(A + B^2 + aC), \end{cases} \\
Y: &Y_3 = (D + X_3)(AC + Z_3) + (y_2 + x_2)Z_3^2
\end{aligned} \tag{6.2}$$

where $a \in \{0, 1\}$ for Koblitz curves and hence its cost reduces to $8\mathbf{M} + 5\mathbf{S} + 9\mathbf{A}$.

The binary Koblitz curves `sect163K1` with $a = 1$ [11], is specified in SEC2 [77] as the mostly standardized and specified binary curve at the 83-bit security level.

6.1.2 Point Multiplication on Koblitz Curves

The algorithm for computing point multiplication i.e., $Q = kP$ on Koblitz curves is given in Algorithm 6.1, where the scalar k is presented in τNAF [11]. This algorithm requires on average $m - 1$ Frobenius maps and $m/3 - 1$ point additions or subtractions. Since, Frobenius maps can be computed with free squarings in normal basis, the computation of point addition determines the efficiency of point multiplication. Therefore, our main focus is on high performance computation of point multiplica-

Algorithm 6.1 Point multiplication on Koblitz curves using Double-and-add-or-subtract algorithm [11].

Inputs: A point $P = (x, y) \in E_K(GF(2^m))$ on curve
and integer k, $k = \sum_{i=0}^{l-1} k_i \tau^i$ for $k_i \in \{0, \pm 1\}$.
Output: $Q = kP$.
1: **initialize**
  a: **if** $k_{l-1} = 1$ then $Q \leftarrow (x, y, 1)$
  b: **if** $k_{l-1} = -1$ then $Q \leftarrow (x, x + y, 1)$
2:**for** i **from** $l - 2$ **downto** 0 **do**
 $Q \leftarrow \phi(Q) = (X^2, Y^2, Z^2)$
 if $k_i \neq 0$ **then**
  $Q \leftarrow Q + k_i P = (X, Y, Z) \pm (x, y)$
 end if
end for
3: **return** $Q \leftarrow (X/Z, Y/Z^2)$

tion employing multiple efficient digit-level finite field multipliers. In the following we study the parallelization of point addition on Koblitz curves.

6.2 High-Speed Parallelization of Point Addition

Parallelization for hardware implementation of point addition on Koblitz curves has been investigated employing different number of field multipliers in [10], [78], [82], and [81]. In [10], it is shown that employing two finite field multipliers reduces the number of multipliers (and hence the latency of ECC point multiplication) in the data path to five multiplications. Also, it is shown in [10] that the maximum number of parallel finite field multipliers that can be employed to implement the fastest point multiplication is three. It is shown that employing three parallel finite field multipliers reduces the number of multipliers in the longest data path to four multipliers. The data dependency graph for point addition employing three multipliers is depicted in Fig. 6.1a [10]. As one can see, the latency of point addition is $4M + 13$, where M is the latency for a multiplication. In Step S4 only one multiplier is operating and the other two multipliers are idle. This is mainly because of the dependency of computing C to B (as shown in 6.2). This does not allow us to compute B and C in parallel. As seen from Fig. 6.1a, a potential bottleneck occurs in computing C which uses only one multiplier in Step S4. This results in 66% multiplier utilization for the data

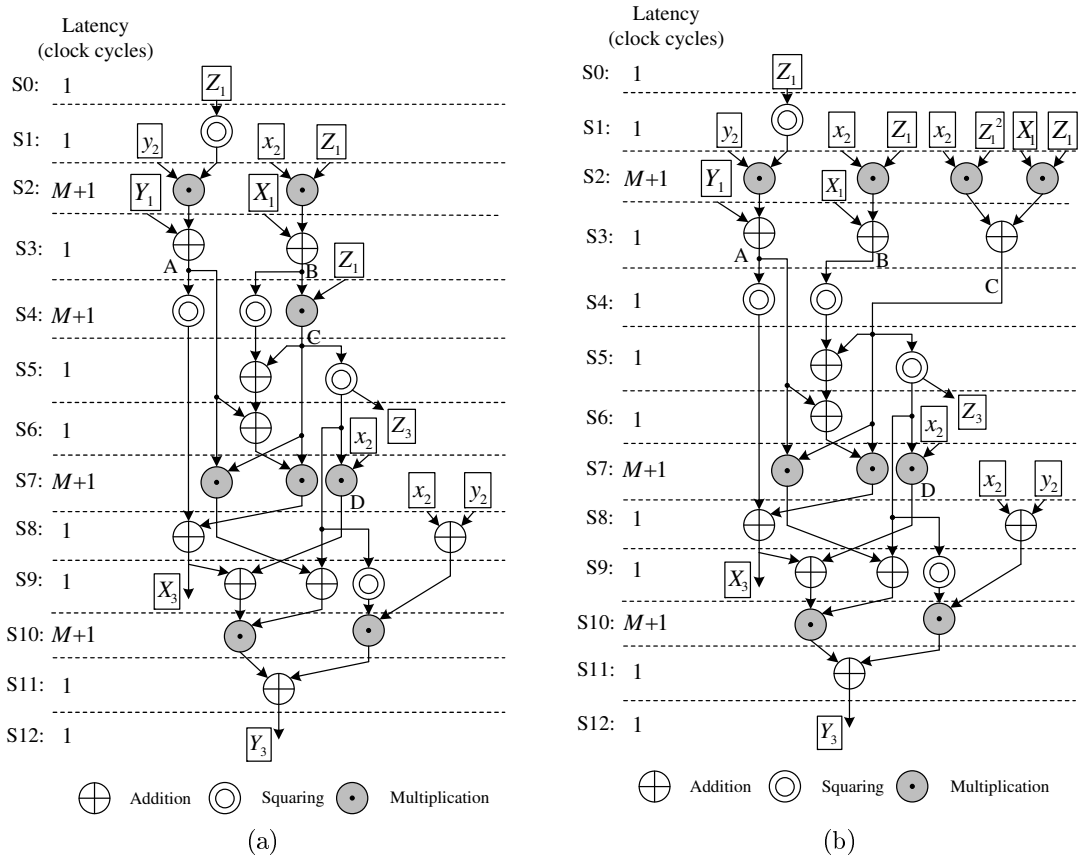


Figure 6.1: Data dependency graph for parallel computation of point addition on Koblitz curves (a): using three finite field multipliers adopted from [10] (b): proposed scheme employing four multipliers.

dependency graph presented in Fig. 6.1a employing three parallel multipliers.

The formulation of point addition [55] can be modified to employ one additional parallel multiplication to reduce its latency as stated in the following proposition.

In computing the Z coordinate of the point addition formulation of (6.2), the data dependency in computing C can be eliminated by the following

$$Z : \begin{cases} A = Y_1 + y_2 Z_1^2, B = X_1 + x_2 Z_1, \\ C = x_2 Z_1^2 + X_1 Z_1, Z_3 = C^2, \end{cases} \tag{6.3}$$

As one can see from (6.3), computation of C can be performed in parallel with B at the cost of employing one more multiplier as compared to the formulation presented in (6.2). Therefore, we can employ four multipliers in parallel to compute point addition. The data dependency graph for computing point addition based on (6.3) is depicted in Fig. 6.1b which employs four parallel multipliers. As one can see, in Step S2 of Fig. 6.1b four multipliers are operating in parallel. Therefore, the multiplication in Step S4 in Fig. 6.1a is eliminated. As seen in Fig. 6.1b, the number of field multipliers in the data path is reduced to three multipliers with the overall latency of $3M + 13$ clock cycles. Therefore, employing four parallel multipliers results in 25% reduction in the latency in comparison with the case where three multipliers are employed. Note that the multipliers utilization is increased to 75%, as 9 out of 12 multiplications are performed using four multipliers. Our presented approach reduces the latency of the point addition using four field multipliers and consequently speeds up the point multiplication as explained before.

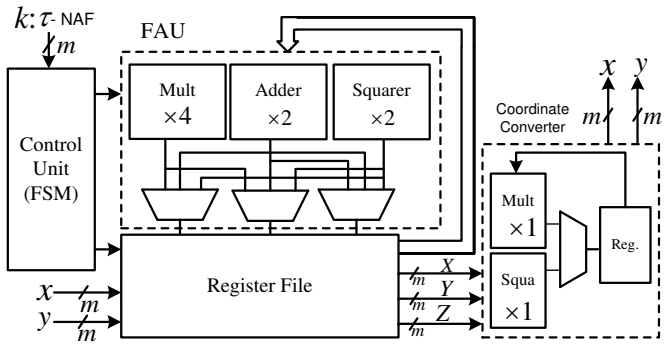
6.2.1 Latency of Point Multiplication

The point multiplication on Koblitz curves composed of three main blocks: τNAF converter, the main processor (addition and Frobenius map), and the coordinate converter. In [88], an efficient circuitry is presented for τ-NAF conversion which requires $m + 6$ clock cycles for $m = 163$. Also, the latency of coordinate conversion from projective Lopez-Dahab to affine is $11M + 11$ based on Itoh-Tsujii method [38]. Since these latencies are the fixed for all implementations, we only compare the latency for the main processor in computing point additions as given in Table 6.1. We assume that two adders and two squarers are available based on the data dependency graph depicted in Fig 6.1b. In this table, $H(k)$ is the Hamming weight of τ-NAF expansion of k and the total latency of point addition is computed by multiplying the number of

Table 6.1: Comparison of the latency for performing point addition in the main loop on Koblitz curves in terms of number of multipliers .

# of Multipliers	E_K [10]	This work
4	$(H(k)-1)(4M+13)$	$(H(k)-1)(3M+13)$

Figure 6.2: The architecture of point multiplication crypto-processor



non-zero terms in k to the latency of a point addition. the As shown in Table 6.1, for higher speed implementations our proposed data dependency graph provides smaller latency in comparison to the others assuming to have equal cost for Frobenius maps. We note that if one employs polynomial basis to represent field elements, the cost of Frobenius map should be considered as well.

6.3 Proposed Crypto-processor for Point Multiplication

In this section, we present a hardware architecture for point multiplication on Koblitz curves. The architecture of the crypto-processor is depicted in Fig. 6.2. As one can see, it consists of a field arithmetic unit (FAU), register file, coordinate converter, and a control unit. The registers are to store point coordinates, intermediate and final values during point additions. In the following, we explain how the proposed architecture operates and produces the point multiplication results for a given point P and scalar k represented in τNAF.

6.3.1 Field Arithmetic Unit (FAU)

The FAU performs four basic arithmetic operations employing: four digit-level GNB multipliers, two $GF(2^m)$ adders, and two squarers. Multiplication in $GF(2^m)$ plays

the main role in determining the efficiency of the point multiplication in the crypto-processor. Finite field multipliers are available in bit-level (with area complexity of $O(m)$ and time complexity of $O(m)$), digit-level (with area complexity of $O(md)$ and time complexity of $O(m/d)$), and bit-parallel (with area complexity of $O(m^2)$ and time complexity of $O(1)$) architectures depending on the available resources. We employ a low-complexity and pipelined digit-level parallel-in parallel-out GNB multiplier presented in Chapter 3. Recall that in a digit-level parallel-in parallel-out GNB multiplier both input operands, A and B should be present through multiplication process and the results will be available in parallel after $M = \lceil \frac{m}{d} \rceil$ clock cycles. Thus the latency of the multiplier (in terms of clock cycles) is given by $M = \lceil \frac{m}{d} \rceil + 1$, $1 \leq d \leq m$ considers one clock cycle for one level of pipelining. For the given field size $m = 163$ (which is type 4 GNB), digit-size d is chosen in such a way to reduce the latency while increasing d. Therefore, we choose the digit sizes from the set $d = \{11, 21, 33, 41, 55\}$ for $m = 163$. We note that the finite field multiplier determines the time and area requirements of the point multiplier of the crypto-processor. A digit-level version of Massey-Omura multiplier [35] is investigated for FPGA implementation of ECC in [90], [91], [23], [26], and [10] on Koblitz curves. In terms of area complexity, Massey-Omura multiplier requires dm AND gates and $dT(m-1)$ XOR gates and its critical-path delay is $T_A + (\lceil \log_2 T \rceil + \lceil \log_2 m \rceil) T_X$ for type T GNB. Note that our employed multiplier in this work requires smaller area in comparison to the counterparts used in [91], [23], [26], and [10]. The $GF(2^m)$ adder uses m XOR gates to perform the addition and requires only a clock cycle to store the results in the registers. The squarer is simple rewiring in normal basis and requires a clock cycle to store its results in the registers. Note that Frobenius map is performed for coordinates of X, Y, and Z, independently.

6.3.2 Control Unit and the Register File

The control unit is designed with a finite state machine (FSM) to perform the point multiplication with other units. First, the coordinates of $P = (x, y)$ are loaded to the registers. Once k is available inthe τNAF representation, at the input of control unit, the FAU starts the computations based on the FSM stored in the control unit. The final and intermediate results are stored in the registers. The data bus width is set to 163 bits.

Table 6.2: The implementation results of the point multiplication on Koblitz curves on Altera® Stratix™ II EP2S180F1020C3 FPGA device.

d	$M+1$	**Latency** (L_{Total})	$f_{\max}$ (MHz)	Area (ALMs)	**P.M. Time** [μs]
11	16	3791	198	7,978	19.15
21	9	2601	195	13,032	13.45
33	6	2091	192	20,386	10.89
41	5	1921	191	24,815	10.22
55	4	1751	165	32,856	10.62

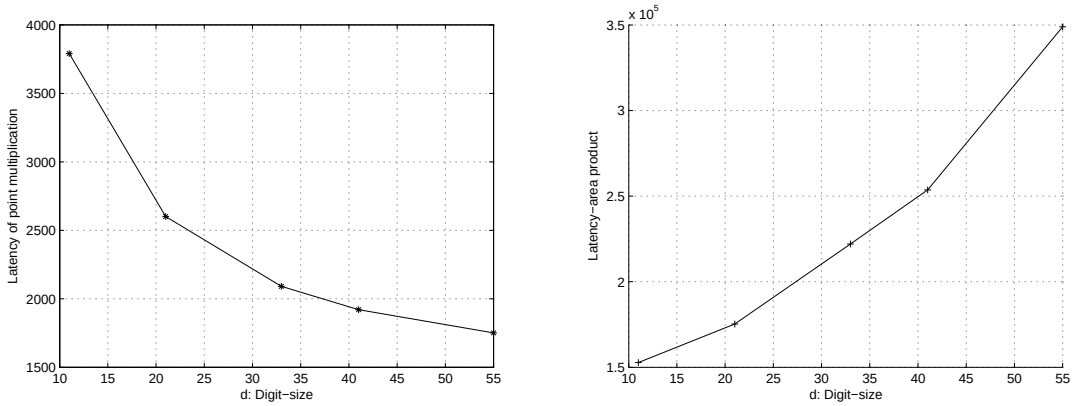
6.3.3 Coordinate Converter

The coordinate converter, gets the projective coordinates of $Q = kP$, i.e., (X, Y, Z), and provides affine coordinate of $Q = (x, y) = (X/Z, Y/Z^2)$ using an inversion based on the Itoh-Tsujii's scheme [38] and a field multiplication. As one can see in Fig. 6.2, it employs a multiplier and a squarer. Coordinate converter is implemented as a dedicated hardware and its latency and area is included in the implementation results presented in Table 6.2.

6.4 FPGA Implementations

FPGAs have advantages for prototyping and the proof of concepts. To have a fair comparison with previous works, we have selected Altera® Stratix™ II EP2S180F1020C3 device as the target FPGA for our implementations. In terms of available resources the target FPGA contains 71,760 ALMs (143,520 ALUTs and 143,520 registers) and 743 input/output (I/O) pins. Each ALMs contains two flip-flops (FFs) and two adaptive look-up tables (ALUTs). ALUTs are flexible and can be used to implement up to a 7-to-1-bit LUT. The presented architecture for point multiplication of the crypto-processor presented in Section 6.3 is coded in VHDL and synthesized for different digit sizes d, $d \in \{11, 21, 33, 41, 55\}$ for the Koblitz curve defined over $GF(2^{163})$.

We use Altera® Quartus® II version 11 design software for our implementations. The results of the area and maximum clock frequencies of the implementations after the place and route (provided by the fitter) are reported in Table 6.2. As one can see, increasing the digit-size results in the reduction of the latency of the point multiplication, i.e., L_{Total}, at the cost of increase in the area and decrease in the operating clock frequency. The point multiplication time is provided by diving the total number of clock cycles (L_{Total}) by the maximum operating clock frequency (f_{max}). To achieve



(a) (b)

Figure 6.3: (a): Latency of point computation on Koblitz curves over $GF(2^{163})$ for different digit sizes. (b): Latency-area product of the proposed architecture for point multiplication.

higher clock frequencies, we pipelined the digit-level GNB multiplier with only one level of pipelined registers. Therefore, we add one clock cycle to the latency of multiplier as seen in the second column of Table 6.2 (i.e., $M + 1$). The latency of loading the operands to the multipliers are counted in the total latency as shown in the data dependency graph illustrated in Fig. 6.1. Note that the fastest computation of point multiplication is obtained for $d = 41$ which is 10.22 μs employing 24,815 ALMs.

In Fig. 6.3a, the latency of point multiplication is plotted in terms of digit sizes. As one can see, as d increases the latency of point multiplication decreases and $d = 41$ is the largest digit-size than results in significant reductions in latency. To investigate the efficiency of the proposed architecture in term of time-area trade-offs, we plot the latency-area product in terms of different digit sizes in Fig. 6.3b. As one can see, the latency-area product always increases as digit-size increases but the increase is moderate when $d \leq 41$.

In what follows, we compare the implementation results to the counterparts especially the ones recently proposed in the literature.

6.4.1 Comparisons

High performance FPGA implementation of point multiplication on Koblitz curves have been considered in [90], [91], [23], [79], [26], and [10]. In Table 6.3, their best results in terms of time and area are summarized for point multiplications on Koblitz curve over $GF(2^{163})$, i.e., NIST K-163. As one can see, we implement our point

multiplication crypto-processor on the same FPGA device used by the counterparts. This makes our time and area comparisons to be fair and feasible.

As mentioned in Subsection 6.3.1, the finite field multiplier determines the area and time requirements of an ECC crypto-processor. We note that the finite field multiplier employed in this work, i.e., digit-level GNB multiplier with parallel-in and parallel-out, requires smaller area and operates in higher clock frequencies as compared to the ones used in [90], [91], [23], [79], [26], and [10].

The latency of the proposed architecture for point addition is less than the counterparts and is comparable with the one proposed in [26]. In [26] and [79], a new scheme known as interleaving is proposed to reduce the latency of point addition on Koblitz curves. The interleaving idea is based on the fact that the point addition requires the result of the previous point addition. Thus, some parts of it (i.e., coordinates Z and X) can be processed with the data available before the previous operation (computing Y) is finished. This scheme reduces the latency of point addition about 50% of the one proposed in [10] employing four finite field multipliers. We note that in a reliable crypto-processor, a check for validating the resulting point not to be at infinity is required. Employing interleaving in [26] and [79] may result redundant computations in the case of the existence of a point at infinity. Therefore, our proposed scheme provides faster result in computing point multiplication after the one proposed in [26] which is slightly faster.

In [23], a method to reduce the number of point additions for computing point multiplication on Koblitz curves is proposed. Instead of representing k in τ-adic NAF, a two-dimensional Frobenius expansion (based on Kleinian integers) is introduced. This reduces the number of non-zero terms in k and consequently reduces the number of point additions. Also, instead of taking advantage of parallelism in lower level, multiple processors are used to compute the point multiplication and the best results (in terms of time-area trade-off) have been reported with choosing the number of processors to be four. A digit-level version of Massey-Omura multiplier with the digit size $d = 25$ over $GF(2^{163})$ is employed in each processor to perform finite field multiplications. With efficient choosing of the parameters for two-dimensional Frobenius expansion of k, the smallest latency and time to compute a point multiplication are obtained as 2033 clock cycles and 17.15 μs (13.38 μs without conversion), respectively. It is worth mentioning that parallelization in arithmetic level is more beneficial than parallelization at higher levels, i.e., point multiplication as employed in [23]. Furthermore, one can achieve higher speeds employing two-dimensional Frobenius expansion and our parallelization scheme.

Table 6.3: Comparison of related works for FPGA implementations of point multiplication on Koblitz curves using digit-level finite field multipliers.

Work	τ-adic Conv.	FPGA device	Basis	Curve	# Multipliers[1]	Area	Time [μs]
[10]	Yes	Stratix II	NB	K-163	3	23,346 ALMs	34.57
[23]	Yes	Stratix II	NB	K-163	4	28,328 ALMs	17.15
[10]	NO	Stratix II	NB	K-163	3	22,416 ALMs	28.95
[26]	NO	Stratix II	NB	K-163	4	23,580 ALMs	9.48
This work	NO	Stratix II	GNB	K-163	4	24,815 ALMs	10.22

1. The number of multipliers in the main loop of point multiplication.

In [90], a double point multiplication algorithm proposed which employs a digit-level Massey-Omura multiplier with the digit size $d = 4$. It only employs one multiplier to perform point addition on Koblitz curves. Since double point multiplication is required in digital signature algorithm and its fast computation is important, our highly parallel scheme can improve its timing results.

The proposed scheme to employ four parallel multipliers can also be applied for the schemes based on polynomial basis and hence similar improvement can be achieved. Note that in this chapter we did not consider resistivity against side channel attacks as the main focus of this chapter is on highly parallel implementation of point multiplication. The reader is referred to [89] for detail information about countermeasures against side channel attacks.

6.5 Conclusion

We have proposed a new fast data flow graph for the point addition formulation using lopez-Dahab mixed coordinates employing four parallel multipliers on Koblitz curves. It is shown that the data flow graph has three multipliers in its critical path as compared to four multipliers for the best scheme available in the literature. We have used a low-complexity digit-level GNB multiplier to perform finite field multiplications. The analyzes results show that our method results in smaller latencies in computing point addition. Moreover, the implementations results on Altera® Stratix™ II indicates that our parallel multipliers operates at higher clock frequencies and the point multiplication results are faster than the ones previous ones available in the literature and favorably comparable in terms of area with the one proposed in [26]. Our proposed architecture performs a point multiplication on NIST K-163 in 10.22 μs employing 24,815 ALMs.

Chapter 7

Summary and Future Work

7.1 Thesis Contributions

IN this thesis, we have investigated finite field multipliers using Gaussian normal basis and proposed different architectures. This includes novel high speed digit-level multiplier architecture for ECC to make it fast. We have also considered the design, implementation, and evaluation of different elliptic curve crypto-processors for binary elliptic curves. The following summarizes the contributions of this work.

- In Chapter 3, which has been published in [9] and [61], we have presented a low complexity architecture for digit-level parallel in parallel out (DL-PIPO) GNB multiplier and proposed a common subexpression elimination algorithm to reduce its area complexity. We have also reduced the complexity of digit-level parallel in serial out (DL-PISO) GNB multiplier architecture in this chapter. Moreover, an improved architecture for digit-level serial in parallel out (DL-SIPO) GNB multiplier architecture is proposed and its time and area complexities are derived. It is noted that the proposed architecture outperforms the leading ones in the literature in terms of time and area. Further, we have extended the digit-level architectures to a low-complexity bit-parallel architecture and compared it with the counterparts. To evaluate the performance of the proposed multiplier architectures, we have implemented them on FPGA and ASIC and their area and timing results are reported which appear as the best results in comparison to the counterparts in the literature.

- In Chapter 4, which recently has been appeared in [65], for the first time, we have proposed an efficient hardware architecture for point multiplication on binary Edwards and generalized Hessian curves incorporating higher level par-

allelization and optimum lower level scheduling. We have proposed an efficient pipelining method for digit-level GNB multiplier architecture and employed it for the proposed ECC crypto-processor over $GF(2^m)$. Then, we have obtained the optimum digit sizes in terms of time-area trade-offs for the proposed crypto-processor. Further, we have performed efficient FPGA implementations of point multiplication on binary Edwards and generalized Hessian curves over $GF(2^{163})$ on a Xilinx® Virtex™-5 FPGA device and have investigated the LUT-based time-area efficiency for different digit sizes. The implementation results have been compared with the counterparts using binary generic curves.

- In Chapter 5, which has been outlined in [61], for the first time, we have proposed a new digit-level hybrid architecture which performs two multiplications together (double-multiplication) with the same number of clock cycles required as the one for one multiplication. The hybrid structure takes advantage of digit-level data interleaving and its structure is developed by combining the architecture of the proposed digit-level PISO GNB multiplier and a digit-level SIPO multiplier architecture. We have employed the proposed hybrid multiplier to reduce the latency of finite field double-exponentiation and point multiplication on binary elliptic curves. The analysis results indicated that the proposed architecture is suitable for the high speed applications whenever higher level of parallelization fails due to the data dependencies in computing point operations. Finally, we have implemented the hybrid architecture on a Xilinx® Virtex™-4 FPGA device and 65-nm ASIC and timing and area results have been reported.

- In Chapter 6, which has been presented in [92], we have proposed a highly parallel and fast crypto-processor for point multiplication on Koblitz curves. We have performed a latency analysis to determine where potential bottlenecks may occur and then find a balance between desired performance and the cost of implementing the design. In this effect, we have modified the point addition formulation to employ four parallel finite field multipliers and reduced the latency of point multiplication about 25% in comparison with the fastest one available in the literature. For investigating the practical performance of the proposed architecture, we have implemented the proposed ECC crypto-processor on an Altera® Stratix™ FPGA for different digit sizes over $GF(2^{163})$ targeting the applications where high speed is required and area usage should be considered as well. The implementation results have indicated that the proposed architecture outperforms the most recent ones available in the literature.

7.2 Future Work

As future works, for this thesis, the following can be pursued.

- Recently, a method to employ efficiently computable endomorphism to speed up point multiplication on ECC over quadratic extensions has been proposed. As a future work the idea can be extended to binary Edwards curves with some reasonable modifications which make it possible to use differential addition and efficient endomorphism to speed up point multiplication. This scheme is more efficient than many traditional doublings and the results from this will provide new set of standards for efficient implementations of ECC crypto-processor. These standards are applicable for a wide range of ECC applications.

- Pairing-based cryptography has a potential for solving many open problems in cryptography such as identity-based encryption and short signatures. The pairing computation is the most time-consuming operation in pairing-based schemes. The development of techniques and methods to optimize the pairing computation is of great importance and remains as a challenging effort for cryptosystems in commercial applications. There has been little research in the literature on implementation of pairing on binary elliptic curves. Therefore, as the lower level computations of pairing based cryptography relies on finite field arithmetic, the proposed low-complexity multiplier architectures in this thesis can be employed for efficient implementation of pairing as future works.

- Another future work for the proposed ECC crypto-processors that can be explored is the investigation against side channel attacks including simple power analysis attack and differential power analysis attack. Binary Edwards and generalized Hessian curves provide complete and unified addition formulation and they are very suitable for the applications where side channel attacks should be prevented. Therefore, fast computations of point multiplication on these curves should be considered for such applications.

- Finally, one can work on devising reliable architectures for the proposed ECC crypto-processors in this thesis against known faults and fault attacks in the literature. In this effect, a novel concurrent error detection scheme should be designed and tested for the point multiplication architectures presented in this thesis. For this purpose, parity based approaches can be utilized as they provide reasonable time/area overhead and efficient error detection capability.

Bibliography

[1] D. Bernstein, T. Lange, and R. Farashahi, "Binary Edwards Curves," in *Proceedings of Workshop on Cryptographic Hardware and Embedded Systems (CHES 2008)*, vol. 5154, 2008, pp. 244–265.

[2] R. Farashahi and M. Joye, "Efficient Arithmetic on Hessian Curves," in *Proceedings of The 13th International Conference on Practice and Theory of Public Key Cryptography (PKC 2010)*, 2010, pp. 243–260.

[3] J. López and R. Dahab, "Fast Multiplication on Elliptic Curves Over $GF(2^m)$ Without Precomputation," in *Proceedings of Workshop on Cryptographic Hardware and Embedded Systems (CHES 1999)*, 1999, pp. 316–327.

[4] T. Beth and D. Gollman, "Algorithm Engineering For Public Key Algorithms," *IEEE Journal on Selected Areas in Communications*, vol. 7, no. 4, pp. 458–466, 1989.

[5] A. Reyhani-Masoleh, "Efficient Algorithms and Architectures for Field Multiplication Using Gaussian Normal Bases," *IEEE Transactions on Computers*, vol. 55, no. 1, pp. 34–47, 2006.

[6] C. H. Kim, S. Kwon, and C. P. Hong, "FPGA Implementation of High Performance Elliptic Curve Cryptographic Processor over $GF(2^{163})$," *Journal of System Architcture*, vol. 54, no. 10, pp. 893–900, 2008.

[7] C.-Y. Lee, "Concurrent Error Detection Architectures for Gaussian Normal Basis Multiplication over $GF(2^m)$," *Integration, the VLSI Journal*, vol. 43, no. 1, pp. 113–123, 2010.

[8] F. Rodriguez-Henriquez, N. Saqib, and A. Díaz-Pérez, "A Fast Parallel Implementation of Elliptic Curve Point Multiplication over $GF(2^m)$," *Microprocessors and Microsystems*, vol. 28, no. 5-6, pp. 329–339, 2004.

[9] R. Azarderakhsh and A. Reyhani-Masoleh, "A Modified Low Complexity Digit-Level Gaussian Normal Basis Multiplier," in *Proceedings of Third International Workshop on Arithmetic of Finite Fields (WAIFI 2010)*, vol. 6087, 2010, pp. 25–40.

[10] K. Järvinen and J. Skyttä, "On Parallelization of High-Speed Processors for Elliptic Curve Cryptography," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 16, no. 9, pp. 1162–1175, 2008.

[11] D. Hankerson, S. Vanstone, and A. Menezes, *Guide to Elliptic Curve Cryptography*. Springer-Verlag New York Inc, 2004.

[12] J. Lopez and R. Dahab, "Fast Multiplication on Elliptic Curves over $GF(2^m)$ without Precomputation," *Cryptographic Hardware and Embedded Systems: First International Workshop, CHES'99, Worcester, MA, USA, August 1999: Proceedings*, 1999.

[13] P. Montgomery, "Speeding the Pollard and Elliptic Curve Methods of Factorization," *Mathematics of computation*, pp. 243–264, 1987.

[14] W. Diffie and M. Hellman, "New Directions in Cryptography," *IEEE Transactions on Information Theory*, vol. 22, no. 6, pp. 644–654, 1976.

[15] R. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, vol. 21, no. 2, pp. 120–126, 1978.

[16] N. Koblitz, "Elliptic Curve Cryptosystems," *Mathematics of Computation*, vol. 48, no. 177, pp. 203–209, 1987.

[17] V. S. Miller, "Use of Elliptic Curves in Cryptography," in *Proceedings of Advances in Cryptology-CRYPTO 85*, ser. Lecture Notes in Computer Science, Vol. 218, 1986, pp. 417–426.

[18] IEEE Std 1363-2000, "IEEE Standard Specifications for Public-Key Cryptography," Jan. 2000.

[19] U.S. Department of Commerce/NIST, "National Institute of Standards and Technology," *Digital Signature Standard, FIPS Publications 186-2*, January 2000.

[20] R. Cheung, N. Telle, W. Luk, and P. Cheung, "Customizable Elliptic Curve Cryptosystems," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 13, no. 9, pp. 1048–1059, 2005.

[21] B. Ansari and M. Hasan, "High-Performance Architecture of Elliptic Curve Scalar Multiplication," *IEEE Transactions on Computers*, vol. 57, no. 11, pp. 1443–1453, 2008.

[22] Y. K. Lee, K. Sakiyama, L. Batina, and I. Verbauwhede, "Elliptic-Curve-Based Security Processor for RFID," *IEEE Transactions on Computers*, vol. 57, no. 11, pp. 1514–1527, 2008.

[23] V. S. Dimitrov, K. U. Järvinen, M. J. J. Jr., W. F. Chan, and Z. Huang, "Provably Sublinear Point Multiplication on Koblitz Curves and its Hardware Implementation," *IEEE Transactions on Computers*, vol. 57, no. 11, pp. 1469–1481, 2008.

[24] W. Chelton and M. Benaissa, "Fast Elliptic Curve Cryptography on FPGA," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 16, no. 2, pp. 198–205, 2008.

[25] M. Keller, A. Byrne, and W. P. Marnane, "Elliptic Curve Cryptography on FPGA for Low-Power Applications," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 2, no. 1, pp. 1–20, 2009.

[26] K. Järvinen and J. Skyttä, "Fast Point Multiplication on Koblitz Curves: Parallelization Method and Implementations," *Microprocessors and Microsystems*, vol. 33, no. 2, pp. 106–116, 2009.

[27] Y. Zhang, D. Chen, Y. Choi, L. Chen, and S.-B. Ko, "A High Performance ECC Hardware Implementation with Instruction-level Parallelism over $GF(2^m)$," *Microprocessors and Microsystems - Embedded Hardware Design*, vol. 34, no. 6, pp. 228–236, 2010.

[28] H. Cohen, G. Frey, and R. Avanzi, *Handbook of Elliptic and Hyperelliptic Curve Cryptography*. CRC Press, 2006.

[29] A. Menezes, I. Blake, S. Gao, R. Mullin, S. Vanstone, and T. Yaghoobian, *Applications of Finite Fields*. Kluwer Academic Publisher, 1993.

[30] R. Lidl and H. Niederreiter, *Introduction to Finite Fields and Their Applications*, 2nd Edition, Cambridge University Press, 1997.

[31] T. Beth and D. Gollman, "Algorithm Engineering for Public Key Algorithms," *IEEE Journal on Selected Areas in Communications*, vol. 7, no. 4, pp. 458–466, 1989.

[32] J. Imana and J. Sanchez, "Bit-Parallel Finite Field Multipliers for Irreducible Trinomials," *IEEE Transactions on Computers*, vol. 55, no. 5, pp. 520–533, 2006.

[33] S. Kumar, T. Wollinger, and C. Paar, "Optimum Digit Serial $GF(2^m)$ Multipliers for Curve-Based Cryptography," *IEEE Transactions on Computers*, vol. 55, no. 10, pp. 1306–1311, 2006.

[34] A. Reyhani-Masoleh and M. Hasan, "Low Complexity Bit Parallel Architectures for Polynomial Basis Multiplication over $GF(2^m)$," *IEEE Transactions on Computers*, vol. 53, no. 8, pp. 945–959, 2004.

[35] J. Massey and J. Omura, "Computational Method and Apparatus for Finite Arithmetic," *US Patent*, no. 4587627, 1986.

[36] R. C. Mullin, I. M. Onyszchuk, S. A. Vanstone, and R. M. Wilson, "Optimal Normal Bases in $GF(p^n)$," *Discrete Appl. Math.*, vol. 22, no. 2, pp. 149–161, 1989.

[37] D. W. Ash, I. F. Blake, and S. A. Vanstone, "Low Complexity Normal Bases," *Discrete Applied Mathematics*, vol. 25, no. 3, pp. 191–210, 1989.

[38] T. Itoh and S. Tsujii, "A Fast Algorithm for Computing Multiplicative Inverses in $GF(2^m)$ Using Normal Bases," *Information and Computation*, vol. 78, no. 3, pp. 171–177, 1988.

[39] G. Feng, "A VLSI Architecture for Fast Inversion in $GF(2^m)$," *IEEE Transactions on Computers*, vol. 38, no. 10, pp. 1383–1386, 1989.

[40] C. Lee, P. Meher, and J. Patra, "Concurrent Error Detection in Bit-Serial Normal Basis Multiplication Over $GF(2^m)$ Using Multiple Parity Prediction Schemes," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 18, no. 8, pp. 1234–1238, 2010.

[41] W. Geiselmann and D. Gollmann, "Symmetry and Duality in Normal Nasis Multiplication," in *Proceedings of Sixth Symposium Applied Algebra, Algebraic Algorithms and Error-Correcting Codes (AAECC 1989)*, July 1989, pp. 230–238.

[42] G. B. Agnew, R. C. Mullin, I. M. Onyszchuk, and S. A. Vanstone, "An Implementation for a Fast Public-Key Cryptosystem," *Journal of Cryptology*, vol. 3, no. 2, pp. 63–79, 1991.

[43] A. Reyhani-Masoleh and M. A. Hasan, "Efficient Digit-serial Normal Basis Multipliers over Binary Extension Fields," *ACM Transactions Embedded Computing Systems (TECS).*, vol. 3, no. 3, pp. 575–592, Aug 2004.

[44] S. Kwon, K. Gaj, C. H. Kim, and C. P. Hong, "Efficient Linear Array for Multiplication in $GF(2^m)$ using a Normal Basis for Elliptic Curve Cryptography," in *Proceedings of Workshop on Cryptographic Hardware and Embedded Systems (CHES 2004)*, 2004, pp. 76–91.

[45] A. H. Namin, H. Wu, and M. Ahmadi, "A Word-Level Finite Field Multiplier Using Normal Basis," *IEEE Transactions on Computers*, vol. 99, no. Preprints, 2010.

[46] C. Lee and P. Chang, "Digit-Serial Gaussian Normal Basis Multiplier over $GF(2^m)$ Using Toeplitz Matrix-Approach," in *Proceedings of International Conference on Computational Intelligence and Software Engineering (CiSE 2009)*, 2009, pp. 1–4.

[47] C. C. Wang, T. K. Truong, H. M. Shao, L. J. Deutsch, J. K. Omura, and I. S. Reed, "VLSI Architectures for Computing Multiplications and Inverses in $GF(2^m)$," *IEEE Transactions on Computers*, vol. 34, no. 8, pp. 709–717, 1985.

[48] Ç. K. Koç and B. Sunar, "An Efficient Optimal Normal Basis Type II Multiplier over $GF(2^m)$," *IEEE Transaction on Computers*, vol. 50, no. 1, pp. 83–87, 2001.

[49] M. Hasan, M. Wang, and V. Bhargava, "A modified Massey-Omura Parallel Multiplier For a Class of Finite Fields," *IEEE Transactions on Computers*, vol. 42, no. 10, pp. 1278–1280, 2002.

[50] A. Reyhani-Masoleh and M. A. Hasan, "A New Construction of Massey-Omura Parallel Multiplier over $GF(2^m)$," *IEEE Transactions on Computers*, vol. 51, no. 5, pp. 511–520, 2002.

[51] L. Gao and G. E. Sobelman, ""Improved VLSI Designs for Multiplication and Inversion in $GF(2^M)$ over normal bases"," in *Proceedings of 13th Annual IEEE International ASIC/SOC Conference*, 2000, pp. 97–101.

[52] U. Kocabas, J. Fan, and I. Verbauwhede, "Implementation of Binary Edwards Curves for Very-Constrained Devices," in *Proceedings of 21st International Conference on Application-specific Systems Architectures and Processors (ASAP 2010)*, 2010, pp. 185–191.

[53] L. Batina, J. Hogenboom, N. Mentens, J. Moelans, and J. Vliegen, "Side-channel Evaluation of FPGA Implementations of Binary Edwards Curves," in *Proceedings of 17th IEEE International Conference on Electronics, Circuits, and Systems (ICECS 2010)*, 2010, pp. 1255–1258.

[54] R. Moloney, A. O'Mahony, and P. Laurent, "Efficient Implementation of Elliptic Curve Point Operations Using Binary Edwards Curves," Cryptology ePrint Archive, Report 2010/208, 2010, http://eprint.iacr.org/.

[55] E. Al-Daoud, R. Mahmod, M. Rushdan, and A. Kilicman, "A New Addition Formula for Elliptic Curves Over $GF(2^m)$," *IEEE Transactions on Computers*, vol. 51, no. 8, pp. 972–975, 2002.

[56] B. Sunar and Ç. K. Koç, "An Efficient Optimal Normal Basis Type II Multiplier over $GF(2^m)$," *IEEE Transaction on Computers*, vol. 50, no. 1, pp. 83–87, 2001.

[57] S. Kwon, "A Low Complexity and a Low Latency Bit Parallel Systolic Multiplier over $GF(2^m)$ Using an Optimal Normal Basis of Type II," in *Proceedings of 16th IEEE Symposium on Computer Arithmetic (Arith-16 2003)*, 2003, pp. 196–202.

[58] J. Gathen, A. Shokrollahi, and J. Shokrollahi, "Efficient Multiplication Using Type 2 Optimal Normal Bases," in *Proceedings of First International Workshop on Arithmetic of Finite Fields, (WAIFI 2007)*, vol. 4547, 2007, pp. 55–68.

[59] H. Fan and M. Hasan, "Subquadratic Computational Complexity Schemes for Extended Binary Field Multiplication Using Optimal Normal Bases," *IEEE Transactions on Computers*, vol. 56, no. 10, p. 1435, 2007.

[60] D. Bernstein and T. Lange, "Type-II Optimal Polynomial Bases," in *Proceedings of Third International Workshop on Arithmetic of Finite Fields (WAIFI 2010)*, vol. 6078, 2010, pp. 41–61.

[61] R. Azarderakhsh and A. Reyhani-Masoleh, "A Low Complexity Hybrid Architecture for Double-MUltiplication Using Gaussian Normal Basis," *IEEE Transactions on Computers,*, 2011.

[62] O. Gustafsson and M. Olofsson, "Complexity reduction of constant matrix computations over the binary field," in *WAIFI*, ser. Lecture Notes in Computer Science, vol. 4547. Springer, 2007, pp. 103–115.

[63] J. Gathen, A. Shokrollahi, and J. Shokrollahi, "Efficient multiplication using type 2 optimal normal bases," in *WAIFI*, ser. Lecture Notes in Computer Science, C. Carlet and B. Sunar, Eds., vol. 4547. Springer, 2007, pp. 55–68.

[64] Xilinx, "Xilinx Virtex-5 device data sheet," *www.xilinx.com/support/documentation/virtex-5.htm*, vol. ver5.0, Febraury 2009.

[65] R. Azarderakhsh and A. Reyhani-Masoleh, "Efficient FPGA Implementation of Point Multiplication on Binary Edwards and Generalized Hessian Curves Using Gaussian Normal Basis," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, no. 99, 2011.

[66] N. Koblitz, "Elliptic Curve Cryptosystems," *Mathematics of Computation*, vol. 48, pp. 203–209, 1987.

[67] D. J. Bernstein, "Batch Binary Edwards," in *Proceedings of the 29th Annual International Cryptology Conference on Advances in Cryptology (CRYPTO 2009)*, 2009, pp. 317–336.

[68] E. Brier and M. Joye, "Weierstraß Elliptic Curves and Side-channel Attacks," in *Proceedings of International Conference on Practice and Theory of Public Key Cryptography (PKC 2002)*, 2002, pp. 183–194.

[69] B. Baldwin, R. Moloney, A. Byrne, G. McGuire, and W. P. Marnane, "A Hardware Analysis of Twisted Edwards Curves for an Elliptic Curve Cryptosystem," in *Proceedings of 5th International Workshop on Reconfigurable Computing: Architectures, Tools and Applications (ARC 2009)*, vol. 5453, 2009, pp. 355–361.

[70] C.-P. Schnorr, "Efficient Signature Generation by Smart Cards," *Journal of Cryptology*, vol. 4, no. 3, pp. 161–174, 1991.

[71] T. E. Gamal, "A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms," *IEEE Transactions on Information Theory*, vol. 31, no. 4, pp. 469–472, 1985.

[72] C. Wang and D. Pei, "A VLSI design for computing exponentiations in $GF(2^m)$ and its application to generate pseudorandom number sequences," *IEEE Transactions on Computers,*, vol. 39, no. 2, pp. 258–262, feb 1990.

[73] C. Lee, J. Lin, and C. Chiou, "Scalable and Systolic Architecture for Computing Double Exponentiation Over $GF(2^m)$," *Acta Applicandae Mathematicae*, vol. 93, no. 1, pp. 161–178, 2006.

[74] J. H. Cheon, S. Jarecki, T. Kwon, and M.-K. Lee, "Fast Exponentiation Using Split Exponents," *IEEE Transactions on Information Theory*, vol. 57, no. 3, pp. 1816–1826, march 2011.

[75] J. Fan, D. Bailey, L. Batina, T. Guneysu, C. Paar, and I. Verbauwhede, "Breaking Elliptic Curves Cryptosystems using Reconfigurable Hardware," in *Proceedings of 20th International Conference on Field Programmable Logic and Applications (FPL 2010)*, 2010, pp. 133–138.

[76] Certicom, "Certicom ECC Chalenge," *www.certicom.com*, 1997.

[77] Standards for Efficient Cryptography Group, "SEC2: Recommended Elliptic Curve Domain Parameters," 2010, http://www.secg.org/download/aid-784/sec2-v2.pdf.

[78] R. Cheung, N. Telle, W. Luk, and P. Cheung, "Customizable Elliptic Curve Cryptosystems," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 13, no. 9, pp. 1048–1059, 2005.

[79] K. Järvinen, "Optimized FPGA-based elliptic curve cryptography processor for high-speed applications," *Integration, the VLSI Journal*, vol. 44, no. 4, pp. 270–279, 2011.

[80] W. N. Chelton and M. Benaissa, "Fast Elliptic Curve Cryptography on FPGA," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems.*, vol. 16, no. 2, pp. 198–205, 2008.

[81] O. Ahmadi, D. Hankerson, and F. Rodríguez-Henríquez, "Parallel Formulations of Scalar Multiplication on Koblitz Curves," *Journal of Univers. Computing Sci.*, vol. 14, no. 3, pp. 481–504, 2008.

[82] J.-Y. Lai and C.-T. Huang, "Elixir: High-Throughput Cost-Effective Dual-Field Processors and the Design Framework for Elliptic Curve Cryptography," *IEEE Transaction on VLSI Systems*, vol. 16, no. 11, pp. 1567–1580, 2008.

[83] B. Ansari and M. A. Hasan, "High-Performance Architecture of Elliptic Curve Scalar Multiplication," *IEEE Transactions on Computers*, vol. 57, no. 11, pp. 1443–1453, 2008.

[84] N. Koblitz, "CM-curves with Good Cryptographic Properties," in *Advances in Cryptology (CRYPTO 1991)*. Springer, 1992, pp. 279–287.

[85] J. A. Solinas, "Efficient Arithmetic on Koblitz Curves," *Des. Codes Cryptography*, vol. 19, pp. 195–249, March 2000.

[86] K. Järvinen, J. Forsten, and J. Skyttä, "Efficient Circuitry for Computing τ-adic Non-Adjacent Form," in *Proceedings of the 13th IEEE International Conference on Electronics, Circuits and Systems, (ICECS 2006)*. IEEE, 2006, pp. 232–235.

[87] B. B. Brumley and K. U. Järvinen, "Conversion Algorithms and Implementations for Koblitz Curve Cryptography," *IEEE Transactions on Computers*, vol. 59, no. 1, pp. 81–92, 2010.

[88] J. Adikari, V. Dimitrov, and K. Jarvinen, "A Fast Hardware Architecture for Integer to τ-NAF Conversion for Koblitz Curves," *IEEE Transactions on Computers*, vol. PP, no. 99, p. to appear, 2011.

[89] M. A. Hasan, "Power Analysis Attacks and Algorithmic Approaches to Their Countermeasures for Koblitz Curve Cryptosystems," *IEEE Transactions on Computers*, vol. 50, no. 10, pp. 1071–1083, 2001.

[90] J. Adikari, V. S. Dimitrov, and R. J. Cintra, "A New Algorithm for Double Scalar Multiplication Over Koblitz Curves," in *International Symposium on Circuits and Systems (ISCAS 2011),*. IEEE, 2011, pp. 709–712.

[91] C. Vuillaume, K. Okeya, and T. Takagi, "Short-Memory Scalar Multiplication for Koblitz Curves," *IEEE Trans. Computers*, vol. 57, no. 4, pp. 481–489, 2008.

[92] R. Azarderakhsh and A. Reyhani-Masoleh, "Highly Parallel and Fast Crypto-processor for Point Multiplication on Koblitz Curves," *IEEE Transactions on Computers, Special Issue on Computer Arithmetic*, vol. submitted, p. 9 pages, 2011.

Curriculum Vitae

Name:	Reza Azarderakhsh
Post-secondary Education and Degrees:	The University of Western Ontario Ph.D., London, Canada
	Sharif University of Technology M.Sc., Tehran, Iran
	Civil Aviation Technology College Tehran, Iran
Honors and Awards:	NSERC/IRDF Award (2012-2013) Ontario Graduate Scholarship (OGS) 2011-2012. Western Graduate Scholarship 2007-2011. PIMS and Western, Travel Grants 2008 and 2010. Polito TOPMED Scholarship 2006-2007. ITRC Master's Thesis Scholarship 2003-2005.
Related Work Experience:	Limited Duties Faculty Position (2011- present) The University of Western Ontario
	Graduate Teaching Assistant (2007-2011) The University of Western Ontario
	Graduate Research Assistant (2007-2011) The University of Western Ontario
	Visiting Instructor (2004-2007) Civil Aviation Technology College, Tehran, Iran
	Competent Electronic Design Engineer (2003-2007) Iranian Airport Holding Company, Tehran, Iran

PUBLICATIONS

Journal Papers:

1. R. Azarderakhsh and A. Reyhani-Masoleh, "Efficient FPGA Implementation of Point Multiplication on Binary Edwards and generalized Hessian Curves Using Gaussian Normal Basis", IEEE Transactions on VLSI Systems, accepted for publication, 2011, 14 pages.

2. R. Azarderakhsh, A. Reyhani-Masoleh, "Secure Clustering and Symmetric Key Establishments in Heterogeneous Wireless Sensor Networks", EURASIP Journal on Wireless Communication and Networking (JWCN), Special Issue on Security and Resiliency for Smart Devices and Applications, Article ID 893592, 12 pages, 2011, doi:10.1155/2011/893592.

Journal Papers (Under Revision):

1. R. Azarderakhsh and A. Reyhani-Masoleh, "A Low Complexity Hybrid Architecture for Double-Multiplication Using Gaussian Normal Basis", IEEE Transactions on Computers, Submitted, 2011, 14 pages.

2. R. Azarderakhsh and A. Reyhani-Masoleh, "Highly Parallel and Fast Cryptoprocessor for Point Multiplication on Koblitz Curves", IEEE Transactions on Computers, Special Issue on Computer Arithmetic, Submitted, 2011, 9 pages.

Conference Papers:

1. R. Azarderakhsh and A. Reyhani-Masoleh, "A Modified Low Complexity Digit-Level Gaussian Normal Basis Multiplier," a chapter in proceedings of 3rd International Workshop on the Arithmetic of Finite Fields (WAIFI 2010), LNCS No. 6087, Pages: 25-40, 27-30 Jun. 2010.

2. R. Azarderakhsh and A. Reyhani-Masoleh, and Z. Abid, "A Key Management Scheme for Cluster Based Wireless Sensor Networks," in proceedings of IEEE/IFIP International Conference on Embedded and Ubiquitous Computing (EUC 2008), Volume 2, Pages: 222–227, 17-20 Dec. 2008.

3. X. Yuan, H. Jürgensen, R. Azarderakhsh, and A. Reyhani-Masoleh, "Key Management for Wireless Sensor Networks Using Trusted Neighbors," in proceedings of IEEE/IFIP International Conference on Embedded and Ubiquitous Computing (EUC 2008), Volume 2, Pages: 228-233,17-20 Dec. 2008.

4. A. R. Masoum, A. H. Jahangir, Z. Taghikhaki, R. Azarderakhsh, "A New Multi Level Clustering Model to Increase Lifetime in Wireless Sensor Networks," in proceedings of the 2nd IEEE International Conference on Sensor Technologies and Applications, (SENSORCOMM 2008), Pages: 185-190, 25-31 Aug. 2008.

5. R. Azarderakhsh, A. H. Jahangir, and M. Keshtgary, "Network Survivability Performance Evaluation in Wireless Sensor Networks," in proceedings of the 11th International CSI Computer Conference (CSI 2006), Pages: 567-570, 24-26 Jan. 2006.

6. R. Azarderakhsh, A. H. Jahangir and M. Keshtgary, "A New Virtual Backbone for Wireless Ad Hoc Sensor Network with Connected Dominating Set," in proceedings of the 3rd IFIP Annual Conference on Wireless On demand Network Systems and Services (WONS 2006), Pages: 191-195, 18-20 Jan. 2006.

7. R. Azarderakhsh, A. H. Jahangir, "Optimized Routing Algorithms for Efficient Power Consumption in Wireless Sensor Networks" in proceedings of 13th International Electrical Engineering Conference (IEEC 2005), Pages 178-183, Apr. 2005.

8. R. Azarderakhsh, S.Gh. Miremadi, Gh. Moradi, "Flight Safety Management Systems," in proceedings of the 1st International Conference on Air Transport Industries Management (ICATIM 2005), Pages: 89-99, 19-20 Jan. 2005.