

**HIGH-PERFORMANCE HARDWARE IMPLEMENTATION
OF ELLIPTIC CURVE CRYPTOGRAPHY**

by

Md Selim Hossain



Dissertation submitted in fulfilment of the requirements

for the degree of

DOCTOR OF PHILOSOPHY

Department of Engineering
Faculty of Science and Engineering
Macquarie University
Sydney, Australia

June 2017

Copyright © 2017 Md Selim Hossain

All Rights Reserved

ABSTRACT

Elliptic curve cryptography (ECC), a public-key cryptography (PKC) encryption technique, has gained much interest among cryptography researchers because of its advantages over other commonly used PKC algorithms, such as the Rivest, Shamir and Adleman (RSA) cryptosystem. It offers equivalent security to RSA, but with significantly shorter key lengths. This attractive feature makes ECC very popular for resource-constrained applications such as smart cards, credit cards, pagers, personal digital assistants (PDAs), and cellular phones. ECC is considered to be more efficient in terms of speed, area, and power consumption.

This dissertation introduces several hardware implementations of an efficient ECC cryptosystem both on a field-programmable gate array (FPGA) and on an application-specific integrated circuit (ASIC) using VHDL. The first half of this dissertation discusses the high-performance hardware implementation of an ECC over the binary field \mathbb{F}_2^m and the second half describes an efficient implementation of an ECC over the prime field \mathbb{F}_p . These are implemented both in affine and Jacobian coordinates using the binary method (i.e. the double-and-add method) and the National Institute of Standards and Technology (NIST) recommended standard. The performance or efficiency of an ECC processor (ECP) is based on elliptic curve scalar (or point) multiplication (ECSM or ECPM) which is the most time and resource consuming operation in either a binary field or a prime field. The aim of this dissertation is to implement an efficient ECPM with a trade-off between speed, energy, and area complexities, required for modern security

applications. Various techniques are introduced to improve the performance of the ECPM, such as parallelisation, pre-computations, algorithm or architectural optimisation, and improved finite-field (or modular) arithmetic architectures. Although there is a substantial amount of work on separate point doubling (PD) and point addition (PA) implementations to compute elliptic curve group operations, essential for an ECPM, not much work is focused on a combined hardware architecture for group operations. In this research work, both modular arithmetic and group operations are optimised to improve the performance of ECPM. A combined PDPA architecture is designed which performs the PD and PA operations together in each iteration. Consequently, a uniform power consumption profile may be measured throughout the PDPA hardware, hence the point multiplication computation. Therefore, the proposed ECPM hardware implementation is secure against timing attacks and simple power analysis (SPA) attacks.

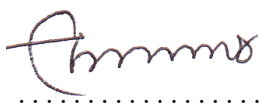
In this dissertation, the parallel ECPM architecture supports two Koblitz and random curves for the key sizes 163 and 233 bits and the ECP using a bit-serial multiplier supports all five NIST curves for the key sizes from 163 to 571 bits. The delay of a 233-bit parallel point multiplication is only $3.05\ \mu\text{s}$ in a Xilinx Virtex-7 FPGA and $0.81\ \mu\text{s}$ in an ASIC 65-nm technology. In addition, an energy-efficient ECP implementation over \mathbb{F}_2^m is achieved in which the $\text{Area} \times \text{Time} \times \text{Energy}$ value is lower in an ASIC platform than in all comparable work in the literature. On the other hand, an ASIC-based implementation of an ECP over prime fields supports three prime fields of the five NIST-recommended primes p , with sizes 192, 224, and 256 bits. The delay for ASIC-based ECP over \mathbb{F}_p is between 0.207 and 0.366 ms. To the best of the author's knowledge, these are the fastest hardware implementation result reported in the literature to date. Moreover, the energy dissipation is only about 0.3% of that of other similar designs.

STATEMENT OF CANDIDATE

I certify that the work in this thesis entitled “*High-Performance Hardware Implementation of Elliptic Curve Cryptography*” has not previously been submitted for a higher degree nor has it been submitted as part of the requirements for a degree to any other university or institution other than Macquarie University.

I also certify that the thesis is an original piece of research and it has been written by me. Any help and assistance that I have received in my research work and the preparation of the thesis itself have been appropriately acknowledged.

In addition, I certify that all information sources and literature used are indicated in the thesis.

A handwritten signature in dark ink, appearing to read 'Selim Hossain', written over a dotted line.

Md Selim Hossain

ACKNOWLEDGMENTS

I would like to thank my principal supervisor Dr Yinan Kong, director of the VLSI Research Group, Macquarie University, for providing me with the opportunity to pursue my PhD research at Macquarie University, Australia. He has provided me with many helpful suggestions and constant encouragement over the last three and a half years. Dr Kong has been very supportive and given me a proper direction for my research. I appreciate his contribution of invaluable time, creative ideas, and effective discussion. This thesis would not have been possible without his unconditional dedication, help and support, and guidance. I would also like to thank my associate supervisor A/Professor Christophe Doche for his initial direction, kind support, and inspiration.

I am grateful to Professor Joachim Rodrigues, Department of Electrical and Information Technology, Lund University, Sweden, for providing me with the opportunity to work under his guidance for the ASIC-based designs. I am immensely grateful to Dr Shahzad Asif, Dr Ehsan Saeedi, and Niras C. Vayalil for their continuous help and discussion on my research. I am thankful to the digital ASIC research team at Lund University for their valuable discussions on my work. I am also thankful to STMicroelectronics for use of their ASIC 65-nm CMOS technology library. I am grateful to Dr Oskar Andersson and my VLSI Group Members Dr Tariq M. Khan, Dr Azadeh Safari, Mr Abdullah-Al Nahid, and Mrs Naila Mukhtar for their help and support. My special thanks to Dr Debabrata Kumar Karmokar for his continuous help and advice during this research.

I am immensely grateful to Dr Keith Imrie, Honorary Associate, Department of Engineering, Macquarie University, for his continuous help in proofreading my research papers/thesis. I appreciate his corrections as these have helped considerably in the preparation of my papers/dissertation. His valuable advice and useful comments improved the quality of this dissertation. I am also grateful to Mr Phillip Thomas for proofreading one of my research papers. I am thankful to the admirable staff in the Department of Engineering, Macquarie University, to all my teachers, and all anonymous reviewers for their wonderful support.

I gratefully acknowledge Macquarie University for awarding me the international Macquarie University Research Excellence Scholarship (iMQRES) for providing financial support during three and a half years. I am thankful to Macquarie University for providing me with the Macquarie University Postgraduate Research Fund (MQ PGRF) for external collaboration and to attend an international conference and Faculty of Science HDR Research and Conference Funds to attend national conferences during this project. I am immensely grateful to Khulna University of Engineering & Technology (KUET), Bangladesh for allowing me to have study leave from my Assistant Professor position with the Department of Electrical & Electronic Engineering to pursue the PhD degree.

This thesis might not exist at all without the love and support of my family. Lastly but not least, thanks also goes to my brother-in-law Mr Hasanuzzaman, my father-in-law and mother-in-law, and my friends and colleagues Dr Shahidul Islam, Dr Raheel Hashmi, Mr Ariful Huq, Mr K. M. Morshed, Mr G. Nabi, Mr Kallol K. Karmakar, Mr Shahzamal, Mr Khizir, Mrs Sudipta, and Mr Affan for their endless help and support. Many more persons participated in various ways to ensure that my research succeeded than those and I am thankful to them all. Finally, to the Almighty Allah, for bestowing countless blessings on me.

Dedicated to

My loving wife, my cute daughter, my sisters, and my most amazing parents.

Contents

Table of Contents	xiii
List of Publications	xxiii
List of Contributors	xxvii
List of Figures	xxx
List of Tables	xxxvii
1 Introduction	1
1.1 Overview	1
1.2 Challenges and Objectives	3
1.3 Main Contributions	5
1.4 Dissertation Outline	8
2 Background	15
2.1 Overview of Cryptography	15
2.2 Private-key Cryptography	16
2.3 Public-key Cryptography	17
2.3.1 Discrete Logarithm Problem	18
2.3.2 Integer Factoring Problem	19

2.3.3	Elliptic Curve Discrete Logarithm Problem	19
2.4	Finite Field Arithmetic	20
2.4.1	Groups, Rings, and Fields	20
2.4.2	Finite Field	21
2.5	Binary Field \mathbb{F}_2^m Arithmetic Background	21
2.5.1	Normal Basis	22
2.5.2	Polynomial Basis	22
2.5.3	Addition in \mathbb{F}_2^m	23
2.5.4	Multiplication in \mathbb{F}_2^m	24
2.5.5	Squaring in \mathbb{F}_2^m	27
2.5.6	Inversion in \mathbb{F}_2^m	28
2.5.7	Complexity Analysis of Finite Field Arithmetic over \mathbb{F}_2^m	30
2.5.8	Reduction in \mathbb{F}_2^m	30
2.5.9	NIST Reduction Polynomials over \mathbb{F}_2^m	30
2.6	Prime Field \mathbb{F}_p Arithmetic Background	31
2.6.1	Modular Adder and Subtractor over \mathbb{F}_p	32
2.6.2	Modular Multiplier/Squarer over \mathbb{F}_p	33
2.6.3	Modular Inversion over \mathbb{F}_p	35
2.6.4	Complexity Analysis of Modular Arithmetic over \mathbb{F}_p	36
2.6.5	NIST Primes p over \mathbb{F}_p	37
2.7	Elliptic Curve Cryptography	38
2.7.1	Overview of Elliptic Curves	39
2.7.2	Elliptic Curve Point Arithmetic	41
2.8	Point Representation	43
2.8.1	Point doubling and point addition over \mathbb{F}_2^m in Jacobian Projective Coordinates	44

2.8.2	Point doubling and point addition over \mathbb{F}_p in Jacobian Projective Coordinates	45
2.9	Elliptic Curve Point Multiplication	47
2.9.1	Double-and-add point multiplication	48
2.9.2	Example of double-and-add point multiplication	49
2.10	Security Analysis	49
2.11	Platforms for Hardware Implementation	50
2.12	Elliptic Curve Digital Signature Algorithm	52
2.13	Summary of Proposed Designs	52
3	Efficient Hardware Implementation of Finite Field Arithmetic for Elliptic Curve Cryptography	55
3.1	Abstract	55
3.2	Introduction	56
3.2.1	Related Work	57
3.2.2	Our Contribution	59
3.2.3	Structure of the Paper	60
3.3	Preliminaries	60
3.4	Hardware for Finite Field Arithmetic	63
3.4.1	Finite Field Addition	63
3.4.2	Finite Field Multiplication	63
3.4.3	Finite Field Modular Reduction	72
3.4.4	Finite Field Inversion	73
3.5	Implementation Results and Comparison	75
3.5.1	FPGA Implementation Results	76
3.5.2	ASIC Implementation Results and Comparison	83
3.6	Conclusion	92

4 High-Performance FPGA Implementation of Elliptic Curve Cryptography Processor over Binary Field $\text{GF}(2^{163})$	93
4.1 Abstract	93
4.2 Introduction	94
4.3 Background	96
4.3.1 Groups and Fields	96
4.3.2 Elliptic Curve Cryptography (ECC)	97
4.4 Hardware Implementation for Finite Field	99
4.4.1 Polynomial Basis Representation	99
4.4.2 Addition in $\text{GF}(2^m)$	100
4.4.3 Multiplication in $\text{GF}(2^m)$	100
4.4.4 Squaring in $\text{GF}(2^m)$	102
4.4.5 Inversion in $\text{GF}(2^m)$	102
4.4.6 Proposed EC Group Operations	103
4.5 Proposed ECPM	105
4.6 FPGA Implementation Results and Performance Analysis	107
4.7 Conclusions	110
5 High-Speed, Area-Efficient, FPGA-Based Elliptic Curve Cryptographic Processor over NIST Binary Fields	111
5.1 Abstract	111
5.2 Introduction	112
5.3 Background	114
5.3.1 Finite Field	114
5.3.2 Coordinate Systems for Elliptic Curve Point Representation	115
5.3.3 Elliptic Curve Cryptography	116
5.4 Implementation of Finite-Field Arithmetic	118

5.4.1	Polynomial Basis Representation	118
5.4.2	Addition in $\text{GF}(2^m)$	120
5.4.3	Multiplication in $\text{GF}(2^m)$	120
5.4.4	Squaring in $\text{GF}(2^m)$	122
5.4.5	Inversion in $\text{GF}(2^m)$	123
5.4.6	Elliptic Curve Group Operations (ECPD and ECPA)	125
5.5	Elliptic Curve Scalar Multiplication (ECSM)	126
5.6	Results and Performance Analysis	128
5.7	Conclusion	131
6	Parallel Point-Multiplication Architecture using Combined Group Operations for High-Speed Cryptographic Applications	133
6.1	Abstract	133
6.2	Introduction	134
6.3	ECC Background	137
6.4	Proposed Point Multiplication in Projective Coordinates	140
6.4.1	Point Multiplication Algorithm	140
6.4.2	Architecture for ECPM	141
6.4.3	Security Analysis	143
6.5	Proposed Group Operations	143
6.6	Proposed Field Multiplication for \mathbb{F}_2^m	145
6.7	Comparisons and Performance Analysis	147
6.8	Conclusion	158
7	Efficient Hardware Implementation of Elliptic Curve Cryptography Processor Over NIST Binary Fields	159
7.1	Abstract	159

7.2	Introduction	160
7.2.1	Related Work	161
7.2.2	Our Contribution	161
7.2.3	Organisation of the Chapter	162
7.3	Mathematical Background of ECC	163
7.4	Hardware for Finite Field Arithmetic	164
7.4.1	Finite Field Modular Reduction	166
7.4.2	Finite Field Multiplication	166
7.4.3	Finite Field Inversion	170
7.5	Proposed Group Operations for ECP	172
7.6	Proposed ECC Processor	175
7.6.1	Proposed Point Multiplication Architecture	176
7.6.2	Jacobian-to-Affine Coordinate Conversion	178
7.6.3	Main Controller of Point Multiplication	179
7.6.4	Overall Architecture of ECC Processor	181
7.6.5	Security Analysis	182
7.7	Results and Performance Comparison	182
7.7.1	Hardware Implementation Results	183
7.7.2	Performance Comparison	194
7.8	Conclusion	196
8	FPGA-Based Efficient Modular Multiplication for Elliptic Curve Cryptography	199
8.1	Abstract	199
8.2	Introduction	200
8.3	Preliminaries	202
8.3.1	Elliptic-Curve Cryptography	202

8.3.2	Modular Multiplication over $\text{GF}(p)$	204
8.4	Hardware Architecture of Modular Multiplication over $\text{GF}(p)$ for ECC . . .	205
8.5	Hardware Implementation Results and Performance Analysis	208
8.6	Conclusion	211
9	High-Performance FPGA Implementation of Modular Inversion over \mathbb{F}_{256} for Elliptic Curve Cryptography	213
9.1	Abstract	213
9.2	Introduction	214
9.3	Mathematical Background	216
9.3.1	ECC	216
9.3.2	Coordinate Systems for EC Point Representation	218
9.3.3	Modular Inversion over $\text{GF}(p)$	218
9.4	Hardware Implementation of Modular Inverter over $\text{GF}(p)$	221
9.5	Results and Performance Analysis	225
9.6	Conclusion	228
10	High-Performance Elliptic Curve Cryptography Processor Over NIST Prime Fields	229
10.1	Abstract	229
10.2	Introduction	230
10.2.1	Related Work	231
10.2.2	Our Contribution	232
10.3	Mathematical Background	233
10.3.1	Elliptic Curve Cryptography	233
10.4	Hardware Architecture over $\text{GF}(p)$ for ECC	235
10.4.1	Modular Multiplier/Squarer over \mathbb{F}_p	235

10.4.2	Modular Inversion over \mathbb{F}_p	237
10.4.3	Proposed EC Group Operations	240
10.4.4	Proposed ECSM	242
10.5	Implementation Results and Performance Analysis	247
10.6	Conclusion	260
11	Energy-Efficient ASIC-Based Elliptic Curve Cryptography Processor	261
11.1	Abstract	261
11.2	Introduction	262
11.3	Preliminaries	265
11.3.1	ECC	265
11.3.2	Coordinate Systems for EC Point Representation	267
11.4	Hardware for FFMA over \mathbb{F}_p	268
11.4.1	Modular Adder and Subtractor	268
11.4.2	Modular Multiplier	269
11.4.3	Modular Inversion	274
11.5	Proposed PDPA	276
11.6	Proposed ECC Processor	278
11.6.1	Proposed Architecture for ECPM	278
11.6.2	Jacobian to Affine Coordinates Conversion	280
11.6.3	Control Unit of ECPM	281
11.6.4	Overall architecture	283
11.6.5	ECDSA	284
11.7	Implementation Results and Performance Comparison	286
11.8	Conclusion	293

12 Conclusions and Future Work	295
12.1 Conclusions	295
12.1.1 Conclusions for Binary Field Elliptic Curve Cryptography	296
12.1.2 Conclusions for Prime Field Elliptic Curve Cryptography	299
12.2 Future Work	301
A Simulation Waveforms and Results Sample	305
A.1 Simulation Results for Finite Field Arithmetic	305
A.1.1 Simulation waveform for Bit-Serial Multiplication	305
A.1.2 Simulation Results for Bit-Serial Multiplication	307
A.1.3 Simulation Waveform and Results for Traditional Digit-Serial Multiplication	309
A.1.4 Simulation Waveform and Results for Modified Digit-Serial Multiplication	310
A.1.5 Simulation Waveform and Results for Inversion	311
A.2 Simulation Waveform and Results for ECC Processor over NIST Binary Fields	312
A.3 Simulation Waveform and Results for ECC Processor over Prime Fields . .	313
B TCL Scripts Sample for Finite Field Arithmetic	315
B.1 Sample TCL scripts for bit-serial multiplier in Design Compiler	315
B.2 Sample TCL scripts for traditional digit-serial multiplier in Design Compiler	317
B.3 Sample TCL scripts for modified digit-serial multiplier in Design Compiler	318
B.4 Sample TCL scripts for inverter in Design Compiler	320
C Sample TCL Scripts for Elliptic Curve Cryptography Processor Over NIST Binary Fields	323

C.1	Sample TCL scripts for ECC Processor over \mathbb{F}_2^{571} using bit-serial multiplier in Design Compiler	323
C.2	Sample TCL scripts for ECC Processor over \mathbb{F}_2^{233} using digit-serial multiplier in Design Compiler	325
D	Sample TCL Scripts for ASIC-Based Elliptic Curve Cryptography Processor over Prime Fields	329
D.1	Sample TCL scripts for ECC Processor over \mathbb{F}_{256} in Design Compiler . . .	329
D.2	Sample TCL scripts for ECC Processor over \mathbb{F}_{256} in Prime Time	332
D.3	Sample TCL scripts to write a SDF file for ECC Processor over \mathbb{F}_{256}	332
D.4	Sample TCL scripts for ModelSim Simulation of ECC Processor over \mathbb{F}_{256} .	333
E	List of Acronyms	335
	Bibliography	339

List of Publications

This thesis is based on the following original publications, which are referred to in the text by Roman numbers. Original publications are reproduced with permission from their copyright holders.

Discussed in Thesis

- I M. S. Hossain, K. Ruangsantikorakul and Y. Kong, “Efficient Hardware Implementation of Finite Field Arithmetic for Elliptic Curve Cryptography”, *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, in review.
- II M. S. Hossain, E. Saeedi and Y. Kong, “High-Performance FPGA Implementation of Elliptic Curve Cryptography Processor over Binary Field $GF(2^{163})$ ”, *Proceedings of the 2nd International Conference on Information Systems Security and Privacy (ICISSP)*, Rome, Italy, pp. 415-422, 19-21 February, 2016, DOI:10.5220/0005741604150422.
- III M. S. Hossain, E. Saeedi and Y. Kong, “High-Speed, Area-Efficient, FPGA-Based Elliptic Curve Cryptographic Processor over NIST Binary Fields”, *2015 IEEE International Conference on Data Science and Data Intensive Systems (DSDIS)*, UTS, Sydney, Australia, pp. 175-181, 11-13 December, 2015, DOI: 10.1109/DSDIS.2015.44.

- IV** M. S. Hossain, E. Saeedi and Y. Kong, “Parallel Point-Multiplication Architecture using Combined Group Operations for High-Speed Cryptographic Applications”, *PLOS ONE*, vol. 12, pp. 1-18, May, 2017, DOI: 10.1371/journal.pone.0176214.
- V** M. S. Hossain, S. Asif and Y. Kong, “Efficient Hardware Implementation of Elliptic Curve Cryptography Processor Over NIST Binary Fields”, *IET Computers & Digital Techniques*, in review.
- VI** M. S. Hossain and Y. Kong, “FPGA-Based Efficient Modular Multiplication for Elliptic Curve Cryptography”, *International Telecommunication Networks and Applications Conference (ITNAC)*, UNSW, Sydney, Australia, pp. 191-195, 18-20 November, 2015, DOI: 10.1109/ATNAC.2015.7366811.
- VII** M. S. Hossain and Y. Kong, “High-Performance FPGA Implementation of Modular Inversion over F_{256} for Elliptic Curve Cryptography”, *2015 IEEE International Conference on Data Science and Data Intensive Systems (DSDIS)*, UTS, Sydney, Australia, pp. 169-174, 11-13 December, 2015, DOI: 10.1109/DSDIS.2015.47.
- VIII** M. S. Hossain, Y. Kong, E. Saeedi and N. C. Vayalil, “High-Performance Elliptic Curve Cryptography Processor over NIST Prime Field”, *IET Computers & Digital Techniques*, vol. 11, no. 1, pp. 33-42, 2016, DOI: 10.1049/iet-cdt.2016.0033.
- IX** M. S. Hossain, S. Asif, O. Andersson, J. N. Rodrigues and Y. Kong, “Energy-Efficient, High-Speed, ASIC-Based Elliptic Curve Cryptography Processor”, *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, major revision submitted.

Other Publications

These publication are related to this thesis but are not discussed in the thesis.

-
1. Y. Kong and M. S. Hossain, “An RNS-based FPGA Implementation of a Modular Multiplier for Public-Key Cryptography”, *to be submitted in a conference*.
 2. S. Asif, M. S. Hossain and Y. Kong, “A Fully RNS based ECC Processor”, *IEEE Transaction on Computers*, *in review*.
 3. E. Saeedi, Y. Kong and M. S. Hossain, “Feed-Forward Back-Propagation Neural Networks in Side-Channel Information Characterization”, *Journal of Systems Engineering and Electronics*, *in review*.
 4. S. Asif, M. S. Hossain and Y. Kong, “High-Throughput Multi-Key Elliptic Curve Cryptosystem Based on Residue Number System”, *IET Computers & Digital Techniques*, *Accepted*.
 5. Y. Kong and M. S. Hossain, “FPGA Implementation of Modular Multiplier in Residue Number System”, *IEEE International Conference on Big data Analysis (IEEE ICBDA 2017)*, *Accepted*.
 6. E. Saeedi, M. S. Hossain and Y. Kong, “Side Channel Information Characterization based on Cascade Feed-Forward Back-Propagation Neural Network”, *Journal of Electronic Testing*, Vol. 32, Issue 3, pp. 345-356, June, 2016, *doi:10.1007/s10836-016-5590-4*.
 7. E. Saeedi, Y. Kong and M. S. Hossain, “Side Channel Attacks and Learning Vector Quantization”, *Frontiers of Information Technology and Electronic Engineering* (Accepted in February 2016). [Online]. Available: <http://www.zju.edu.cn/jzus/ipa-article.php?doi=10.1631/FITEE.1500460>.
 8. E. Saeedi, Y. Kong and M. S. Hossain, “Multi-class SVMs Analysis of Side-Channel Information of Elliptic Curve Cryptosystem”, *2015 International Symposium on*

Performance Evaluation of Computer and Telecommunication Systems (SPECTS), Chicago, IL, USA, pp. 1-6, 26-29 July, 2015, DOI: 10.1109/SPECTS.2015.7285297.

9. E. Saeedi, Y. Kong and M. S. Hossain, “Side Channel Analysis of an Elliptic Curve Crypto-system Based on Multi-Class Classification”, *The Sixth International Conference on Computing, Communications and Networking Technologies (ICCCNT)*, Denton, Texas, USA, pp. 1-7, 13-15 July, 2015, DOI: 10.1109/ICCCNT.2015.7395195.

List of Contributors

Supervisor Dr Yinan Kong, Department of Engineering, Macquarie University, Sydney, Australia

VLSI research group members Dr Ehsan Saeedi, Dr Shahzad Asif, Niras Cheeckottu Vayalil, and Kuntapong Ruangsantikorakul, Department of Engineering, Macquarie University, Sydney, Australia

External Collaborator Dr Oskar Andersson, Department of Electrical and Information Technology (EIT), Lund University, Lund, Sweden

External Collaborator A/Professor Joachim Neves Rodrigues, Associate Professor, Department of Electrical and Information Technology (EIT), Lund University, Lund, Sweden

In all the publications listed in this thesis, I have conducted the major investigations, designs, measurements, data processing, and drafting. Dr Yinan Kong, who is my principal supervisor, provided helpful suggestions, important advice and invaluable guidance at every stage of this research. He also reviewed, proofread and corrected all the manuscripts. Dr Saeedi provided suggestions regarding planning and data analysis as well as reviewed and corrected manuscripts II, III, IV, and VIII. In addition, he performed a side-channel analysis of my proposed ECC processors. The main focus of his dissertation was side-channel attacks on elliptic curve cryptosystems based on machine learning techniques.

My internal collaborator Dr Shahzad Asif provided initial support with the Xilinx ISE simulation environment. He also helped me with the ASIC implementation platform and provided support to prepare a script file for an ASIC-based ECC processor (ECP). He implemented RNS-based elliptic curve point multiplication (ECPM) over the prime field \mathbb{F}_{256} . On the other hand, I have implemented binary-based ECC processors both in the binary field \mathbb{F}_2^m and the prime field \mathbb{F}_p . Niras C. Vayalil, VLSI group member, helped me in various ways to simulate my design of the high-performance ECP over NIST prime fields. K. Ruangsantikorakul, my internal collaborator and undergraduate student, provided support with the hardware implementation of a finite field multiplier in $\text{GF}(2^m)$ (or \mathbb{F}_2^m). My external collaborator, Dr Oskar Andersson from Lund University, Sweden, provided support for the ASIC 65-nm CMOS simulation and implementation environment. For an ASIC-Based Elliptic Curve Cryptography Processor over prime fields, my external collaborator A/Professor J. N. Rodrigues provided me invaluable guidance. He also reviewed, corrected, and proofread the last article in the list. A detailed list of contributors is depicted in Table 1.

LIST OF CONTRIBUTORS

DIVISION OF LABOUR IN CO-AUTHORED ARTICLES
MSH-Md Selim Hossain; YK-Yinan Kong; ES-Ehsan Saeedi; SA-Shahzad Asif; NCV-Niras Cheekottu Vayalil; KR-Kuntapong Ruangsantikorakul; OA-Oskar Andersson; JNR-Joachim Neves Rodrigues

	I	II	III	IV	V	VI	VII	VIII	IX
Conception & design	MSH	MSH, YK	MSH	MSH, YK	MSH	MSH, YK	MSH	MSH, YK	MSH
Planning & implementation	MSH, KR	MSH, ES	MSH	MSH	MSH, YK	MSH	MSH, YK	MSH, NCV	MSH, SA
Data collection	MSH	MSH	MSH, ES	MSH	MSH, SA	MSH	MSH	MSH	MSH, OA
Analysis & interpretation	MSH	MSH	MSH	MSH, ES	MSH, SA	MSH	MSH	MSH, ES	MSH, SA
Writing - original article	MSH, KR	MSH	MSH	MSH	MSH	MSH	MSH	MSH	MSH
Writing - review & editing	YK	ES, YK	ES, YK	ES, YK	YK	YK	YK	YK, ES	YK, JNR
Overall responsibility	YK	YK	YK	YK	YK	YK	YK	YK	YK, JNR

Table 1: Author's Contributions

List of Figures

1.1	Thesis outline	9
2.1	Private-key cryptography (or Symmetric cryptography) [28]	17
2.2	Public-key cryptography (PKC) (or Asymmetric cryptography) [28]	18
2.3	Elliptic curves over real number \mathbb{R}	40
2.4	Geometric point addition and point doubling on elliptic curve.	41
3.1	Implementation hierarchy of the ECC operations over $\text{GF}(2^m)$	62
3.2	Hardware Architecture of addition over $\text{GF}(2^m)$	64
3.3	Proposed bit-serial finite field multiplication architecture in $\text{GF}(2^m)$	66
3.4	Traditional digit-serial multiplication architecture over $\text{GF}(2^m)$: (a) original) [43, 46] and (b) proposed.	68
3.5	Proposed modified digit-serial field multiplication architecture in $\text{GF}(2^m)$	69
3.6	Hardware for modular reduction in $\text{GF}(2^m)$	72
3.7	Hardware Architecture for field inversion in \mathbb{F}_2^m	73
3.8	Performance analysis of traditional digit-serial multiplication with various digit sizes in Virtex-7 FPGA.	78
3.9	Performance analysis of traditional digit-serial multiplication with various digit sizes in Virtex-6 FPGA.	79

3.10	Performance analysis of modified digit-serial multiplication with various digit sizes in Virtex-7 FPGA.	81
3.11	Performance analysis of modified digit-serial multiplication with various digit sizes in Virtex-6 FPGA.	81
3.12	Area \times time \times energy (ATE) comparison of traditional digit-serial multiplication with digit sizes in \mathbb{F}_2^m	91
3.13	Area \times time \times energy (ATE) comparison of modified digit-serial multiplication with digit sizes in \mathbb{F}_2^m	91
4.1	Squaring a binary polynomial $U(x)$	102
4.2	Hardware architecture of the elliptic curve point doubling (ECPD) operation.	104
4.3	Hardware architecture of the elliptic curve point addition (ECPA) operation.	105
4.4	Hardware architecture of Elliptic Curve Point Multiplication (ECPM) processor.	106
5.1	Implementation hierarchy of the ECC operations over $\text{GF}(2^m)$	115
5.2	Squaring a binary polynomial $U(x)$	122
5.3	Proposed hardware architecture of the elliptic curve (a) point doubling (PD) and (b) point addition (PA).	126
5.4	Hardware architecture of elliptic curve scalar multiplier (ECSM).	127
6.1	Implementation hierarchy of the ECC operations over \mathbb{F}_{2^m}	139
6.2	Hardware architecture of proposed ECPM in Jacobian coordinates.	142
6.3	Proposed data-flow architecture for parallel computation of elliptic curve: (a) PD, (b) PA, and (c) PDPA.	144
6.4	Proposed parallel field multiplication architecture in $\text{GF}(2^m)$	146
6.5	Comparison ((a) AT and (b) performance) of point multiplication over $\text{GF}(2^{233})$	148

a	AT	148
b	Performance	148
6.6	AT comparison of point multiplication ([a], [b], [c], [d], [e], and [f] represent our work) over $\text{GF}(2^{163})$	154
6.7	Performance comparison of point multiplication ([a], [b], [c], [d], [e], and [f] represent our work) over $\text{GF}(2^{163})$	154
6.8	AT comparison of point multiplication ([g], [h], [i], and [j] represent our work) with references.	156
6.9	ATE comparison of point multiplication ([g], [h], [i], and [j] represent our work) with related designs.	157
7.1	Implementation hierarchy of the ECC operations over \mathbb{F}_{2^m}	165
7.2	Proposed bit-serial field multiplication architecture in $\text{GF}(2^m)$	167
7.3	Proposed digit-serial field multiplication architecture in $\text{GF}(2^m)$	170
7.4	Hardware Architecture for field inversion in \mathbb{F}_2^m	171
7.5	Proposed PDPA architecture for Koblitz Curves (a) K-163 and (b) K-233–K-571.	174
7.6	Proposed PDPA architecture (a) using bit-serial multiplication and (b) using digit-serial multiplication, for random or binary curves.	175
7.7	Detailed hardware architecture of proposed ECPM in Jacobian coordinates.	178
7.8	Proposed hardware architecture of Jacobian to affine conversion (a) using one inversion and (b) using two inversions.	179
7.9	Proposed main controller of ECPM in Jacobian coordinates.	180
7.10	Overall block diagram of ECP top module.	181
8.1	Implementation hierarchy of the ECC operations over $\text{GF}(p)$	204
8.2	A hardware architecture for a modular multiplier [80].	208

9.1	A hardware architecture for a modular inverter for finding (a) u_{new} , (b) v_{new} , (c) x_{new} , and (d) y_{new} [174].	224
10.1	Hardware architecture for a modular multiplier/squarer using Montgomery method [80,182]	236
10.2	Proposed hardware architecture for a modular multiplier/squarer.	238
10.3	Proposed hardware architecture for EC (a) PDBL, (b) PADD, and (c) PDPA.	241
10.4	Overall hardware architecture of proposed ECSM in affine coordinates for prime field.	244
10.5	Overall hardware architecture of proposed ECSM in Jacobian coordinates.	245
10.6	Proposed hardware architecture of (a) Jacobian to affine conversion and (b) top module of ECP.	246
10.7	Comparison of relative AT values between our ECC design ([b], [c], and [d]) and similar work.	259
11.1	Hardware architecture of modular (a) adder (b) subtractor, and (c) combined modular adder and subtractor.	269
11.2	Hardware architecture for the interleaved modular multiplier.	271
11.3	Proposed hardware architecture for (a) modular multiplier and (b) Radix-4 modular multiplier.	273
11.4	Proposed hardware architecture for PDPA.	277
11.5	Detailed hardware architecture of proposed ECPM in Jacobian coordinates.	280
11.6	Proposed hardware architecture of Jacobian to affine conversion (a) using one inversion and (b) using two inversions.	281
11.7	Proposed main controller of ECPM in Jacobian coordinates.	283
11.8	Overall block diagram of ECP top module.	284
A.1	Simulation waveform for bit-serial multiplier over \mathbb{F}_2^{163} , \mathbb{F}_2^{409} , and \mathbb{F}_2^{571}	306

A.2	Implementation results for bit-serial multiplier over \mathbb{F}_2^{163} in Virtex-7 FPGA.	308
A.3	Simulation waveform and implementation results in Virtex-7 FPGA for traditional digit-serial multiplier over \mathbb{F}_2^{233} with digit size of 4 bits.	309
A.4	Simulation waveform and implementation results in Virtex-7 FPGA for modified digit-serial multiplier over \mathbb{F}_2^{283} with digit size of 32 bits.	310
A.5	Simulation waveform and implementation results in Virtex-7 FPGA for finite field inversion over \mathbb{F}_2^{571}	311
A.6	Simulation waveform and implementation results in Virtex-7 FPGA for point multiplication over \mathbb{F}_2^{233} with digit size of 64 bits.	312
A.7	Simulation waveform and implementation results in Virtex-5 FPGA for point multiplication over \mathbb{F}_{256} in Jacobian coordinates.	313

List of Tables

1	Author's Contributions	xxix
2.1	NIST recommended irreducible polynomials over $GF(2^m)$ [22]	31
2.2	Complexity of $GF(p)$ FFMA operations in terms of clock cycles	37
2.3	NIST recommended primes over \mathbb{F}_p [22, 54]	38
2.4	Prime p for Koblitz curves over \mathbb{F}_p [54]	38
3.1	Elliptic curve point doubling (PD) and point addition (PA) in affine coordinates over $GF(2^m)$	61
3.2	Elliptic curve point doubling (PD) and point addition (PA) in Jacobian projective coordinates over $GF(2^m)$	61
3.3	Registers used for traditional digit-serial multiplication in $GF(2^m)$	69
3.4	The scheduling of V inputs for each processing unit in $GF(2^{233})$ when $d = 8$	71
3.5	The number of processing units (PUs) and clock cycles for each digit size implemented in $GF(2^{163})$, $GF(2^{233})$, $GF(2^{283})$, $GF(2^{407})$, and $GF(2^{571})$	71
3.6	Complexity of $GF(2^m)$ finite field arithmetic operations in terms of latency	75
3.7	Performance and comparison of FPGA-based implementation results of finite field multiplication using bit-serial approach over $GF(2^m)$	76
3.8	Performance and comparison of FPGA-based implementation results of digit-serial finite field multiplication over $GF(2^m)$	79

3.9	FPGA-based implementation results of modified digit-serial finite field multiplication over $GF(2^m)$	80
3.10	Performance and comparison of FPGA-based implementation results of finite field inversion over $GF(2^m)$	83
3.11	Performance and comparison of ASIC-based synthesis results of finite field arithmetic over \mathbb{F}_2^m	85
3.12	Performance and comparison of ASIC-based digit-serial finite field multiplication over \mathbb{F}_2^m	86
3.13	Performance analysis of ASIC-based modified digit-serial finite field multiplication over \mathbb{F}_2^m	89
4.1	comparison of Key length for equivalent security of Symmetric-key and public-key Cryptography [2]	98
4.2	NIST-recommended elliptic curves over \mathbb{F}_2^{163}	100
4.3	Synthesis Results of the finite-field arithmetic for $GF(2^{163})$ in Kintex-7 . .	107
4.4	Elliptic curve Group Operation Results for $GF(2^m)$ in Kintex-7.	107
4.5	Synthesis results for elliptic curve point multiplication (ECPM) over $\mathbb{F}_{2^{163}}$. .	108
4.6	Comparison between our ECC design and related work over $GF(2^{163})$. . .	109
5.1	comparison of Key length for equivalent security of Symmetric-key and public-key Cryptography [2, 33]	117
5.2	NIST-recommended elliptic curves over \mathbb{F}_2^{233} and \mathbb{F}_2^{283} [22]	119
5.3	Example of Computing Multiplication in $GF(2^4)$ Based on Algorithm 5.1 $(f(x) = x^4 + x + 1 = 10011, U(x) = x^3 + x^2 + x + 1 = 1111, V(x) = x^3 + x^2 + x = 1110)$	121
5.4	Example of Computing Inversion in $GF(2^4)$ Based on Algorithm 5.2 $(f(x) = x^4 + x + 1 = 10011, U(x) = x^3 + x^2 + x + 1 = 1111)$	124

5.5	Synthesis Results of the finite-field arithmetic for $\text{GF}(2^m)$ in Kintex-7 . . .	128
5.6	Elliptic Curve Group Operation Results for $\text{GF}(2^m)$ in Kintex-7	129
5.7	Elliptic Curve Scalar Multiplication Results for different GFs in Kintex-7 (XC7K325T-2FFG900)	129
5.8	Comparison between our ECC design and related work over $\text{GF}(2^m)$	130
6.1	Performance analysis of point multiplication on FPGA over $\text{GF}(2^{233})$. . .	149
6.2	Performance comparison of point multiplication on FPGA over $\text{GF}(2^{163})$.	150
6.3	Performance analysis of ASIC-based point multiplication over binary fields	155
7.1	Performance comparison of FPGA-based ECP using bit-serial field multi- plication over NIST Binary Fields $\text{GF}(2^m)$	184
7.2	Performance and comparison of FPGA-Based ECP using digit-serial mul- tiplication over NIST Binary Field $\text{GF}(2^{233})$	189
7.3	Performance analysis and comparison of ASIC-based ECC processor (ECP) over \mathbb{F}_2^m	191
8.1	NIST-recommended elliptic curves over \mathbb{F}_{256} [2, 22]	206
8.2	Performance of Modular Multiplication over \mathbb{F}_{256} on Virtex-7 FPGA. . . .	209
8.3	Device Utilization Summary (Estimated Values) for Modular Multiplica- tion over \mathbb{F}_{256}	209
8.4	Performance comparison of modular multiplication between our imple- mented design and other related work over \mathbb{F}_{256}	210
9.1	comparison of Key length for equivalent security of Symmetric-key and public-key Cryptography [2, 33]	217
9.2	NIST-recommended elliptic curves over \mathbb{F}_{256} [2, 22]	221

9.3	Device Utilization Summary (Estimated Values) for Modular Inversion over \mathbb{F}_{256} .	225
9.4	Performance Analysis of modular Inversion of our design and other related designs over \mathbb{F}_{256} .	226
10.1	Performance analysis of FFMA for ECP in affine coordinates over \mathbb{F}_p on FPGA.	249
10.2	FPGA Implementation of modular multiplication for ECP in Jacobian coordinates.	250
10.3	ASIC implementation of modular multiplication for ECP in Jacobian coordinates over \mathbb{F}_p .	252
10.4	EC Group Operation (PDBL and PADD) Results in affine coordinates for $\text{GF}(p)$ on Kintex-7	253
10.5	Comparison between our ECC design and similar work over $\text{GF}(p)$	254
11.1	ASIC implementation of modular multiplication over \mathbb{F}_p .	287
11.2	ASIC implementation of modular inversion over \mathbb{F}_p .	289
11.3	ASIC synthesis results for PDPA over \mathbb{F}_p	289
11.4	Performance comparison between our ECC design and similar ASIC-based designs over \mathbb{F}_p	290

Chapter 1

Introduction

1.1 Overview

With the increasing popularity of digital systems and the Internet the demand for secure transactions over the network has increased rapidly in recent times, because communications between two parties are generally conducted in an accessible environment, such as the Internet and wireless networks. Nowadays, applications of digital systems are appearing everywhere. For example, banking transactions or online shopping using credit card, safely signed important documents, protected confidential data (e.g. tax, medication, images, etc.), and social security numbers are transmitted over the network (e.g. Internet) during the transactions. Therefore, securing transactions over the network becomes a very demanding issue, because potential hackers/attackers could hack the system, alter transmitted data/information, eavesdrop communications, or attack the network with unauthorised devices. To mitigate these risks, strong and efficient cryptography is necessary to ensure authentication, authorisation, data confidentiality, and data integrity [1–4]. Private-key (or symmetric) cryptography and public-key (or asymmetric) cryptography (PKC) are the two main families of cryptography that can be used to protect and se-

cure data [2, 5, 6]. Although private-key cryptography is computationally less expensive than PKC, it has a key distribution problem because it requires pre-distribution of keys. On the other hand, PKC has no issue regarding key management as it allows flexible key management. However, PKC has a huge computational complexity to implement in hardware. With development of new technologies, an efficient hardware implementation of PKC should be designed that requires minimal hardware resources as well as less power consumption with a high throughput rate.

Several public-key cryptosystems have been introduced in the literature, such as Diffie-Hellman [7], Rivest-Shamir-Adleman (RSA) [8], ElGamal [9], and the elliptic curve cryptosystem which was independently proposed by Miller [10] and Koblitz [11] in the mid-80s. Within this, RSA and ECC are the most popular and powerful public-key cryptosystems for practical cryptographic applications. RSA's security is based on factoring of large integers, called the factoring problem [12]. On the other hand, ECC's security is based on the elliptic curve discrete logarithm problem (ECDLP) over the integers modulo a large prime, making it mathematically infeasible to determine 'key' for an intruder. Also, ECC offers the same level of security as RSA systems, but with significantly shorter key lengths and lower computational complexity than RSA [2, 13–17]. For example, a 283-bit ECC over a binary field or a 256-bit ECC over a prime field provides equivalent security to a 3072-bit RSA cryptosystem [2, 18, 19]. This feature creates the possibility to implement a much more efficient cryptographic hardware solution. Consequently, less memory, bandwidth, power and hardware resources are required to implement ECC. Nowadays, ECC is one of the most widely used and popular public-key cryptosystems for practical cryptographic applications. For example, ECC can be used for lightweight applications, such as smart cards [20]. Also, with the swift growth of Internet of things (IoT) applications, there is an inevitable demand for communication security, such as near-field communications (NFC), radio frequency identification (RFID), digital tags, and mobile

phones. To ensure ultimate security in communication networks, public-key cryptography (PKC), especially ECC, is mandatory. ECC needs much smaller key sizes for an equivalent security level, which means it can be developed with less area, power consumption, and higher computation speed. In addition, it is a very challenging job to implement high-performance cryptographic hardware for the IoT because, using limited resources, faster computation is required without degrading the security level. Therefore we can say that ECC requires low hardware resources to implement with high speed, which is suitable for the IoT. For this reason, this is an interesting topic for many researchers nowadays [21]. Different parameters can be selected for an elliptic curve cryptosystem to optimise the performance of hardware implementations. However, if the parameters (e.g. field bit length) are not selected carefully, they may lead to an insecure system. In this regard, the National Institute of Standards and Technology (NIST) recommends, in FIPS 186-2, ten finite fields, five binary fields $\text{GF}(2^m)$ and five prime fields $\text{GF}(p)$, for use in the elliptic curve digital signature algorithm (ECDSA) [22]. The ECC parameters have also been presented by IEEE [23] and ANSI [24]. Different bit lengths in each field can be used for different security levels. For each field a specific curve has been selected carefully for both cryptographic strength and efficient implementation. Implementing various bit-length elliptic curve cryptosystems in hardware makes ECC protocols more attractive.

1.2 Challenges and Objectives

PKC algorithms (e.g. ECC) have high computational complexity, which reduces their throughput rate and makes them difficult to implement. However, a high-throughput elliptic curve cryptosystem is mandatory for several practical applications like banking and email servers. Also, energy efficiency during computation has emerged as a major design parameter in diverse portable applications, such as smartphones, tablets, notebooks

(or laptops), and personal digital assistants (PDAs). Cryptosystems can be implemented with either a hardware or software approach. However, software implementations are usually very slow or much too power hungry to fulfil the requirements for real-time and embedded applications. In this dissertation, the hardware approach is considered because considerably faster implementations can be achieved in hardware than with software implementations. However, an efficient hardware implementation of an elliptic curve cryptosystem for modern practical applications is a very demanding job.

A cryptosystem performs a huge number of encryption and decryption processes which mostly rely on the efficiency of modular arithmetic, e.g. modular addition, subtraction, multiplication, squaring, and inversion. In the literature, a substantial amount of work is concentrated on algorithm optimisation of modular arithmetic [25]. This thesis also optimises and implements efficient modular arithmetic in either a binary field or a prime field, because a modular arithmetic unit is mandatory for elliptic curve group operations and elliptic curve point multiplication (ECPM), where ECPM is the core operation of an ECC processor (ECP). Note that point (or scalar) multiplication is the most time-consuming operation in elliptic curve based cryptosystems in either a binary field or a prime field. The existing literature is mostly focused on separate group operations to implement ECPM, but not much work is focused on combined group operations for optimisation. In addition, only a few hardware implementations focused on energy-efficient design, being mostly concentrated on reducing the computation time of ECPM. On the other hand, a high-speed, low-area, and low-power-consumption ECP is suitable for practical cryptographic applications. Hence, a trade-off between time, area, and power is required.

In this dissertation, the prime focus is to explore high-performance implementations of ECP in hardware and evaluate their energy efficiency and performance. For doing this, several hardware architectures of modular arithmetic, group operations and ECPM are

designed and optimised. The hardware architectures of the proposed ECPM for an ECP are implemented in both binary fields $GF(2^m)$ and prime fields $GF(p)$. High-performance (including high speed, low area, and low power consumption) hardware implementations of ECP, on both a field-programmable gate array (FPGA) and an application-specific integrated circuit (ASIC), are the focus of this thesis. An FPGA-based implementation of an ECC processor offers higher flexibility or reprogrammable hardware design than an application-specific integrated circuit (ASIC), and this means that the cryptographic algorithm can easily be updated. In addition, FPGA implementations are suitable for prototype design because they can drastically reduce the hardware development/test cost and time since they do not incur any fabrication cost. However, high-volume production of ASICs, after the first run, is much cheaper than the corresponding production based on FPGA devices. Besides, an ASIC-based hardware implementation is necessary for fast and low-power applications. Detailed performance analyses of the proposed ECPM architectures based on both FPGA and ASIC are discussed thoroughly in this thesis.

1.3 Main Contributions

In this dissertation, various novel ECPMs have been designed and implemented for an ECP to achieve high speed and low power consumption, which means low energy dissipation in hardware. The proposed designs are based on two finite (Galois) fields, a binary field $GF(2^m)$ (or \mathbb{F}_2^m) and a prime field $GF(p)$ (or \mathbb{F}_p). The optimisation aim is generally to reduce the latency of ECPM. For this, the hardware units are designed for finite field modular arithmetic: addition, subtraction, multiplication, squaring, and inversion, and elliptic curve group operations, either in a binary field or a prime field. These modular arithmetic units and group operations are then integrated to create an elliptic curve cryptographic processor capable of computing point multiplication on elliptic curves, i.e.

ECPM. The proposed high-performance implementations of ECP are synthesised in both FPGA and ASIC and results are discussed in detail. In this dissertation, the first half discusses binary-field ECP implementations and the second half discusses prime-field ECP implementations. Based on the implementation results, the performance of the proposed designs, either in FPGA or ASIC, is competitive (almost 50% better efficiency (or performance)) with existing hardware implementations. Moreover, the following are some of the key scientific contributions:

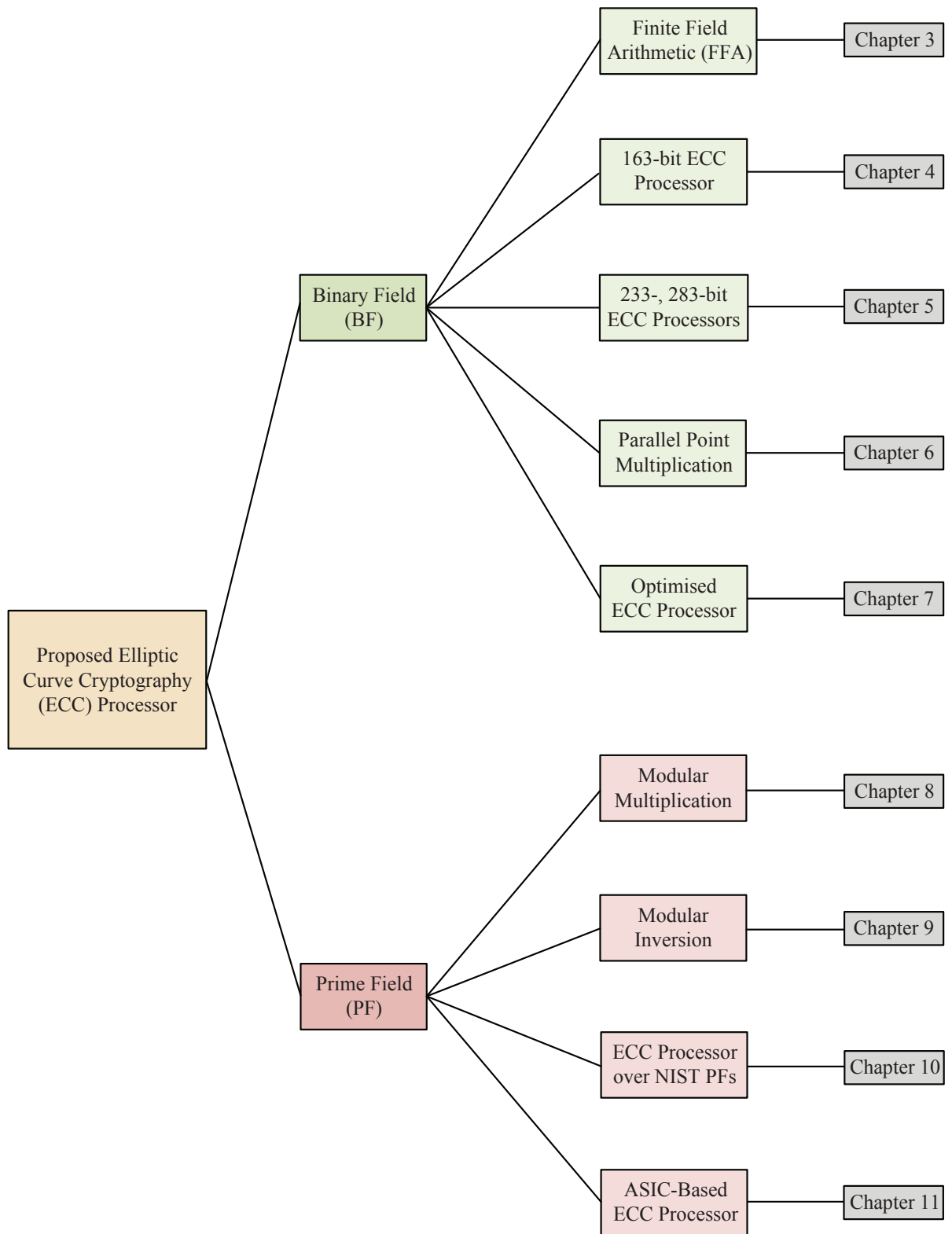
- To design high-performance point multiplication over the binary field, a finite field multiplier architecture is mandatory. This thesis proposes both bit-serial and bit-parallel finite field multiplier architectures using a polynomial basis.
- A digit-serial multiplier over $GF(2^m)$ is implemented using a traditional digit-serial approach, whose latency is $\lceil m/d \rceil$. This significantly reduces the overall computation time of ECPM in the binary field. The proposed digit-serial architecture supports all five NIST binary fields and provides low area \times time (AT) complexity.
- This thesis also proposes a modified digit-serial multiplier architecture over $GF(2^m)$ with a very low latency of $2 * \lceil \sqrt{m/d} \rceil$, which is better for high-throughput ECP over the binary field. The proposed design significantly improves the area \times time \times energy (ATE) performance using ASIC 65-nm technology.
- To implement a point multiplication in affine coordinates, a bit-serial inverter architecture over $GF(2^m)$ is proposed based on a modified Euclid's algorithm. The proposed architecture is computationally less expensive than state-of-the-art bit-serial or digit-serial inversion.
- To compute elliptic curve point operations, separate point doubling (PD) and point addition (PA) architectures over $GF(2^m)$ are designed. The PD and PA operations are implemented using a bit-serial FFA unit.

- 163-, 233-, and 283-bit ECC processors (ECPs) over $GF(2^m)$ are implemented on an FPGA using separate group operations and a bit-serial multiplier and inverter. The proposed ECPs are implemented in affine coordinates using NIST-recommended elliptic curves.
- A novel combined point doubling and point addition (PDPA) operation in Jacobian coordinates is developed for high-performance point multiplication. This combined method manages to efficiently speed up ECPM and improve the resource utilisation of the FPGA over the best existing FPGA hardware.
- For the practical realisation of a cryptosystem, an ECP in affine coordinates is necessary. For this, a conversion unit for Jacobian coordinates to affine coordinates is designed.
- A fast and novel parallel point multiplication architecture using combined PDPA hardware is proposed, and is faster than any hardware implementation in the literature.
- An energy-efficient ECP over NIST binary fields is proposed, and can be used for low-power devices. The proposed energy-efficient ECP using a bit-serial finite field multiplier supports all five Koblitz and random curves for the key sizes between 163 and 571 bits recommended by NIST. To the best of the author's knowledge, only a few hardware solutions for ECC over NIST binary fields can handle all five NIST curves.
- An efficient FPGA-based implementation of modular multiplication based on the Montgomery method and a modular inversion over the prime field $GF(p)$ are developed for an ECP. An ECP over $GF(p)$ is implemented in affine coordinates using the modular multiplier and inverter.

- A novel modular multiplier architecture is proposed based on an interleaved method. In addition, a fast and novel radix-4 modular multiplier architecture is proposed, which saves 50% on clock cycles.
- A separate (PD and PA) as well as a combined (PDPA) group operation unit are designed for prime fields $GF(p)$. The proposed PD and PA operations are implemented in affine coordinates. On the other hand, the proposed combined PDPA architecture is developed in Jacobian coordinates. As far as the author knows, no other point multiplication architecture is based on the combined PDPA.
- A high-performance point multiplication architecture for an ECP is proposed using a radix-4 modular multiplier and a combined PDPA hardware. This ASIC-based implementation is faster than other related work in the literature over prime fields $GF(p)$. It achieved an energy dissipation of only 0.3% that of other similar designs.

1.4 Dissertation Outline

This dissertation follows the non-traditional “Thesis-by-Publication” format which has been approved by the *Macquarie University Higher Degree Research Office (HDRO)*. It consists of a general introduction, background, and a list of the PhD candidate’s major scientific publications. The thesis materials are the original texts and graphics of the candidates, published or under review, that have been reformatted to improve readability. Fig. 1.1 summarises the dissertation outline. There are two main components of the proposed hardware implementation of elliptic curve cryptography (ECC). The first is the high-performance hardware implementation of ECC based on Galois finite field arithmetic $GF(2^m)$. The second is an ECC implementation based on modular arithmetic (e.g. modular addition, subtraction, multiplication, and inversion) in $GF(p)$. Finally, both FPGA and ASIC-based hardware implementations of ECP are achieved.

**Figure 1.1:** *Thesis outline*

Chapter 2 gives a brief overview of the theory and basics of ECC for both the binary field and the prime field, along with the elliptic curve group operations that have been used for this thesis. The use of coordinate systems to implement group operations (PD and PA) is also discussed, where group operations are mandatory to compute a point multiplication. The theory of finite field modular arithmetic and the mathematical operations are briefly discussed in this chapter. A comprehensive review of the literature available on high-performance ECC processors is included in this chapter.

In Chapter 3, the design and use of FFA for a high-performance ECP over the binary field $GF(2^m)$ is presented. It presents hardware implementations of finite field addition, multiplication, squaring, and inversion, which are needed to develop a binary-field ECP. Also, the overall performance of point multiplication mostly relies on the performance of FFA because a large number of FFA operations are required to develop an ECPM. There are four major implementations in this chapter: 1) a bit-serial multiplier, 2) a traditional digit-serial multiplier with various digit sizes, 3) a modified digit-serial multiplier with various digit sizes, and 4) a bit-serial inverter. Multiplication over $GF(2^m)$ is implemented in both bit-serial and digit-serial approaches. On the other hand, inversion over $GF(2^m)$ is implemented only with a bit-serial approach; implementation results show that it is computationally less expensive than the digit-serial approach. All four designs are synthesised in both FPGA and ASIC platforms, which support all five NIST binary fields between 163 and 571 bits, and their performance compared in terms of timing, area, and power consumption with related work in the literature. For example, the proposed inversion provides 2-3 times better performance in terms of area \times time (AT) than related work in the literature. The detailed implementation results are discussed in Chapter 3.

A high-performance hardware implementation of an ECC processor over $GF(2^{163})$ is presented in Chapter 4. It is implemented in a Kintex-7 FPGA with the Very High Speed Integrated Circuit (VHSIC) Hardware Description Language (VHDL). It is developed in

affine coordinates using a bit-serial multiplier and inverter and separate PD and PA operations. The design is area-efficient, needing only 2253 slices without any digital signal processing (DSP) blocks. In this design, the proposed ECPM provides nearly 50% better delay performance than related work in the literature.

Recall that the 163-bit ECC processor is not secure based on today's security-level requirements. For this reason, 233- and 283-bit FPGA-based ECC processors over $GF(2^m)$ are implemented in Chapter 5. This chapter thoroughly explains the hardware designs and implementations of finite field adder, multiplier, squarer, and inverter on a polynomial basis. In this chapter, elliptic curve point doubling (ECPD) and elliptic curve point addition (ECPA) are also implemented in affine coordinates, and then an ECC processor created capable of computing ECPM. The proposed design was simulated using both Xilinx ISim and ModelSim PE and all results are checked using Maple software. The implemented design in a Xilinx Kintex-7 FPGA is area-efficient, as it needs only 3016 and 4625 slices for the 233- and 283-bit ECP, respectively.

In Chapter 6, a novel parallel architecture for fast hardware implementation of point multiplication over the binary fields $GF(2^{163})$ and $GF(2^{233})$ is proposed. The ECC processor in affine coordinates presented in the previous two chapters takes many clock cycles, which means that it is not very efficient in terms of speed. Therefore, a highly parallel architecture for ECPM over the binary field $GF(2^m)$ is developed, which takes only m clock cycles. In addition, this chapter proposes a novel combined PDPA architecture in Jacobian coordinates to achieve high speed for ECPM. Moreover, a parallel architecture for field multiplication on a polynomial basis is proposed for the combined PDPA. The proposed ECPM supports two Koblitz and random curves for the key sizes 233 and 163 bits and is implemented on both FPGA and ASIC. Implementation results reveal that the proposed design is faster than any other related work, but takes more area. For this reason, the area \times time (AT) and area \times time \times energy (ATE) products of the proposed

point multiplication are calculated. Based on the implementation results, the proposed ECPM provides almost 50% better performance than recent implementations.

Chapter 7 reveals the optimisation techniques explored to improve the area, time, and energy efficiency of the overall ECP hardware architecture over NIST binary fields. These include the design and implementation of bit-serial and digit-serial multipliers, bit-serial inverter for conversion from Jacobian to affine coordinates, the optimisation of combined PDPA operations for both Koblitz and random elliptic curves, and the optimisation of point multiplication to get a high-performance ECP. The proposed ECPs over NIST binary fields $GF(2^m)$ support all five Koblitz and random curves for the key size from 163 to 571 bits. This chapter also presents the FPGA energy dissipation from a proper power analysis technique. For power analysis, an SAIF file is generated and shows a high confidence level and more than 95% switching activity profile, which is enough to get the accurate power consumption. Overall, nearly 50% of improvement is achieved over recent FPGA implementations. In addition, the ASIC-based energy dissipation is also computed for all designs from the power consumption and latency. Therefore, a high-speed, low-area and low-power ECP hardware is designed for modern high-security applications whose area \times time \times energy (ATE) value is lower in an ASIC platform than all comparable work in the literature. The proposed ASIC-based ECP takes 3 to 100 times and 2 to 55 times less ATE value than the most significant work in the literature.

Chapters 3-7 discuss hardware implementations over the binary field $GF(2^m)$. The second half (Chapters 8-11) of this dissertation presents hardware implementations over the prime field \mathbb{F}_p . In Chapter 8, a high-performance modular multiplier over \mathbb{F}_{256} is presented. This multiplier is based on the Montgomery multiplication method, and based on the multiplier the prime field ECP is developed. The required area and time in a Xilinx Virtex-7 FPGA are very low, namely 605 slices and 1.683 μs , respectively. The FPGA-based implementation results show that the proposed design is faster and more

area-efficient than recent hardware implementations.

In Chapter 9, an efficient modular inversion based on the efficient binary Inversion algorithm (which is based on the extended Euclidean algorithm (EEA)) is implemented for the prime field ECP. A modular inverter is mandatory for ECP in affine coordinates or conversion from projective to affine coordinates. The inverter architecture over \mathbb{F}_{256} is implemented in a Xilinx Virtex-7 FPGA and takes a small amount of resources, only 1480 slices. The computation time per inversion is only $2.329 \mu s$ at the maximum frequency of 146.39 MHz. From the performance analysis, it uses a better area and time than available hardware implementations in the literature.

Chapter 10 provides a description of an efficient hardware implementation of ECP over NIST prime fields $GF(p)$. Separate scalar (or point) multiplication architectures are presented for both affine and Jacobian coordinates, and their relative performance is compared. A scalar multiplication architecture in affine coordinates is proposed by designing separate point doubling (PDBL) and point addition (PADD) operations. The proposed ECP based on affine-coordinate scalar multiplication is implemented in Xilinx Kintex-7 and Virtex-5 FPGAs. On the other hand, the point multiplication architecture in Jacobian coordinates is developed by designing a novel combined PDPA operation and a proposed modular multiplier. The proposed ECP in Jacobian coordinates is implemented both in FPGA and ASIC. The trade-off between area and time is then discussed, where the most efficient design of elliptic curve scalar multiplication (ECSM) in both FPGA and ASIC for \mathbb{F}_{224} and \mathbb{F}_{256} is proposed. To the best of the author's knowledge, the ECP over \mathbb{F}_p proposed in this chapter provides around 20% better delay performance than all comparable work in the literature.

The number of clock cycles required for an ECP over the prime field proposed in the previous chapter is greater. Chapter 11 introduces a novel high-performance ASIC-based ECC processor over \mathbb{F}_p which saves almost 50% in clock cycles. This chapter discusses

the development of a novel radix-4 modular multiplier algorithm and a corresponding hardware architecture. The proposed radix-4 modular multiplier is then used in the implementation of the novel combined PDPA architecture in Jacobian coordinates. Finally, a novel point-multiplication architecture over the prime field for ECP is proposed. In this chapter, a control unit for point multiplication in Jacobian coordinates is explained clearly. A separate unit for Jacobian-to-affine coordinate conversion is developed because ECP in affine coordinates is required for the practical realisation of a cryptosystem. The proposed ECP is synthesised on the ASIC 65-nm CMOS STMicroelectronics standard cell library using Synopsys Design Compiler. The delay per point multiplication is between 0.21 and 0.37 ms for the key sizes from 192 to 256 bits recommended by NIST. To the best of the author's knowledge, these are the fastest hardware implementation results over the prime field reported in the literature to date. The ASIC-based implementation results for timing, area, and power consumption are discussed in detail to highlight the benefits of the proposed ECP architecture.

Finally, in Chapter 12 concluding remarks are drawn and a non-exhaustive list of future research directions is discussed.

Chapter 2

Background

This chapter provides background knowledge regarding cryptography, Galois (or finite) field operations, and elliptic curve cryptography (ECC). Also, this chapter describes the elliptic curve discrete logarithm problem (ECDLP) which assures the computational hardness or security of ECC. This chapter also introduces a combined group operations, point doubling and point addition (PDPA), for elliptic curve point multiplication (ECPM). There are various coordinates to represent elliptic curve points, such as affine coordinates and projective coordinates, which will be discussed in this chapter. Moreover, the mathematics and the hardware implementation procedure behind point multiplication will be explained clearly, where point (or scalar) multiplication is the fundamental building block of the elliptic curve digital signature algorithm (ECDSA). Finally, this chapter concludes with a summary of designs proposed in this dissertation.

2.1 Overview of Cryptography

Cryptology is a more general term than Cryptography, defined as the science of secure communication. The cryptology domain concerns mathematics as well as computer science because modern cryptology combines digital data and digital systems. Therefore,

a secure mathematical algorithm and its efficient implementation in a digital system is essential for cryptographic techniques. Cryptology consists of two main branches: cryptography and cryptanalysis.

Cryptography is the science or study of techniques of secret writing with hiding information (message) from malicious adversaries. Also, it is the mathematical techniques to encrypt and decrypt data and supporting information security, such as confidentiality, integrity, non-repudiation, and authentication. In addition, it is handled with the design and implementation of algorithms and facilitates cryptographic services. The cryptographic algorithm for providing information security is classified in two groups: private-key cryptography and public-key cryptography (PKC) [2, 3, 12, 26, 27].

Cryptanalysis is the science of getting the plaintext of an information (message) without knowing the key. It is also the study of methods to examine the security of a cryptographic algorithm or to find the weakness of a cryptographic system. Successful cryptanalysis may get the key or the plaintext. Modern cryptanalysis is about breaking mathematical entities and the physical devices implementing them. An attempted cryptanalysis is named an attack. Various attacks exist, such as analytical attacks and brute-force attacks. Moreover, there have been some hardware attacks, such as side-channel attacks (SCA), electromagnetic attacks (EMA), fault attacks (FA), timing attacks, power attacks [2, 3, 12, 26, 27].

2.2 Private-key Cryptography

Private-key cryptography (or symmetric, or single-, or secret-key cryptography) is a cryptosystem which has the capability to secure exchange of information (messages) between the two ends. In symmetric cryptography, the same key is shared between the sender and recipient to perform both encryption and decryption, where encryption is the process to

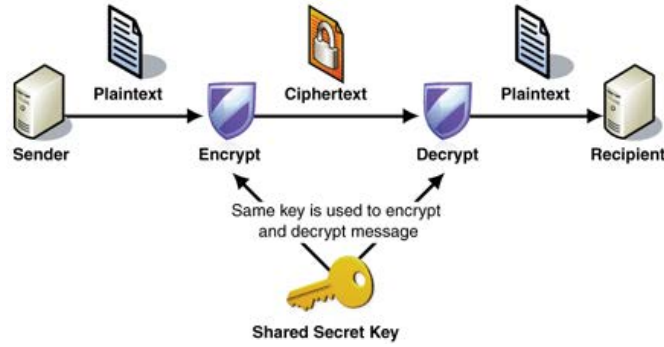


Figure 2.1: *Private-key cryptography (or Symmetric cryptography) [28]*

produce ciphertext from plaintext and decryption is the process to recover plaintext from ciphertext. The process of encrypting and decrypting data with a shared key is illustrated in Fig. 2.1. To perform encryption, private-key cryptography is commonly used. However, it has a key distribution problem because for encryption and decryption the same key is used. There are various symmetric algorithms currently in use, such as the advanced encryption standard (AES) or Rijndael AES, Triple data encryption standard (3DES), DES, Blowfish, Twofish, Serpent, CAST5, Kuznyechik, Rivest Cipher 4 (RC4), Skipjack, and IDEA [26, 28–30].

2.3 Public-key Cryptography

Public-key cryptography (PKC) is an asymmetric cryptosystem, and can be used for a secure distribution of keys. The process of PKC is illustrated in Fig. 2.2. As shown in Fig. 2.2, the sender encrypts plaintext (message or data) with one key and the recipient employs a different key to decrypt ciphertext. Therefore, two different keys are used to encrypt and decrypt information (message), and the problem of key distribution is solved by PKC. The encryption and decryption keys are generally called public/private key pairs [17, 26, 28]. The PKC was first introduced by Diffie and Hellman (DH) [7] in 1976 and later on proposed by Markle [31] in 1978. Numerous PKCs have been proposed in the

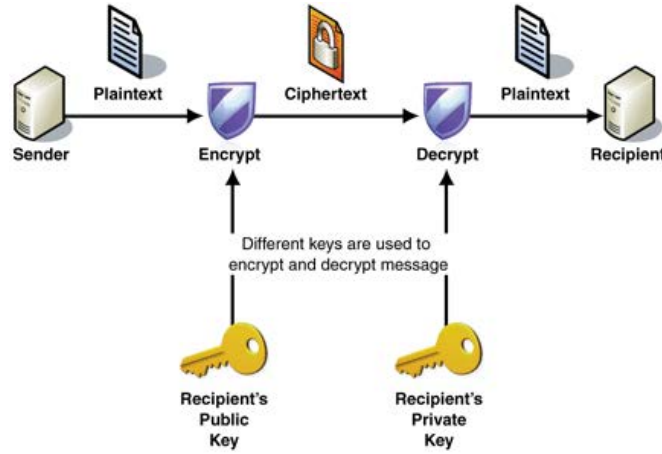


Figure 2.2: *Public-key cryptography (PKC) (or Asymmetric cryptography) [28]*

available literature. Based on the number theory, PKC algorithms can be classified into three sub-groups: 1) discrete logarithm problem (DLP), 2) integer factorisation problem (IFP), and 3) elliptic curve discrete logarithm problem (ECDLP).

2.3.1 Discrete Logarithm Problem

The key agreement protocol using the discrete logarithm system was first introduced in 1976 [7]. Later on, discrete logarithm public-key encryption and the signature scheme were described by ElGamal [9] in 1984. The DLP depends on the difficulty of evaluating logarithms in a large finite field, where the hardness increases with increasing field size. Mathematically the discrete logarithm system is defined by: let g be a generator of an abelian group G (or multiplicative cyclic group) of n elements, and given $g, h \in G$, find x such that $g^x = h$. The DLP is not always a hard problem because the hardness of finding discrete logarithms depends upon groups, e.g, Z_p^* which is a choice of groups for discrete logarithm cryptosystems, where p is a prime number. However, the Pohlig-Hellman algorithm [32] can solve the DLP very efficiently when the multiplicative inverse of p is a product of small primes. Based on discrete logarithm systems, numerous schemes

have been proposed in the literature, e.g. the digital signature algorithm (DSA), the DH key agreement protocol, ElGamal public-key encryption scheme, and the Schnorr signature scheme. Details of discrete logarithm schemes can be found in [2, 12, 27].

2.3.2 Integer Factoring Problem

Integer factorisation, in number theory, is the factoring of a composite number into a product of smaller integers. For example, it can be produced by multiplying together two smaller positive integers. The process is defined as prime factorisation if the factors are prime numbers. To compute the prime factorisation of a given integer n , is the hardness of an RSA cryptosystem. Therefore, the security of RSA public-key encryption and signature schemes relies on the difficulty of the integer factorisation problem (IFP). In RSA, the IFP is the product of p and q (large primes) which are factors of a composite number n . However, when the factors of n are known, then the RSA problem can easily be solved, as explained in [2, 17]. The RSA cryptosystem generally consists of four major steps: key generation, key distribution, encryption, and decryption. Details of the RSA cryptosystem can be found in [8, 17, 33].

2.3.3 Elliptic Curve Discrete Logarithm Problem

The last group of public-key algorithms relies on the discrete logarithm problem (DLP) over an elliptic curve, simply called the elliptic curve discrete logarithm problem (ECDLP). It is a difficult mathematical problem in which, if two elliptic curve points P and Q over a finite field are given, then it is computationally infeasible to compute the positive integer $[k]$ from the point multiplication $Q = [k] \cdot P$ because the size of k is large, e.g. 224 or 256 bits. This is the hardness of ECDLP [2, 17]. However, if k and P are known, then it is easy to calculate Q , where $Q = [k] \cdot P$ is the ECPM over a finite field. On the other hand, modular exponentiation is the most computationally intensive operation

for discrete logarithm systems, e.g. the RSA cryptosystem. Modular exponentiation operations are accomplished using very long operands to meet the required size, whereas the operands are smaller in the ECDLP [2, 17]. However, the smaller operands (field bit length) of an elliptic curve cryptosystem can provide equivalent security to large operands of the RSA cryptosystem. A comparison of equivalent cryptographic key sizes for various cryptographic algorithms is discussed in Sections 4.3.2 and 9.3.1. Therefore, the main focus in this dissertation is the high-performance hardware implementation of ECC. The implementation hierarchy of the ECC operations is discussed in Sections 5.3, 6.3, 7.4, and 8.3.1. In this chapter, the subsequent sections followed from the bottom level (e.g. finite field arithmetic) to the top level (e.g. ECPM) of an ECC.

2.4 Finite Field Arithmetic

The bottom level in the hierarchy of elliptic curve cryptosystems consists of finite field arithmetic (FFA). This section presents a brief introduction to the FFA which is the most important for many public-key cryptosystems in use today, including ECC. It is noted that a high-performance hardware implementation of FFA is vital for an ECC processor. The group and field theorems are discussed first in the following section.

2.4.1 Groups, Rings, and Fields

In this section, the definitions of the group (G), ring (R), and field (\mathbb{F}) are provided, and their properties are discussed in Section 4.3.1.

A **group** $(G, *)$ consists of a set of elements together with a binary operation $*$ which satisfies some important properties, given in Section 4.3.1. The group is said to be finite if G is a finite set; in this case the number of elements in G is called its order [12, 27].

A **ring** $(R, +, \times)$ is a set R with two binary operations arbitrarily denoted $+$ (addition)

and \times (multiplication) which follow the field (\mathbb{F}) properties.

A **field** is a ring (R) in which the non-zero elements form an abelian group under \times [2, 12, 27]. A field $(\mathbb{F}, +, \times)$ is a set of elements with two operations, denoted as $+$ (addition) and \times (multiplication), satisfying some important properties, given in Section 4.3.1.

2.4.2 Finite Field

When the field consists of a finite number of elements, it is simply called a Galois field (GF) or finite field. A finite cyclic group is needed for a cryptosystem in which the group operations are efficiently calculable, but the DLP is difficult to solve. When the underlying field is finite, then elliptic curve groups appear to fulfil the discrete logarithm problem criteria [2, 12, 27]. A finite field is denoted as $GF(q = p^m)$ where m is a positive integer and p is a prime number called the characteristic of field \mathbb{F} . Three types of finite field exist in the literature: prime field, extension field, and binary field. The field is said to be a prime field if $m = 1$ and an extension field [34] if m is greater than 2. A Galois field is said to be characteristic-two finite field or a binary field if $q = 2^m$. In this dissertation, the binary field $GF(2^m)$ and the prime field $GF(p)$ are considered for hardware implementations of ECC [2, 35].

2.5 Binary Field \mathbb{F}_2^m Arithmetic Background

The binary field \mathbb{F}_2^m or $GF(2^m)$ is a finite field, also called a characteristic-two finite fields which contains 2^m different elements and \mathbb{F}_2 has two elements 0 and 1 with respect to a basis. A binary field is quite simple as well as efficient for hardware implementation using more-efficient modulo-2 arithmetic because they provide carry-free operations. It can be represented using a normal basis (NB), a dual basis (DB), a redundant basis, or a

polynomial basis (PB). However NB and PB are two common types of bases for hardware implementations, whereas PB is beneficial from an implementation perspective because a field element can be characterised by m binary bits [2, 35].

2.5.1 Normal Basis

A normal basis (NB) exists for all positive integer m in the binary field \mathbb{F}_2^m . In a NB, field elements \mathbb{F}_2^m (or $GF(2^m)$) are defined with a basis of the form $\{\beta, \beta^2, \beta^{2^2}, \dots, \beta^{2^{m-1}}\}$, where β is a root of a reduction polynomial of degree m . Suppose that $U = \sum_{i=0}^{m-1} u_i \beta^{2^i}$ and $V = \sum_{i=0}^{m-1} v_i \beta^{2^i}$, where $u_i, v_i \in \mathbb{F}_2$. Then, addition can be computed by the bit-wise XOR operation: $U + V = \sum_{i=0}^{m-1} (u_i + v_i) \beta^{2^i}$, which is the same as polynomial-basis addition. For an addition, the identity element is $(0, 0, 0, \dots, 0, 0)$ and for a multiplication, the identity element is $(1, 1, 1, \dots, 1, 1)$ as $1 = \beta + \beta^2 + \dots + \beta^{2^{m-1}}$. One main advantage of a NB is that squaring can be performed efficiently by using a simple shift operation, where the squaring of U can be represented as $U^2 = (\sum_{i=0}^{m-1} u_i \beta^{2^i})^2 = \sum_{i=0}^{m-1} u_i \beta^{2^{i+1}}$ and one can get $\beta = \beta^{2^m}$ which can be defined as Fermat's Little Theorem [2, 13, 36]. There are different types of NB available in the literature, such as Gaussian NB (GNB) and optimal NB (ONB). Other operations in an NB, e.g. multiplication and inversion, may have similar complexity to the polynomial-basis representation. Details of NB operations for binary fields are given in [36–39].

2.5.2 Polynomial Basis

A polynomial basis (PB) is a well-known binary extension field used to represent field elements. In this representation, the elements of \mathbb{F}_2^m are the binary polynomials of degree at most $m - 1$, i.e.

$$\mathbb{F}_2^m = u_{m-1}.x^{m-1} + u_{m-2}.x^{m-2} + \dots + u_1.x + u_0 = \sum_{i=0}^{m-1} u_i x^i : u_i \in \{0, 1\},$$

where u_{m-1} is denoted as the most significant bit (MSB) and u_0 is denoted as the least significant bit (LSB). The identity element of addition in PB can be represented as $(0, 0, \dots, 0, 0)$ and multiplication's identity element is $(0, 0, \dots, 0, 1)$.

In PB, $x^4 + x^3 + x + 1$ is a polynomial-basis representation for the 5-bit number 11011_2 , or $x^3 + 1$ is a polynomial-basis representation of the 4-bit number 1001_2 . For an irreducible polynomial, let $(f(x))$ be an irreducible or reduction binary polynomial of degree m , $f(x) = x^m + G(x) = x^m + \sum_{i=0}^{m-1} g_i x^i$ where $g_i \in \{0, 1\}$ for $i = 1, \dots, m-1$ and $g_0 = 1$ [2, 40]. For example, $f(x) = x^4 + x + 1 = (10011)_2$ is a reduction polynomial of the finite field $\text{GF}(2^4)$. This dissertation uses a PB for hardware implementations of FFA, such as addition, multiplication, squaring, and inversion in binary fields \mathbb{F}_2^m .

2.5.3 Addition in \mathbb{F}_2^m

Addition in \mathbb{F}_2^m or finite field addition or modulo-2 addition is the simplest operation over the binary field. In PB, it is called polynomial addition and is simply a bit-wise XOR (or \oplus) for hardware implementation. Adding two elements $U(x)$ and $V(x)$ of size m in $\text{GF}(2^m)$ can be achieved as depicted in (2.1) [41]:

$$Z(x) = U(x) + V(x) = \sum_{i=0}^{m-1} u_i x^i + \sum_{i=0}^{m-1} v_i x^i = \sum_{i=0}^{m-1} (u_i + v_i) x^i = \sum_{i=0}^{m-1} z_i x^i \quad (2.1)$$

where $z_i = (u_i + v_i) \bmod 2 = u_i \oplus v_i$ and $u_i, v_i, z_i \in \mathbb{F}_2$. For example, if

$$\begin{cases} U = 1100_2, \text{ and } V = 0110_2 \text{ over the field } \text{GF}(2^4), \text{ then} \\ Z = (U + V) = (U \oplus V) = (1100_2 \oplus 0110_2) = 1010_2. \end{cases}$$

The hardware implementation algorithm of finite field addition and the corresponding hardware architecture are explained in Section 3.4.1. Addition in \mathbb{F}_2^m is a carry-free operation and it can be implemented in parallel, taking only one clock cycle.

Subtraction in \mathbb{F}_2^m is the same as finite field addition and is based on the definition of the identity element, e.g. $U(x) + U(x) = 0$ or $U(x) + (-U(x)) = 0$.

2.5.4 Multiplication in \mathbb{F}_2^m

Finite field \mathbb{F}_2^m multiplication or polynomial multiplication is the second-most expensive operation at the arithmetic level of an ECC processor. Besides, this is the most important arithmetic operation for ECPM in Jacobian coordinates. Therefore, it has been necessary to expend substantial efforts in designing an efficient finite field multiplier. Numerous methods or algorithms have been proposed in the literature to perform polynomial multiplication, including multiplication with an interleaved modular reduction algorithm, bit-serial multiplication, the Karatsuba-Ofman algorithm, Montgomery multiplication, higher-radix multipliers, digit-serial multiplication, and digit-parallel multiplication [2, 42–45]. The basic example of polynomial multiplication over $\text{GF}(2^4)$ is as follows: Assume that

$$\left\{ \begin{array}{l} f(x) = x^4 + x + 1 = (10011)_2, \quad U(x) = x^3 + x^2 + x + 1 = (1111)_2, \\ \text{and } V(x) = x^3 + x^2 + x = (1110)_2, \text{ then} \\ Z(x) = U(x).V(x) = (x^3 + x^2 + x + 1).(x^3 + x^2 + x) \\ \quad = x^6 + 2.x^5 + 3.x^4 + 3.x^3 + 2.x^2 + x \\ \quad = x^6 + x^4 + x^3 + x \text{ (applying mod } -2 \text{ operation)} \\ \quad = (x^2 + 1)(x^4 + x + 1) + x^2 + 1 = x^2 + 1 \\ \quad = (0101)_2 \text{ (after polynomial reduction).} \end{array} \right.$$

The polynomial multiplication result must be reduced to a degree < 4 by the irreducible polynomial $f(x) = x^4 + x + 1$. In this dissertation, bit-serial, parallel, and digit-serial methods have been emphasised for hardware implementation.

Bit-Serial Multiplication

Bit-serial polynomial multiplication using the interleaved method is a well-known algorithm for hardware implementation. In this algorithm, instead of separate multiplication

and reduction operations, the two operations can be combined or interleaved [41]. Let $U(x)$ and $V(x)$ be two inputs and $Z(x)$ be their output, then multiplication in \mathbb{F}_2^m can be achieved as depicted in (2.2):

$$Z(x) = U(x).V(x) = U(x). \sum_{i=0}^{m-1} v_i.x^i = \sum_{i=0}^{m-1} (U(x).v_i).x^i \quad (2.2)$$

It computes the product of two polynomials and performs modular reduction with $f(x)$ concurrently, where $f(x)$ is a constant reduction polynomial of degree m and its operation is different from simple integer multiplication. The interleaved algorithm and corresponding hardware architecture are discussed in Section 5.4.3. The detailed operations of a step-by-step solution for bit-serial polynomial multiplication are depicted in Section 5.4.3, Table 5.3. In addition, a bit-serial polynomial multiplication algorithm based on the modified interleaved method and a corresponding hardware architecture are proposed in Section 3.4.2. The result of polynomial multiplication is achieved after m clock cycles. The area complexity of bit-serial multiplication is only $O(m)$. In addition, bit-serial multiplication is better for low-power consumption design. However, it is not as fast as bit-parallel or digit-serial/digit-parallel multiplication.

Parallel Multiplication

Recall that the bit-serial finite field multipliers in the previous section require fewer hardware resources and less power consumption, but they are very slow because of the m clock cycles required. On the other hand, parallel multipliers are very fast because they take fewer clock cycles, which is required for a high-speed ECP. In this dissertation, two parallel architectures have been developed for an ECP based on Algorithm 6.2, which is presented in Section 6.6, then one multiplier architecture chosen based on their performances. The proposed architecture is performed fully in parallel, which is well suited for a high-speed ECP. The parallel multiplier area complexity is $O(m^2)$, which is far more than a bit-serial multiplier, whose area complexity is $O(m)$.

Digit-Serial Multiplication

Recalling the bit-serial and parallel multipliers in the previous two sections, the bit-serial multiplier is better with the area and power, on the other hand, a parallel multiplier is better with timing only. Therefore, a trade-off between area, time, and power is necessary for well-designed ECP. The digit-serial multipliers fill the gap because these are well suited to a high-performance ECP, hence these have been well studied. Digit-serial multiplication using the traditional approach was introduced in [46] and [43].

Traditional digit-serial multiplier: Let $U(x)$ and $V(x)$ be two input elements in \mathbb{F}_2^m and $Z(x)$ be their output in \mathbb{F}_2^m . The final result $Z(x) = U(x) \cdot V(x) \bmod f(x)$ over \mathbb{F}_2^m is achieved by a reduction polynomial $f(x)$. Suppose that $q = \lceil m/d \rceil$, where m is the field size and d is the digit size. A field element needs to be padded with $qd - m$ bit zeros, when m is not a multiple of dq . The input V over \mathbb{F}_2^m can be written as $V = \sum_{i=0}^{q-1} v_i x^{id}$, where $V_i = V_{id} + V_{id+1}x + \dots + V_{id+d-1}x^{d-1}$. The product Z can be rewritten based on LSD-first multiplication,

$$\begin{aligned} Z(x) &= U(x) \cdot V(x) \bmod f(x) = U \cdot (V_0 + V_1 x^d + \dots + V_{q-1} x^{d(q-1)}) \bmod f(x) \\ &= (Z_{v(0)} + Z_{v(1)} + \dots + Z_{v(q-1)}) \bmod f(x) \end{aligned} \quad (2.3)$$

where $Z_{v(i)} = U^{(i)} V_i$, $U_v = U_v \cdot x^d \bmod f(x)$, and $U^0 = U$. The detailed traditional digit-serial multiplier algorithm and the corresponding hardware architecture is explained in Section 3.4.2. Usually, a digit-serial multiplier using the traditional approach is better for low area \times time (AT) complexity. In addition, digit-serial multipliers speed up the multiplication process significantly compared to the bit-serial multiplier, as multiple partial products are generated and added per clock cycle, thus reducing the overall number of clock cycles. Note that a latency of m clock cycles is required for the bit-serial multipliers, whereas the basic digit-serial multiplier's latency is $\lceil m/d \rceil$ which is very low, but with a slightly higher area complexity. In this method, a separate multiplication and reduction unit is needed.

Modified digit-serial multiplier: In this dissertation, a modified pipelined digit-serial multiplier architecture over \mathbb{F}_2^m is proposed with a very low latency of $2 * \lceil \sqrt{m/d} \rceil$, whereas the traditional digit-serial multiplier requires $\lceil m/d \rceil$ latency. For a modified digit-serial multiplier, let $U = (u_{m-1}, \dots, u_1, u_0) = \sum_{i=0}^{m-1} u_i x^i$, $V = (v_{m-1}, \dots, v_1, v_0) = \sum_{i=0}^{m-1} v_i x^i$, and $f(x) = x^m + r(x)$. In this method, a pair of integers p and k is now used such that $kp = q$ and $k = \sqrt{q}$. Zeros also need to be padded to the multiplicand V so that $q = \lceil m/d \rceil = kp$ can be satisfied. Now V can be presented as $\sum_{i=0}^{kp-1} V_i x^{id}$, where $V_i = \sum_{j=0}^{d-1} v_{(id+j)} x^j \bmod f(x)$. Then, the output Z can be rewritten as (2.4)

$$Z(x) = U(x) \cdot V(x) \bmod f(x) = \sum_{i=0}^{kp-1} U \cdot V_i x^{id} \bmod f(x) = \sum_{i=0}^{p-1} Z_i x^{dki} \bmod f(x) \quad (2.4)$$

where $Z_i = \sum_{j=0}^{k-1} U \cdot V_{ik+j} x^{jd} = \sum_{j=0}^{k-1} U^{(jd)} \cdot V_{ik+j} = \sum_{j=0}^{k-1} Z_{ij}$, and $Z_{ij} = V_{ik+j} U^{(jd)}$. The partial product of (2.4) needs to be simplified as, $Z_{v(i)} = Z x^{dki} = \sum_{j=0}^{k-1} U \cdot V_{ik+j} x^{jd} x^{dki} = \sum_{j=0}^{k-1} U_i^{(jd)} V_{ik+j}$, where $U_i = x^{dki} U \bmod f(x) = x^{kd} U \bmod f(x)$. Finally, the output Z can be written as $Z = \sum_{i=0}^{p-1} Z_v \bmod f(x)$.

In this approach, the top-level architecture requires k processing units (PUs), which is the same approach as the systolic digit-serial multiplier. Note that the systolic multipliers are better for high-throughput ECP implementations. The detailed modified (or systolic) digit-serial multiplier hardware architecture is proposed and demonstrated in Section 3.4.2 based on Algorithm 3.4. The modified digit-serial multiplier significantly improves the area \times time \times energy (ATE) performance in an ASIC platform. In this dissertation, different digit sizes, e.g. 1, 2, 4, 8, 16, 32, and 64 bits in digit-serial multipliers are implemented for all five NIST binary fields $\text{GF}(2^m)$, presented in the next chapter.

2.5.5 Squaring in \mathbb{F}_2^m

Finite field squaring or polynomial squaring is similar to polynomial multiplication because squaring over \mathbb{F}_2^m is the polynomial multiplication of two identical binary poly-

nomials. A squaring operation is often used since it can be more efficient than field multiplication. This consideration depends on the performance of the field multiplication utilised, since if multiplication becomes more efficient than squaring, then the squaring method can be replaced. Therefore, it can be implemented by a multiplier and the performance can be improved through the optimisation of the architecture. Squaring over \mathbb{F}_2^m has less difficulty because $U(x)^2 \bmod f(x)$ is a linear operation. It can be computed as shown in (2.5):

$$\begin{aligned} Z(x) = U(x)^2 \bmod f(x) &= \left(\sum_{i=0}^{m-1} U_i x_i \right)^2 \bmod f(x) \\ &= u_{m-1}.x^{2m-2} + \dots + u_2.x^4 + u_1.x^2 + u_0 \bmod f(x) = \sum_{i=0}^{m-1} u_i x^{2i} \bmod f(x) \end{aligned} \quad (2.5)$$

The squaring operation in $\text{GF}(2^m)$ of $Z(x) = U(x)^2$ is achieved by setting a 0 bit between consecutive bits of the binary representation of $U(x)$. Hence, it can be implemented efficiently using a fixed irreducible polynomial $f(x)$ to get the result in one clock cycle without huge hardware resources [47]. The hardware architecture of squaring over \mathbb{F}_2^m is presented in Section 5.4.4.

2.5.6 Inversion in \mathbb{F}_2^m

Inversion over \mathbb{F}_2^m in a PB is the most expensive (or time-consuming) operation at the arithmetic level of an ECC processor. Therefore, the inversion operation should be avoided whenever possible because it slows down the whole cryptosystem [48]. However, it is mandatory as well as an important operation for point multiplication in affine coordinates. On the other hand, inversion for each group operation can be avoided by using projective coordinates. In practice, inversion is required only once per point multiplication to convert affine coordinates from Jacobian coordinates. Hence, it is essential to expend some effort to design an efficient finite field inversion. Inversion of a non-zero field element $U(x) \in \mathbb{F}_2^m$

can be presented as $Z(x) = 1/U(x) \bmod f(x)$, where it should satisfy the condition $UZ = 1 \bmod f(x)$ and 1 is the multiplicative identity element. There are mainly two kinds of inversion algorithm available: (1) Extended Euclidean-based algorithms (EEA) and (2) Fermat's Little Theorem (FLT) based algorithms. Based on FLT, for any a in a finite field with q elements $a^q = a$ or $a^{q-2} = a^{-1}$. For the binary field $\text{GF}(2^m)$ $q = 2^m$, hence $a^{-1} = a^{2^m-2}$. Therefore, the main difficulty for the FLT method is the large exponentiation, which is a very expensive operation. On the other hand, EEA-based algorithms include (a) the simplest EEA, (b) the almost inversion algorithm, and (c) the modified almost inversion algorithm. Also, other inversion algorithms exist in the literature, such as the Itoh-Tsujii inversion algorithm, which is based on FLT [49] and the Montgomery inversion algorithm [50]. The basic inversion algorithm for $\text{GF}(2^m)$ was proposed in [51], then implemented in [52] and [53]. In this dissertation, the modified almost inversion algorithm is utilised, being the most commonly used as well as most efficient algorithm for polynomial inversion [2, 52], presented in Section 5.4.5. The corresponding hardware architecture is demonstrated and explained in Section 3.4.4. Based on this algorithm, an efficient polynomial inversion architecture is achieved. In this method, the polynomial reduction is performed by using the addition operation when the degree of the polynomial is m or higher than m . The field inversion requires $2m + 1$ clock cycles to complete. An example of inversion is given below: Assume $f(x) = x^4 + x + 1 = (10011)_2$, $U(x) = x^3 + x^2 + x + 1 = (1111)_2$, then

$$\left\{ \begin{aligned} Z(x) &= Z(x) = 1/U(x) \bmod f(x) = 1/(1111)_2 = 1/g^{12} = g^{-12} = g^{(15-12)} = g^3 = (1000)_2. \end{aligned} \right.$$

To check that this is the multiplicative inverse of $U(x)$ their multiplicative identity should be one, e.g.

$$\left\{ \begin{aligned} Z(x) &= 1/U(x) \bmod f(x) = 1/(1111)_2 = 1/g^{12} = g^{-12} = g^{(15-12)} \\ &= (x^3 + x^2 + x + 1)x^3 = x^6 + x^5 + x^4 + x^3 = (x^4 + x + 1)(x^2 + x + 1) + 1 = 1. \end{aligned} \right.$$

The division $Z(x) = V(x)/U(x) \bmod f(x)$ over \mathbb{F}_2^m is performed by first finding the inverse $U^{-1}(x)$ and then performing the multiplication, where $V(x) \cdot U^{-1}(x) = V(x)/U(x)$.

2.5.7 Complexity Analysis of Finite Field Arithmetic over \mathbb{F}_2^m

Finite field addition, multiplication, and inversion operations have different latency (the number of clock cycles). Complexity analysis of all operations in terms of clock cycles is reported in Section 3.4, Table 3.6. As shown in Table 3.6, addition is the simplest operation, needing only a single clock cycle, whereas inversion is the most time-consuming and complex operation, taking $2m + 1$ clock cycles.

2.5.8 Reduction in \mathbb{F}_2^m

FFA for a binary field is performed modulo the corresponding reduction polynomial $f(x)$ of the binary field used. A modular reduction needs to be run on the result of the multiplication, squaring, and inversion to ensure that it exists within the binary field \mathbb{F}_2^m chosen. The reduction is performed by connecting all bit positions representing a binary polynomial with a degree more than m to be selectors to each multiplexer. For reduction, the NIST irreducible polynomial $f(x)$ of the corresponding binary field is used, so that a binary polynomial $Z_v(x)$ is reduced to $Z(x)$, where $Z(x) = Z_v(x) \bmod f(x)$. The reduction used in the architecture of traditional and systolic digit-serial multiplication to perform $\bmod f(x)$ is presented in Section 3.4.3, Fig. 3.6.

2.5.9 NIST Reduction Polynomials over \mathbb{F}_2^m

There are various elliptic curve domain parameters over \mathbb{F}_2^m including the field size m and irreducible binary polynomial or field generator $f(x)$ of degree m , which specifies the PB representation of \mathbb{F}_2^m . The irreducible polynomials for either the random curves or

Koblitz curves recommended by NIST in the FIPS 186-2 standard for 163, 233, 283, 409 and 571 bits are presented in Table 2.1. Certicom has also provided NIST-recommended EC domain parameters, standard for efficient cryptography in SEC2 [54]. The details of the fast reduction modulo algorithms for the NIST irreducible polynomial $f(x)$ can be found in [2].

Table 2.1: *NIST recommended irreducible polynomials over $GF(2^m)$ [22]*

Field size m	Irreducible polynomial $f(x)$
163	$f(x) = x^{163} + x^7 + x^6 + x^3 + 1$
233	$f(x) = x^{233} + x^{74} + 1$
283	$f(x) = x^{283} + x^{12} + x^7 + x^5 + 1$
409	$f(x) = x^{409} + x^{87} + 1$
571	$f(x) = x^{571} + x^{10} + x^5 + x^2 + 1$

2.6 Prime Field \mathbb{F}_p Arithmetic Background

The most basic and lowest level of point multiplication is modular arithmetic over prime fields \mathbb{F}_p . All modular arithmetic operations over a prime field \mathbb{F}_p (or $GF(p)$) are accomplished using modulo p , consisting of the integers between 0 and $p - 1$. For any integer x , $x \bmod p$ can be defined as the unique integer remainder q , $0 \leq q \leq p - 1$, obtained upon dividing x by p ; this operation is called reduction modulo p , which is required for cryptographic schemes. In cryptographic applications, e.g. an elliptic curve cryptosystem, x and p are large integers, such as 224 or 256 bits, and p must be a prime number.

The computation of an elliptic curve cryptosystem over $GF(p)$ consists of three levels: modular arithmetic, group operations, and point multiplication. The detailed hierarchy of ECC over the prime field $GF(p)$ is presented graphically in Section 8.3.1. The finite

field modular arithmetic (FFMA) unit, such as the modular addition, subtraction, multiplication, squaring, and inversion operations, is mandatory to implement elliptic curve group operations over $GF(p)$. The group operations are then utilised to develop point multiplication, where ECPM can be utilised for the elliptic curve digital signature algorithm (ECDSA). In this chapter, the subsequent section discusses modular arithmetic first, then followed by group operations and point multiplication.

2.6.1 Modular Adder and Subtractor over \mathbb{F}_p

The modular adder and subtractor are the fundamental operations for an ECP. The basic modular adder and subtractor operations are presented in (2.6).

$$Z = (x \pm y) \pmod{p}, \quad (2.6)$$

The output Z of the modular addition is obtained by adding x and y and then subtracting the modulus p from the sum until the output is less than p . The modular reduction is usually very simple for an adder because the inputs x and y are in the range between 0 and $p-1$, hence the output Z must be $\leq 2p$. Consequently, the output Z can be reduced by simply subtracting p . Modular subtraction is very similar to modular addition. In a modular subtractor, if $x \geq y$ then it can be computed easily by simple subtraction or 2's-complement addition and the output Z will be in the range. Otherwise, if $x \leq y$ the modulus p should be added to the input x before the subtraction starts [27]. A basic example of modular addition is as follows; consider

$$\begin{cases} x = 28, y = 20, \text{ and modulus } p = 29, \text{ then} \\ z = (x + y) \pmod{p} = (28 + 20) \pmod{29} = (48 - 29) = 19. \end{cases}$$

The basic modular subtraction example is as follows; consider

$$\begin{cases} x = 20, y = 28, \text{ and modulus } p = 29, \text{ then} \\ z = (x - y) \pmod{p} = (20 - 28) \pmod{29} = (20 + 29 - 28) \pmod{29} = 21. \end{cases}$$

The modular adder and subtractor can be implemented separately or in combination. The adder or subtractor takes only one clock cycle to implement, whereas the combined unit takes two clock cycles with a slightly higher area complexity. In this dissertation, both separate and combined modular adders and subtractors are implemented; see in Section 11.4.1.

2.6.2 Modular Multiplier/Squarer over \mathbb{F}_p

Modular multiplication, including squaring, is expensive and the most important operation over the prime field \mathbb{F}_p . To implement high-performance public-key cryptosystems, e.g. RSA, Diffie-Hellman key exchange, and elliptic curve cryptosystems, efficient implementation of a modular multiplier is required. The basic modular multiplier operation of two elements $x, y \in \mathbb{F}_p$ is presented as $Z = (x \times y) \bmod p$. A basic modular multiplication example is as follows: consider $x = 28, y = 20$, and $p = 29$, then

$$\left\{ z = (x \times y) \bmod p = (28 \times 20) \bmod 29 = (19 \times 29 + 9) \bmod 29 = 9. \right.$$

For a normal computation, the reduction operation can be performed by divisions, but in a hardware implementation modular reduction using division is quite slow. Various methods to perform modular multiplication are reported in [55–65]. Two efficient methods, such as Montgomery modular reduction proposed in 1985 [66] and Barrett modular reduction proposed in 1987 [67], are popular choices for hardware implementations. The Barrett algorithm consists of division operations which are usually slow, but it can be implemented efficiently by using right-shift operations. It has been shown in [68] that Montgomery modular multiplication is slightly faster than the Barrett algorithm. For this reason, the Montgomery modular multiplication method is used in this dissertation. A detailed comparison of Montgomery and Barrett modular multiplication algorithms is presented in [68].

The Montgomery modular product can be presented as $R = \text{Montgpro}(x, y, p) = x \times y \times 2^{-(m+2)} \bmod p$. In this method, the costly trial division by modulus p can be avoided, keeping the intermediate result bounded to $(m + 2)$ bits all over the calculation. This method calculates the Montgomery product using a series of simple additions and right shifts. In this method, two Montgomery multiplications are required for one complete modular multiplication [69]. The modular multiplier architecture corresponding to the Montgomery algorithm is presented in Section 8.3.2, Fig. 8.2. Also, most of the other modular multiplication algorithms follow the basic concept of the Montgomery method. For example, [65] proposed a fast interleaved modular multiplier based on the Barrett and Montgomery methods. In this dissertation, there are also two modular multiplication algorithms, and their corresponding architectures are proposed based on an interleaved method. The first one is the modified interleaved method and the second one is radix-4 modular multiplication.

In the first method, the multiplication and the calculation of the remainder of division are interleaved. This is a very simple method as the first operand is multiplied bitwise with the second operand, then added to the intermediate result. The benefit of the interleaved method is that the intermediate result is only one or two bits larger than the operands because the intermediate result is always reduced by taking the modulus [60, 61]. The detailed hardware architectures are explained in Section 10.4.1, Fig. 10.2 and Section 11.4.2, Figs. 11.2 and 11.3(a). The latency (number of clock cycles) of this method is $m + 1$, where m is the bit length of the operands A, B or p . To reduce the cycle counts, a higher radix, e.g. radix-4 modular multiplier, is required.

In the second method, a novel radix-4 modular multiplication algorithm is proposed. It can be used for high-performance public-key cryptosystems, including ECP, because of their lower cycle count than the bit-serial multiplier [55–57]. In a radix-4 multiplier, it processes 2 bits at a time, which halves the total number of clock cycles. Usually,

higher-radix modular multipliers are better with speed or timing, but the area increases significantly [70]. For this reason even higher-radix modular multipliers, such as radix-8, are not suitable for low-AT-complexity designs. The hardware architecture based on the proposed radix-4 modular multiplication algorithm 11.3 is depicted in Section 11.4.2, Fig. 11.3(b). The latency of this method is only $m/2 + 1$ clock cycles to complete the modular multiplication. Therefore, a high-performance modular multiplier is designed which is essential for ECP in Jacobian coordinates.

A **modular squarer** is similar to a modular multiplier except that only one input is required for a modular squarer rather than the two inputs for a modular multiplier; otherwise all other operations are the same as for modular multiplication.

2.6.3 Modular Inversion over \mathbb{F}_p

The modular inversion ($U^{-1} \bmod p$) over a prime field is the most complex, most expensive, and the slowest among all other ECC arithmetic computations. Therefore, it is impossible to avoid modular inversion for ECP over prime fields $GF(p)$. However, in affine coordinates inversion is mandatory for elliptic curve group operations, hence ECPM. On the other hand, it can be avoided for group operations in Jacobian coordinates. However to convert back to affine coordinates, still one modular inversion is required.

The inverse of an integer a modulo p is defined as an integer R such that $a.R \equiv 1 \pmod{p}$. The classical definition of the modular inversion operation can be presented as

$$R = a^{-1} \pmod{p} \tag{2.7}$$

where p is a prime and a is an integer. From (2.7), the inverse of a exists if and only if a is relatively prime with p . Therefore, the Greatest Common Divisor (gcd) of a and p

must be 1 or $\gcd(a, p) = 1$. A modular inversion example is as follows: consider

$$\begin{cases} a = 12 \text{ and modulus } p = 29, \text{ then} \\ R = a^{-1} \pmod{p} = 12^{-1} \pmod{29} = 17 \text{ because } 12 \times 17 \pmod{29} = 1. \end{cases}$$

To compute modular inversion, numerous methods have been presented in the literature [71–83]; two well-known methods are often used: the first is Fermat’s Little Theorem (FLT) and the second is based on the Extended Euclidean gcd Algorithm (EEA). The multiplicative inverse of the FLT method is obtained by modular exponentiation, which is a very expensive operation for finding the inverse because it needs a series of complicated multiplication and squaring operations. On the other hand, there are many variants of EEA reported in the available literature [2, 71]. For example, [71] proposed a Montgomery modular inversion algorithm which is a variant of the EEA. However, the algorithm proposed in [71] requires $3m + 2$ clock cycles to complete the modular inversion. In this dissertation, an efficient inversion algorithm has been used based on the binary method, which is well known as the binary inversion algorithm, depicted in Section 9.4, Algorithm 9.1 [2]. Based on this algorithm, the modular inversion is accomplished using a series of additions, subtractions, and shift operations [84]. The detailed architecture is explained in Section 9.4, Fig. 9.1. The result of a modular inversion is achieved after $2m$ iterations.

2.6.4 Complexity Analysis of Modular Arithmetic over \mathbb{F}_p

Modular addition, multiplication, and inversion operations take different number of clock cycles. Their complexity analysis in terms of clock cycles is reported in Table 2.2. As one can see from Table 2.2, addition takes only one clock cycle, whereas inversion takes a huge number of clock cycles. In addition, three different types of modular multipliers are implemented, where the radix-4 modular multiplier provides the best performance.

Table 2.2: *Complexity of $GF(p)$ FFMA operations in terms of clock cycles*

$GF(p)$ operation	Complexity of FFMA operations (in terms of latency (clock cycles))
Addition/subtraction	1
Combined addition and subtraction	2
Montgomery multiplier	$m + 1$
Modified interleaved multiplier	$m + 1$
Radix-4 multiplier	$m/2 + 1$
Inversion (bit-serial)	$2m + 1$

2.6.5 NIST Primes p over \mathbb{F}_p

There are numerous elliptic curve domain parameters over \mathbb{F}_p including prime field moduli bit sizes and their corresponding numerical values, which are required to implement ECP over prime fields $GF(p)$. The prime field for random curves recommended by NIST in the FIPS 186-2 standard for 192, 224, 256, 384 and 521 bits are presented in Table 2.3. Certicom has provided elliptic curve domain parameters for both random and Koblitz curves, where Koblitz curve parameters are depicted in Table 2.4 [54]. The fast-reduction modulo algorithms for the NIST random curves can be found in [2]. In this dissertation, the proposed ECP over $GF(p)$ supports three prime fields of the five NIST-recommended primes p , with sizes 192, 224, and 256 bits.

Table 2.3: *NIST recommended primes over \mathbb{F}_p [22, 54]*

Prime	Size (bits) m	Numerical Value
p192	192	$2^{192} - 2^{64} - 1$
p224	224	$2^{224} - 2^{96} + 1$
p256	256	$2^{256} - 2^{224} + 2^{192} + 2^{224} - 1$
p384	384	$2^{384} - 2^{128} - 2^{96} + 2^{32} - 1$
p521	521	$2^{521} - 1$

Table 2.4: *Prime p for Koblitz curves over \mathbb{F}_p [54]*

Prime	Size (bits) m	Numerical Value
p192	192	$2^{192} - 2^{32} - 2^{12} - 2^8 - 2^7 - 2^6 - 2^3 - 1$
p224	224	$2^{224} - 2^{32} - 2^{12} - 2^{11} - 2^9 - 2^7 - 2^4 - 2 - 1$
p256	256	$2^{256} - 2^{32} - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 - 1$

2.7 Elliptic Curve Cryptography

The theory of elliptic curves is a deep branch of mathematics, and has been extensively studied since the second half of the 20th century, initially, without any cryptographic applications. In the mid-80s, the use of elliptic curves in PKC, named elliptic curve cryptography (ECC), was first proposed independently by Koblitz [11] and Miller [10]. ECC has attractive features, such as that it provides equivalent security to RSA or discrete logarithm systems (e.g. DH) with considerably shorter key lengths (approximately 160-256 vs 1024-3072 bits) [2, 17, 85]. The following section presents a brief introduction to elliptic curves and elliptic curve point arithmetic, e.g. point doubling (PD) and point addition (PA), which are required to implement ECPM.

2.7.1 Overview of Elliptic Curves

An elliptic curve E over a field K , which is denoted E/K , is defined by the long form of the Weierstrass equation

$$E : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6 \quad (2.8)$$

where the coefficients a_1, a_2, a_3, a_4 and a_6 belong to the field K and the discriminant $\Delta \neq 0$ which guarantees that there are no points at which the elliptic curve has more than one tangent i.e. the elliptic curve is “smooth”. The discriminant Δ over elliptic curves E must be $-d_2^2d_8 - 8d_4^3 - 27d_6^2 + 9d_2d_4d_6$, where $d_2 = a_1^2 + 4a_2$, $d_4 = 2a_4 + a_1a_3$, $d_6 = a_3^2 + 4a_6$, and $d_8 = a_1^2a_6 + 4a_2a_6 - a_1a_3a_4 + a_2a_3^2 - a_4^2$.

Now, if x and y are a pair of elliptic curve points, which is denoted by (x, y) , and L is any extension field of K , let the set of L -rational points on E be presented as

$$E(L) = \{(x, y) \in L \times L : y^2 + a_1xy + a_3y - x^3 - a_2x^2 - a_4x - a_6 = 0\} \cup \{\infty\} \quad (2.9)$$

where ∞ is the point at infinity. Equation (2.8) needs to be simplified by changing the variables underlying the field K , because the short form of the Weierstrass equation can be used for practical implementations of an elliptic curve cryptosystem. To get the short form of the Weierstrass equation, there are two conditions: 1) if the underlying field K has characteristic = 2 or 3 and 2) if the underlying field K has the characteristic $\neq 2$ and $\neq 3$.

Based on the first condition, when the underlying field K has characteristic 2 and $a_1 = 0$, the elliptic curve point

$$(x, y) \rightarrow (a_1^2x + \frac{a_3}{a_1}, a_1^3y + \frac{a_1a_4 + a_3^2}{a_1^3}) \quad (2.10)$$

then the elliptic curve E becomes

$$y^2 + xy = x^3 + ax^2 + b \quad (2.11)$$

where $a, b \in K$ and the elliptic curve is called the non-supersingular elliptic curve and $\Delta = b$. Equation (2.11) is called the elliptic curve E over the binary field $GF(2^m)$.

Based on the second condition, the elliptic curve point becomes

$$(x, y) \rightarrow \left(\frac{x - 3a_1^2 - 12a_2}{36}, \frac{y - 3a_1x}{216} - \frac{a_1^3 + 4a_2a_1 - 12a_3}{24} \right) \quad (2.12)$$

and the elliptic curve E becomes

$$y^2 = x^3 + ax + b \quad (2.13)$$

where a and b are the two coefficients belonging to the field K and the discriminant $\Delta = -16(4a^3 + 27b^2)$. Equation (2.13) is called the elliptic curve over $GF(p)$.

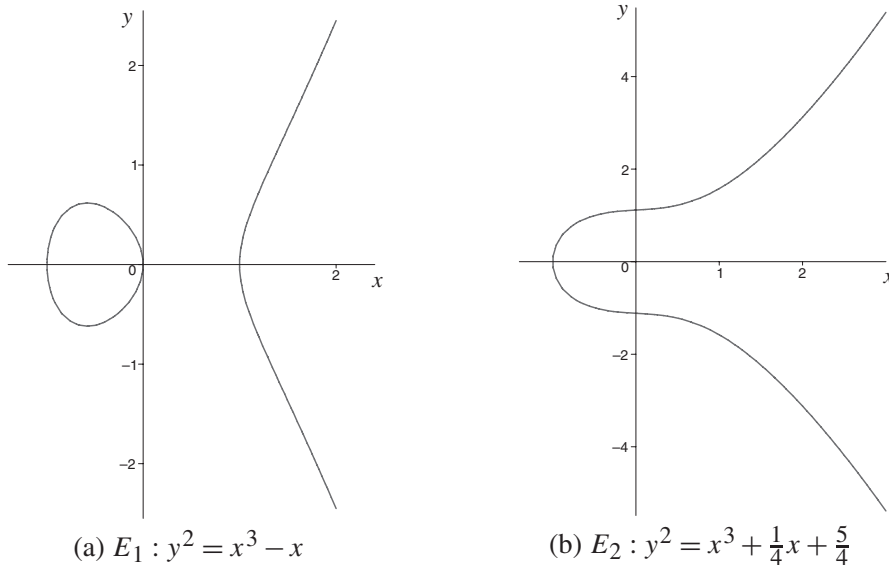


Figure 2.3: *Elliptic curves over real number \mathbb{R} .*

The elliptic curve defined over \mathbb{R} , the field of real numbers, is depicted in Fig. 2.3. However, an elliptic curve defined over a finite field is required for practical cryptographic applications. There have been some other equations based on the first condition. However, in this dissertation the elliptic curve equations (2.11) and (2.13) are used to implement ECP over the binary field and the prime field, respectively.

2.7.2 Elliptic Curve Point Arithmetic

The second level or mid level in the hierarchy of an elliptic curve cryptosystem consists of elliptic curve point arithmetic (or group operations), point doubling (PD) and point addition (PA). These group operations can be performed by the chord-and-tangent rule [2] for doubling or adding two points to get the third point. The PA and PD on elliptic curves in affine coordinates are displayed geometrically in Fig. 2.4. Given two distinct points $P = (x_1, y_1)$ and $Q = (x_2, y_2)$, then the PA of P and Q is the point R ($R = P + Q$) which is the line that crosses the points P and Q on the elliptic curve E , also the point R is the reflection of the point about the x -axis, as shown in Fig. 2.4(a). Similarly, the PD is the results of adding a point P itself which can be defined as $R = 2P$. First draw the tangent line to the elliptic curve at P then the projection over the x axis of the point is defined as R , as shown in Fig. 2.4(b). The PD and PA can be computed both in the binary field $GF(2^m)$ and the prime field $GF(p)$; this dissertation emphasizes both.

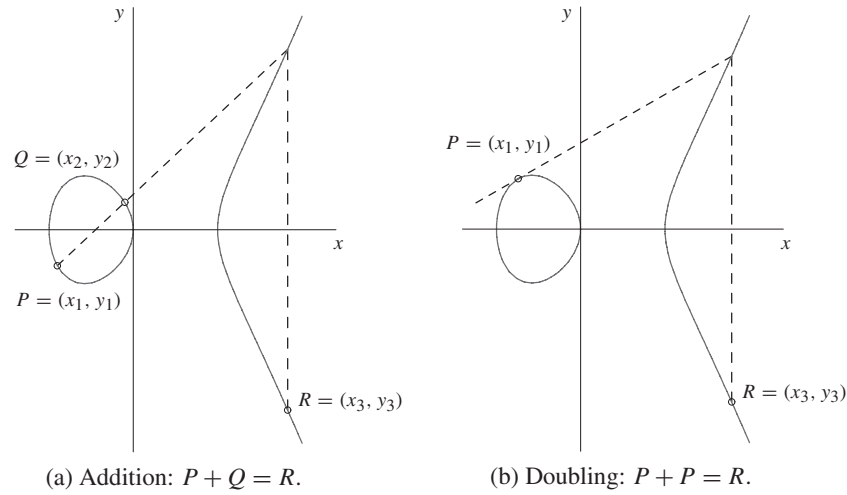


Figure 2.4: *Geometric point addition and point doubling on elliptic curve.*

Point Arithmetic over \mathbb{F}_2^m in Affine Coordinates: $y^2 + xy = x^3 + ax^2 + b$

Point doubling over \mathbb{F}_2^m : Let $P = (x_1, y_1) \in E(\mathbb{F}_2^m)$ in affine coordinates, then the PD in affine coordinates $R = 2P = (x_3, y_3)$ can be computed over $GF(2^m)$ as

$$\left. \begin{array}{l} x_3 = \lambda^2 + \lambda + a, \\ y_3 = x_1^2 + \lambda x_3 + x_3, \\ \text{where } \lambda = x_1 + y_1/x_1. \end{array} \right\} \quad (2.14)$$

Point addition over \mathbb{F}_2^m : Given two points $P = (x_1, y_1)$ and $Q = (x_2, y_2)$ over the binary field, where $P \neq \pm Q$ then the PA in affine coordinates $R = P + Q = (x_3, y_3)$ can be obtained over $GF(2^m)$ as follows:

$$\left. \begin{array}{l} x_3 = \lambda_1^2 + \lambda_1 + x_1 + x_2 + a, \\ y_3 = \lambda_1(x_1 + x_3) + x_3 + y_1, \\ \text{where } \lambda_1 = (y_2 + y_1)/(x_2 + x_1). \end{array} \right\} \quad (2.15)$$

The implementation costs of PD and PA over $GF(2^m)$ in affine coordinates are $1I + 2M + 1S$ and $1I + 2M + 2S$, respectively, where I , M , and S , are the costs of polynomial inversion, multiplication, and squaring, respectively.

Point Arithmetic over \mathbb{F}_p in Affine Coordinates: $y^2 = x^3 + ax + b$, characteristic $(K) \neq 2, 3$

Point doubling over \mathbb{F}_p : Given a point $P = (x_1, y_1)$ over the prime field, then PD in affine coordinates $R = 2P = (x_3, y_3)$ can be obtained over $GF(p)$ as follows:

$$\left. \begin{array}{l} x_3 = \lambda^2 - 2x_1 \\ y_3 = \lambda(x_1 - x_3) - y_1 \\ \text{where } \lambda = (3x_1^2 + a)/2y_1. \end{array} \right\} \quad (2.16)$$

Point addition over \mathbb{F}_p : Given two points $P = (x_1, y_1)$ and $Q = (x_2, y_2)$ over $GF(p)$, where $P \neq \pm Q$ then the PA in affine coordinates $R = P + Q = (x_3, y_3)$ can be computed over $GF(p)$ as follows:

$$\left. \begin{aligned} x_3 &= \lambda^2 - x_1 - x_2, \\ y_3 &= \lambda(x_1 - x_3) - y_1, \\ \text{where } \lambda &= (y_2 - y_1)/(x_2 - x_1). \end{aligned} \right\} \quad (2.17)$$

The implementations of PD and PA in affine coordinates have costs of $1I + 2M + 2S$ and $1I + 2M + 1S$, respectively, where I , M , and S are the costs of modular inversion, multiplication, and squaring, respectively.

In equations (2.14), (2.15), (2.16), and (2.17), some special cases must be considered. For example, the PD and PA can be computed by applying the identity element, called the point at infinity ∞ [2, 17, 85, 86]. For example, if the point $P = (x_1, y_1)$ and its negative identity $-P = (x_1, -y_1)$, then the PD must be $R = 2P = P + (-P) = (x_1, y_1) + (x_1, -y_1) = \infty$, or if $P = (x_1, y_1)$ and $Q = (x_1, y_1)$, then PA is given by: $R = P + Q = (x_1, y_1) + (x_1, -y_1) = \infty$, where $Q = (x_1, y_1)$ is the negative of P . Similarly, $P + \infty = \infty + P = P$ for all $P \in E(K)$ and $-\infty = \infty$. In this dissertation, the equations (2.14), (2.15), (2.16), and (2.17) are used to implement PD and PA operations in affine coordinates.

2.8 Point Representation

Various coordinates can be used to represent elliptic curve points, such as affine and projective representations. A point on the elliptic curve $E(\mathbb{F}_p \text{ or } \mathbb{F}_2^m)$ for affine coordinates can be presented by using two elements $x, y \in \mathbb{F}_p \text{ or } \mathbb{F}_2^m$, i.e. $P(x, y)$. In the previous section, the elliptic curve points have been presented in affine coordinates, which requires a modular/finite field inversion, the most expensive and time-consuming

operation. The inversion for each group operation can be removed by using projective coordinate systems. For this reason, the projective coordinate system is called inversion-free coordinates. However, PD and PA operations in projective coordinates need more modular/finite field multiplications. In projective coordinates, a point P on the elliptic curve needs three elements $X, Y, Z \in \mathbb{F}_p$ or \mathbb{F}_2^m , i.e. $P(X, Y, Z)$. In the literature, several projective coordinates have been proposed to represent elliptic curve points. In the binary field \mathbb{F}_2^m , the projective coordinates are 1) standard, 2) Jacobian, and 3) Lopez-Dahab projective coordinates. In the prime field \mathbb{F}_p , the available projective coordinates are: 1) standard projective coordinates, 2) Jacobian projective coordinates, and 3) Chudnovsky coordinates. In this dissertation, Jacobian projective coordinates have been used to implement the PD and PA operations because it can be used for both fields. Further details of projective coordinates can be found in [2].

2.8.1 Point doubling and point addition over \mathbb{F}_2^m in Jacobian Projective Coordinates

Let the affine point $P = (x, y)$, then the projective coordinates $P = (X, Y, Z)$ are given by (2.18), where Z is simply set to 1 [87].

$$X = x; Y = y; Z = 1. \quad (2.18)$$

The affine point $P = P(x, y)$ can be converted back from the Jacobian projective point $P = (X, Y, Z)$, as is presented in (2.19)

$$x = X/Z^2; y = Y/Z^3. \quad (2.19)$$

Using (2.11), (2.18), and (2.19), the Jacobian projective form of the Weierstrass equation of the elliptic curve becomes

$$Y^2 + XYZ = X^3 + aX^2Z^2 + bZ^6 \quad (2.20)$$

where the point at infinity is $(1, 1, 0)$.

Point doubling over \mathbb{F}_2^m : Let $P = (X_1, Y_1, Z_1)$ be a projective point on the elliptic curve, then the PD operation $R = 2P = (X_3, Y_3, Z_3)$ over $GF(2^m)$ in Jacobian projective coordinates can be computed as [87]:

$$\left. \begin{aligned} Z_3 &= X_1 Z_1^2, \\ X_3 &= (X_1^4 + b Z_1^8) \\ Y_3 &= X_1^4 Z_3 + (X_1^2 + Y_1 Z_1 + Z_3) X_3. \end{aligned} \right\} \quad (2.21)$$

Point addition over \mathbb{F}_2^m : Given two projective points $P = (X_1, Y_1, Z_1)$ and $Q = (X_2, Y_2, Z_2)$ on the elliptic curve, then the PA operation $R = P + Q = (X_3, Y_3, Z_3)$ over $GF(2^m)$ in Jacobian projective coordinates can be computed as [87]:

$$\left. \begin{aligned} Z_3 &= Z_1 Z_2 W, \\ X_3 &= a Z_3^2 + R(R + Z_3) + W^3, \\ Y_3 &= (R + Z_3) X_3 + Z_1^2 W^2 (R X_2 + Y_2 Z_1 W), \end{aligned} \right\} \quad (2.22)$$

where $W = (X_1 Z_2^2 + X_2 Z_1^2)$ and $R = Y_1 Z_2^3 + Y_2 Z_1^3$.

The implementation costs of PD and PA over $GF(2^m)$ in Jacobian projective coordinates based on equations (2.21) and (2.22) are $5M + 5S$ and $14M + 4S$, respectively, where M is the cost of polynomial multiplication and S is the cost of polynomial squaring.

2.8.2 Point doubling and point addition over \mathbb{F}_p in Jacobian Projective Coordinates

The Jacobian projective form of the Weierstrass equation for an elliptic curve over the prime field \mathbb{F}_p can be written as (2.23), where $P = (X, Y, Z)$ is the point of projective coordinates [2, 88, 89] :

$$Y^2 = X^3 + aXZ^4 + bZ^6. \quad (2.23)$$

Point doubling over \mathbb{F}_p : Given a point $P = (X_1, Y_1, Z_1)$ on the elliptic curve, then the PD over $GF(p)$ in Jacobian projective coordinates $R = 2P = (X_3, Y_3, Z_3)$ can be computed as follows [89]:

$$\left. \begin{aligned} X_3 &= (3X_1^2 + aZ_1^4)^2 - 8X_1Y_1^2, \\ Y_3 &= (3X_1^2 + aZ_1^4)(4X_1Y_1^2 - X_3) - 8Y_1^4, \\ Z_3 &= 2Y_1Z_1. \end{aligned} \right\} \quad (2.24)$$

Equation (2.24) represents the PD operation for a random curve. The PD operation for the Koblitz curve can be computed as (2.25), where the coefficient $a = 0$.

$$\left. \begin{aligned} X_3 &= (3X_1^2)^2 - 8X_1Y_1^2, \\ Y_3 &= 3X_1^2(4X_1Y_1^2 - X_3) - 8Y_1^4, \\ Z_3 &= 2Y_1Z_1. \end{aligned} \right\} \quad (2.25)$$

Point addition over \mathbb{F}_p : Given two points $P = (X_1, Y_1, Z_1)$ and $Q = (X_2, Y_2, Z_2)$ on the elliptic curve, then the PA over $GF(p)$ in Jacobian coordinates $R = P + Q = (X_3, Y_3, Z_3)$ can be computed as follows [89]:

$$\left. \begin{aligned} X_3 &= A^2 - B^3 - 2X_1Z_2^2B^2, \\ Y_3 &= A(X_1Z_2^2B^2 - X_3) - Y_1Z_2^3B^3, \\ Z_3 &= Z_1Z_2B^0 \end{aligned} \right\} \quad (2.26)$$

where $A = Y_2Z_1^3 - Y_1Z_2^3$ and $B = X_2Z_1^2 - X_1Z_2^2$.

The implementations of PD and PA over $GF(p)$ in Jacobian projective coordinates have costs of $4M + 4S$ and $12M + 4S$, respectively, where M and S are the costs of modular multiplication and squaring, respectively.

Most of the ECPM implementations in the literature have used separate PD and PA operations, and require more latency than the proposed combined PDPA. In this dissertation, a novel combined hardware is introduced to compute PD and PA together

with lower latency. For example, the equations (2.21) and (2.22) are used to implement a combined PD and PA, called PDPA, over the binary field $GF(2^m)$. Similarly, equations (2.24) or (2.25) and (2.26) are used to implement a combined PD and PA over the prime field $GF(p)$. The details of the combined PDPA hardware over $GF(2^m)$ and $GF(p)$ are discussed in Section 7.5 and Section 11.5, respectively.

2.9 Elliptic Curve Point Multiplication

Elliptic curve point multiplication (ECPM), also known as elliptic curve scalar multiplication (ECSM), is the highest level in the hierarchy of ECC operations and it dominates the computation of cryptographic schemes, such as the elliptic curve digital signature algorithm (ECDSA). ECPM is the core operation of an ECC processor (ECP), and is computationally the most expensive operation, hence to improve the performance of ECPM is a challenging task. The basic point multiplication can be defined as:

$$Q = kP = \underbrace{P + P + \dots + P}_{k \text{ times}} \quad (2.27)$$

where k is an integer (which is the private/secret key) in the range $1 \leq k < \text{order}(P)$, and P and Q are two points on the elliptic curve defined over a field $GF(q)$ [2, 17, 85]. As shown in (2.27), the point multiplication kP represents the addition $k - 1$ times of elliptic curve point P . When the point P on the elliptic curve is fixed, for example in ECDSA signature generation, ECPM can be computed very efficiently because certain fixed parameters can be exploited from pre-computed data. The performance of ECPM relies on finding the minimum number of steps to compute kP from a given elliptic curve point P . For doing this, there are various methods/algorithms to compute ECPM, such as the double-and-add (or binary) method, window methods, the Non-adjacent form (NAF), and the Montgomery method. In the following section, the most used algorithm known as the double-and-add algorithm is discussed.

Algorithm 2.1: Double-and-add (left to right) point multiplication algorithm

Input: $k = (k_{m-1}, \dots, k_1, k_0)_2$, $P \in E(\mathbb{F}_2^m \text{ or } \mathbb{F}_p)$

Output: $Q = k \cdot P$, where $Q \in E(\mathbb{F}_2^m \text{ or } \mathbb{F}_p)$

1. $Q = 0$;
 2. **for** $i = m - 1$ **to** 0 **do** $Q = 2Q$;
 - 2.1 **if** $k(i) = '1'$ **then** $Q = Q + P$; **end if**
 - 2.2 **end for**
 3. **Return** Q
-

2.9.1 Double-and-add point multiplication

The double-and-add (or binary) algorithm is used in this dissertation, and is one of the simplest methods to compute point multiplication. Algorithm 2.1 depicts the double-and-add method [2]. This algorithm can be used to implement point multiplication over either binary fields or prime fields, also either in affine or Jacobian coordinates. As can be seen from Algorithm 2.1, it iterates through each bit of k , where the scalar $k = \sum_{i=0}^{m-1} k_i 2^i$. In this method, a point P is added $k - 1$ times to itself to get the final point Q in affine or Jacobian projective coordinates. ECPM can be computed using execution sequences of PD and PA operations [2], i.e. it can be computed as multiple points on an elliptic curve with the use of the repeated doubling and addition of points on an elliptic curve. The execution schedules are directly related to the bit pattern of the scalar multiplier, $k = \text{'key'}$. Generally, a PD operation performs on every iteration, and a PA operation only performs when the particular bit of k is one. In this method, $m - 1$ iterations are required to compute the point multiplication, where the latency of each iteration depends on the latency of the PD and PA operations. This algorithm requires on average m PDs and $m/2$ PAs, where $m \approx \log_2 m$.

2.9.2 Example of double-and-add point multiplication

An example of the binary point multiplication algorithm is given below [17]:

Example: $Q = 27P = (11011_2)P = (d_4d_3d_2d_1d_0)P$

Step

#0	$Q = 0$	initial setting
#1a	$Q = 2Q = 0_2P$	
#1b	$Q = Q + P = 0 + P = 1_2P$	Add (bit $d_4 = 1$)
#2a	$Q = 2Q = 2(1P) = 2P = 10_2P$	Double (bit d_3)
#2b	$Q = Q + P = 2P + P = 3P = 11_2P$	Add (bit d_3)
#3a	$Q = 2Q = 2(3P) = 6P = 110_2P$	Double (bit d_2)
#3b	$Q = Q = 6P$	No add (bit $d_2 = 0$)
#4a	$Q = 2Q = 2(6P) = 12P = 1100_2P$	Double (bit d_1)
#4b	$Q = Q + P = 12P + P = 13P = 1101_2P$	Add (d_1)
#5a	$Q = 2Q = 2(13P) = 26P = 11010_2P$	Double (bit d_0)
#5b	$Q = Q + P = 26P + P = 27P = 11011_2P$	Add (d_0)

2.10 Security Analysis

The hardness or strength of elliptic curve cryptosystems based on ECPM or ECSM is equal to kP , where it is very hard problem to recover k . This is the so-called ECDLP, which is a harder mathematical problem than integer factorisation (e.g. RSA) and the discrete logarithm problem (e.g. DH and ElGamal). Also, based on the literature survey it is shown that ECDLP is considered to have fully exponential timing complexity, i.e. it takes an exponential time to recover or solve for k (or *key*) [85]. Therefore, the entire security of different protocols, e.g. elliptic curve Diffie-Hellman (ECDH) key exchange

and ECDSA, is based on the hardness of the ECDLP. Sometimes the double-and-add algorithm for computing ECPM is not safe against many side-channel attacks (SCA), e.g. simple power analysis (SPA) and differential power analysis (DPA), and an intruder can get the ‘*key*’ by tracing the power consumption for the PD and PA operations in each iteration. However, if the PD and PA operations have equal cost then it is difficult to extract the ‘*key*’. In this dissertation, a combined PDPA hardware is designed for both binary fields and prime fields, and computes the PD and PA operations concurrently, as explained in Section 7.5 and 11.5. Therefore, the power consumption pattern for the PDPA hardware will be symmetric in nature. As explained in Figs. 7.7 and 11.5, a scalar multiplication hardware is developed using the combined PDPA hardware architecture. Hence, a uniform power consumption profile may be measured throughout the point-multiplication computation. From the hardware analysis, it can be observed that any ‘*key*’ leak information is difficult to extract. Besides, the double-and-add algorithm is secure against timing and SPA attacks [90]. A detailed security analysis can be found in [91–96].

2.11 Platforms for Hardware Implementation

There are various platforms for hardware implementation of cryptosystems currently available. However, two well-known hardware implementation platforms are often used: 1) field-programmable gate array (FPGA) and 2) application-specific integrated circuit (ASIC). Both platforms have their own advantages and disadvantages.

An FPGA is a reconfigurable device which is composed of lots of configurable logic blocks (CLBs), including slices and registers, input/output blocks (IOBs), and programmable interconnects. The FPGA has many features as well as multiple advantages. It is widely used for hardware implementation as a prototype design. In addition, an FPGA has a

high level of flexibility or reprogrammability which means that a cryptographic algorithm (e.g. elliptic curve cryptosystem) can easily be updated. Moreover, FPGAs assure lower cost for prototype design or in small volumes since they do not incur any fabrication cost. Furthermore, an FPGA provides better performance in a shorter design time. For FPGA implementations, a bitstream (e.g. .bit or .bin extension) can be generated after synthesis, mapping, place and route phases, then the bitstream can be loaded into an FPGA for execution. In this dissertation, Xilinx Virtex-7 and Kintex-7 FPGAs are used to implement elliptic curve cryptosystems over either binary fields or prime fields. Also, Xilinx Virtex-6 and Virtex-5 FPGAs are used to synthesise the designs.

An ASIC provides better performance than an FPGA because the ASIC is optimised for specific applications while FPGAs are optimised for generic implementations. For bulk production or in high volumes, ASICs, after the first run, are much cheaper than the corresponding production based on FPGA devices. However in prototype or in small volumes, an ASIC is very expensive due to startup cost, e.g. costs of masks required in manufacturing. Also, ASIC-based implementations may take a huge time to fabricate. Apart from that, ASIC-based implementations are required where the performance is of utmost importance. For example, it is mandatory for faster and low-power customised applications.

These hardware implementation platforms can be programmed by two hardware description languages (HDLs): 1) Very high speed integrated circuit (VHSIC) HDL or simply named VHDL and 2) Verilog. In this dissertation, all the hardware architectures are implemented using VHDL. Both FPGA and ASIC technologies are used to implement ECPs.

2.12 Elliptic Curve Digital Signature Algorithm

The elliptic curve digital signature algorithm (ECDSA) is the application of ECC to the digital signature algorithm (DSA), and it is the most widely used elliptic curve based signature scheme [2, 22–24, 27, 97]. The ECDSA protocol is made up of three main components: 1) key generation, 2) signature generation, and 3) signature verification. The ECDSA has been standardised by international standards bodies and appears in ANSI X9.62 [24], NIST FIPS 186-2 [22], IEEE 1363-2000 [23], and ISO/IEC 15946-2 [2]. The detailed algorithms of elliptic curve digital signature generation and verification are described in Section 11.6.5, Algorithm 11.5 and Algorithm 11.6. As one can see from the algorithms, the execution sequence of signature generation and signature verification rely on the modular arithmetic operations and point multiplication. In this dissertation, all the modular arithmetics and point multiplications are implemented which can be used for the ECDSA.

2.13 Summary of Proposed Designs

The main focus of this dissertation is to implement high-performance (which includes high speed, low area, and low energy dissipation) ECC processors (ECPs) both in the binary field $GF(2^m)$ and the prime field $GF(p)$. To achieve this, a finite field arithmetic (FFA) unit is presented for binary-field ECPs. The FFA unit consists of finite field or polynomial addition, multiplication, squaring, and inversion. In this dissertation, polynomial multiplication is designed in both bit-serial and digit-serial approaches. The proposed FFA unit is implemented on both FPGA and ASIC platforms and their performances in terms of timing, area, energy, AT, and ATE are compared with the existing literature. Based on the performance comparisons of different FFA designs in the literature, the proposed FFA design performs better. In addition, a separate PD and PA (group oper-

ations) hardware and a combined hardware are designed for ECPs over $GF(2^m)$. Using this high-performance FFA unit and group operations, efficient ECPs over the binary field are achieved. Similarly, a finite field modular arithmetic (FFMA) unit is presented for the prime-field ECPs. The FFMA consists of modular addition, subtraction, combined addition and subtraction, multiplication, squaring, and inversion. The modular multiplication is proposed and implemented in two different approaches: 1) modified interleaved method and 2) radix-4 method. Also, a separate PD and PA hardware is designed for an ECP over $GF(p)$ in affine coordinates. Moreover, an optimised combined PDPA hardware architecture is developed for an ECP over $GF(p)$ in Jacobian projective coordinates. Finally, both FPGA- and ASIC-based ECPs over $GF(p)$ are implemented using the designed FFMA unit and group operations. Numerous implementations of FFA, FFMA, ECPM are presented for ECC in the literature, however, it is not easy to say which designs are the best because the algorithms, platforms, and elliptic curve parameter sizes are not always the same. Therefore, the overall performance for both binary-field ECPs and prime-field ECPs is compared with the most relevant implementations in the literature. The detailed performance comparisons of all designs are discussed in each chapter.

Chapter 3

Efficient Hardware Implementation of Finite Field Arithmetic for Elliptic Curve Cryptography¹

3.1 Abstract

For practical use of an elliptic curve cryptography (ECC) processor, an efficient hardware implementation must be achieved in terms of time, area, and power. For a high-performance ECC processor, a large number of finite field additions, multiplications, and inversions are required. In this paper, we propose a high-performance hardware implementation of finite field arithmetic (FFA) for an ECC processor over NIST binary fields. This paper presents a bit-serial and digit-serial architecture using a polynomial basis to compute a finite field multiplication. The latency for the bit-serial and digit-

¹Submitted as: M. S. Hossain, K. Ruangsantikorakul and Y. Kong, "Efficient Hardware Implementation of Finite Field Arithmetic for Elliptic Curve Cryptography," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, in review.

*serial multipliers is m and $\lceil m/d \rceil$, respectively. In addition, an improved version of systolic digit-serial multiplication over $GF(2^m)$ is proposed whose time complexity can be scaled down to $2 * \lceil \sqrt{m/d} \rceil$ clock cycles for the digit-size d . Moreover, an inversion architecture over $GF(2^m)$ is proposed, which involves a very low area complexity in either a field-programmable gate array (FPGA) or an application-specific integrated circuit (ASIC). Based on the overall performance analysis of different implementations, it is shown that the proposed FFA is more area- and energy-efficient than all comparable work in the literature. To the best of the authors' knowledge, the FPGA-based designs provide better efficiency ($1/AT$) and the ASIC-based designs deliver the best $\text{Area} \times \text{Time} \times \text{Energy}$ (ATE) performance.*

3.2 Introduction

Elliptic curve cryptography (ECC), is currently the leading public-key cryptographic technique in terms of security, speed, area, and power consumption rather than the commonly used cryptosystem RSA [8]. It was first proposed in the mid-80s by Koblitz and Miller [1, 10]. The National Institute of Standards and Technology (NIST) recommends elliptic curves (ECs) over binary fields for the digital signature standard (DSS) [22]. IEEE P1363-2000 [23] also has standardized public-key cryptographic techniques, including cryptographic schemes, using the ECC-based key agreement and digital signature algorithm (DSA). Certicom has provided NIST-recommended EC domain parameters, standard for efficient cryptography in SEC2 [54]. Various security protocols like identity-based cryptography [98] and the short signature scheme [99], cryptographic pairing has been used extensively. The Weil and Tate pairings protocols based on elliptic curve arithmetic need huge numbers of multiplications and additions over large finite fields, and

have gained great attention in this area [100, 101]. Finite field arithmetic (FFA) consists of addition, multiplication, squaring, and inversion and is mandatory for all kinds of ECC implementations over the binary field $\text{GF}(2^m)$. It is also widely used in error control coding [43, 102, 103]. The practical realization of pairing-based cryptography in resource-constrained environments is very demanding and challenging, hence a trade-off between time, area, and power is necessary to get the best performance of FFA.

3.2.1 Related Work

Over the decades, numerous hardware implementations of finite field arithmetic for cryptosystems (e.g. ECC processor) have been presented in the literature [2, 35, 43–45, 49, 50, 52, 104–121]. For the practical applications of a cryptosystem, various types of inversion and multiplication algorithms are proposed. For example, for a high-speed computation, a bit-parallel field arithmetic is mandatory, but they need high power and area complexities, whereas a bit-serial inversion/multiplication is better for area-efficient and low-power consumption design; but they are not fast as the bit-parallel or digit-serial/digit-parallel approaches. Generally, digit-serial multiplication is better for low area \times time (AT) complexity and systolic multipliers over $\text{GF}(2^m)$ are better with high-throughput applications [43, 104, 105, 122, 123]. Note that m clock cycles latency is required for the bit-serial multipliers, whereas basic digit-serial multipliers have the very low latency of $\lceil m/d \rceil$ with a slightly higher area complexity, where m is the field size and d is the digit size. In this paper, we have implemented both types of finite field multipliers to compare the performance. Most of the paper discusses only multiplication or inversion, whereas we have implemented and discussed all finite field arithmetics in detail. The previous work in the available literature is classified into two categories: 1) field-programmable gate array (FPGA)-based implementations and 2) application-specific integrated circuit (ASIC)-based implementations. In the first part, FPGA-based implementations of multiplication

and inversion over $\text{GF}(2^m)$ are discussed, and the second part discusses ASIC-based implementations of FFA.

Multiplication over $\text{GF}(2^m)$ using a polynomial basis was proposed in [108]. They implemented, for all five NIST binary fields in a Xilinx Virtex-6 FPGA. In [109] and [118], FPGA-based 163-bit finite field multiplication using a bit-serial approach was proposed for an ECC processor. Both of their designs have higher area complexity than similar work. There are various implementations of digit-serial multipliers available in the literature, however very few provide detailed implementation results. FPGA-based digit-serial multipliers were proposed in [35, 115, 119] with high latency. In addition, we have also implemented modified digit-serial multipliers with lower latency (e.g. $2 * \lceil \sqrt{m/d} \rceil$). Digit-serial finite field inversion was proposed in [107] and [35]. [124] proposed a modified Itoh-Tsujii algorithm and implemented using a Xilinx Virtex-4 FPGA, and [35] implemented inversion in a Virtex-5 FPGA using a polynomial basis. We have observed that digit-serial inversion usually takes a huge area (or higher area complexity) to implement. Inversion in $\text{GF}(2^m)$ using a bit-serial approach was implemented on Virtex-2 in [118] and Virtex-4 in [115]. However, their designs take a high computation time as well as high area complexity.

In [114] a combined finite field multiplication and inversion for all NIST fields have been introduced, but the area complexity is very high. This combined circuit was synthesized in the 0.18- μm CMOS standard cell library, and used the modified extended Euclid's algorithm for inversion and the most-significant bit (MSB) first multiplication algorithm for hardware implementations. [107] proposed a digit-serial based normal-basis ternary Itoh-Tsujii inversion algorithm using hybrid-double multipliers. They implemented inversion for four NIST fields of the five NIST-recommended binary fields using TSMC 65-nm CMOS technology. We find that the digit-serial inverter is very efficient with timing due to fewer clock cycles, but needs a huge area to implement, and the AT value is still

higher than the bit-serial inverter. For this reason, a bit-serial inverter over $\text{GF}(2^m)$ is implemented in which the latency (clock cycles) is $2m + 1$. [106] proposed an ASIC-based low-power versatile multiplier over $\text{GF}(2^m)$ using a polynomial basis. They also implemented [108] and [112] using TSMC 65-nm CMOS technology library. A digit level 283-bit finite field multiplier was presented in [113], and implemented using 0.18- μm VLSI technology. However, their proposed design consumes more power. For the trade-off between the timing, area and power complexities, bit-serial and digit-serial multipliers and a bit-serial inverter have been implemented for this manuscript.

3.2.2 Our Contribution

In this paper, we present several efficient hardware implementations of multiplication and inversion in $\text{GF}(2^m)$ (or \mathbb{F}_2^m) which includes high speed, low area, and low power consumption both in FPGA and ASIC. The proposed finite field arithmetic (FFA) supports all five NIST binary fields between 163 and 571 bits. To improve the performance of an ECC processor, a high-performance FFA unit is mandatory. Besides, the low area and time complexities of FFA operations are crucial for an ECC processor. In this research work, a bit-serial finite field multiplication algorithm and corresponding architecture are proposed; the low area and low power complexity hardware apply both in FPGA and ASIC. However, bit-serial multiplications in $\text{GF}(2^m)$ are slow. Therefore, we propose a structurally improved multiplication architecture in $\text{GF}(2^m)$ based on the traditional digit-serial approach whose latency (clock cycles) is $\lceil m/d \rceil$. It is implemented with a low latency and low area complexity. In addition, a systolic digit-serial finite field multiplication is implemented both in FPGA and ASIC with a very low latency ($2 * \lceil \sqrt{m/d} \rceil$ clock cycles only). It is the fastest, high throughput, and low power consumption hardware in ASIC. Moreover, a bit-serial finite field inversion architecture is proposed based on modified Euclid's algorithm, where an inverter is mandatory for the ECC processor in

affine coordinates. Overall, a high-performance FFA unit is proposed which can be used for large binary field ECC processors, either in affine or projective coordinates.

3.2.3 Structure of the Paper

The paper is organized as follows. The preliminary background of ECC over \mathbb{F}_2^m is presented in Section 3.3, where we show why finite field arithmetic (e.g. addition, multiplication, squaring, and inversion) is so important for an ECC processor. In Section 3.4, the algorithms and the description of the hardware architecture for multiplication and inversion over $\text{GF}(2^m)$ are presented. In this section, the proposed hardware implementation of bit-serial and digit-serial multipliers and a bit-serial inverter are explained in detail. Section 3.5 discusses both FPGA and ASIC-based implementations, and comparisons of the most significant work in the literature. Finally, Section 3.6 presents our concluding remarks.

3.3 Preliminaries

An elliptic curve in affine coordinates over the binary field $\text{GF}(2^m)$ is the set of solutions to the equation

$$y^2 + xy = x^3 + ax^2 + b \quad (3.1)$$

where $x, y, a, b \in \text{GF}(2^m)$, $b \neq 0$. The coefficients $a, b \in \mathbb{F}_2^m$ are defined by the NIST [2,22]. ECC is performed in either prime fields $\text{GF}(p)$ or binary fields $\text{GF}(2^m)$. However, the binary field is emphasized in this paper because it is very efficient for hardware implementation due to the use of modulo-2 arithmetic. The elliptic curve group operations (e.g. point doubling (PD) and point addition (PA)) are required to implement an ECC processor. Various coordinate systems exist in the literature to implement group operations such

as affine, projective, Jacobian projective, Lopez-Dahab, and Chudnovsky coordinates; a detailed coordinate system is discussed in [2]. However, affine and Jacobian projective coordinates are often used for hardware implementation of PD and PA. Table 3.1 depicts the computation procedure of PD and PA over the binary field in affine coordinates.

Table 3.1: *Elliptic curve point doubling (PD) and point addition (PA) in affine coordinates over $GF(2^m)$*

PD ($P_3 = 2P_2$)	PA ($P_3 = P_1 + P_2$)
$\lambda = x_1 + y_1/x_1$	$\lambda_1 = (y_2 + y_1)/(x_2 + x_1)$
$x_3 = \lambda^2 + \lambda + a$	$x_3 = \lambda_1^2 + \lambda_1 + x_1 + x_2 + a$
$y_3 = x_1^2 + \lambda x_3 + x_3$	$y_3 = \lambda_1(x_1 + x_3) + x_3 + y_1$

The Jacobian projective form of the Weierstrass equation of the elliptic curve is defined in (3.2), where $x = X/Z^2$ and $y = Y/Z^3$. Table 3.2 presents the elliptic curve group operations (e.g. PD and PA) in Jacobian projective coordinates.

$$Y^2 + XYZ = X^3 + aX^2Z^2 + bZ^6 \quad (3.2)$$

Table 3.2: *Elliptic curve point doubling (PD) and point addition (PA) in Jacobian projective coordinates over $GF(2^m)$*

PD ($P_3 = 2P_2$)	PA ($P_3 = P_1 + P_2$)
$A = X_1^2 + Y_1Z_1$	$W = X_1Z_2^2 + X_2Z_1^2$
$B = A + Z_3$	$R = Y_1Z_2^3 + Y_2Z_1^3, T = RX_2 + Y_2Z_1W$
$X_3 = (X_1^4 + bZ_1^8)$	$X_3 = aZ_3^2 + R(R + Z_3) + W^3$
$Y_3 = X_1^4Z_3 + BX_3$	$Y_3 = (R + Z_3)X_3 + Z_1^2W^2T$
$Z_3 = X_1Z_1^2$	$Z_3 = Z_1Z_2W$

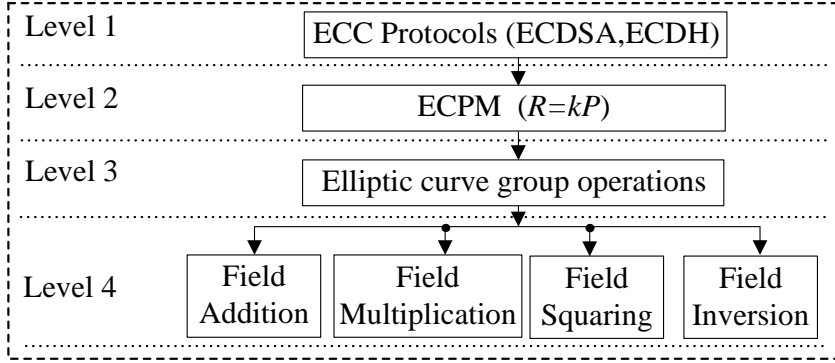


Figure 3.1: *Implementation hierarchy of the ECC operations over $GF(2^m)$.*

The implementation hierarchy of the ECC operations over the binary field is depicted in Fig. 3.1. From this, elliptic curve group operations, e.g. PD and PA, are a series of finite-field arithmetic operations such as field addition, multiplication, squaring, and inversion. The main focuses in this paper is: polynomial-basis modular addition or finite field addition, multiplication, squaring, and field inversion, which are the most crucial for the overall performance of an ECC processor. Generally, the PD operation in affine coordinates requires five finite field additions, two multiplications, two squarings, and one inversion. Similarly, the PA operation in affine coordinates requires eight additions, two multiplications and one squaring, one inversion over $GF(2^m)$. Hence, the overall computation of elliptic curve point multiplication (ECPM) in affine coordinates is slower due to the inversion. On the other hand, the finite field inversion can be avoided for elliptic curve operations in Jacobian projective coordinates. Note that only one inversion is required per ECPM to convert Jacobian to affine coordinates, which is essential to improve the functionality of an ECC processor. Consequently, the computation time for ECPM in Jacobian coordinates is much less than in affine coordinates.

3.4 Hardware for Finite Field Arithmetic

Galois Field (GF) arithmetic, also known as finite field arithmetic (FFA), is crucial in the hardware implementation of an ECC processor, because the overall efficiency relies on the performance of FFA. Three kinds of finite field, such as the prime field \mathbb{F}_p , the binary field \mathbb{F}_2^m , and an optimal extension field \mathbb{F}_p^m , exist for the hardware implementation of ECC. However, modulo-2 (or binary field) arithmetic is the most efficient with hardware implementation, and so a focus on the binary field will be discussed in this research.

3.4.1 Finite Field Addition

Addition in $\text{GF}(2^m)$ or modulo-2 addition is the simplest operation over the binary field. Adding two elements in $\text{GF}(2^m)$ is done using a bit-wise exclusive-or (XOR), as shown in (3) [41]:

$$Z(x) = U(x) + V(x) = \sum_{i=0}^{m-1} u_i x^i + \sum_{i=0}^{m-1} v_i x^i = \sum_{i=0}^{m-1} (u_i + v_i) x^i = \sum_{i=0}^{m-1} z_i x^i \quad (3.3)$$

where $z_i = (u_i + v_i) \bmod 2 = u_i \oplus v_i$. Algorithm 3.1 depicts the hardware implementation of addition for \mathbb{F}_2^m and Figure 3.2 demonstrates the corresponding hardware architecture. As can be seen from Fig. 3.2, it is a carry-free operation and each bit of the output depends upon on the corresponding bits of the input. This operation can be computed in parallel. Therefore, addition over $\text{GF}(2^m)$ takes only one clock cycle. The subtraction operation in $\text{GF}(2^m)$ is the same as addition, because the additive inverse of an element is its identity: $U(x) + U(x) = 0$.

3.4.2 Finite Field Multiplication

Finite field multiplication in a polynomial basis is the second-most expensive operation at the arithmetic level of an ECC processor. Besides, this is the most important arithmetic

Algorithm 3.1: Hardware Implementation of addition over the binary field \mathbb{F}_2^m

Input: $U(x)$ and $V(x) \in E(\mathbb{F}_2^m)$

Output: $Z(x) = U(x) \oplus V(x)$

1. $Z(i) = U(i) \oplus V(i)$;
 2. Return $Z(x)$
-

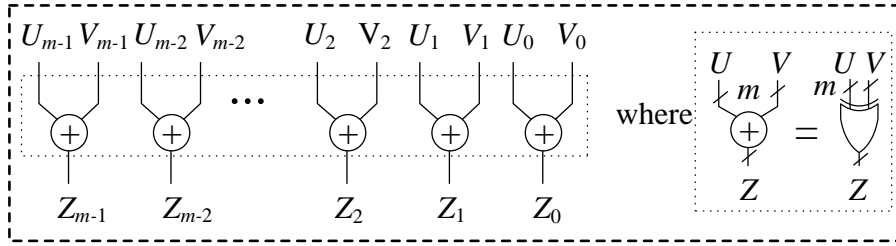


Figure 3.2: Hardware Architecture of addition over $GF(2^m)$.

operation for ECPM, which is the core operation of an ECC processor. A variety of algorithms have been proposed in the literature to perform finite field multiplication including multiplication with an interleaved modular reduction algorithm, bit-serial multiplication, the Karatsuba-Ofman algorithm, higher-radix multipliers, digit-serial multiplication, and digit-parallel multiplication [2, 43–45]. In this paper, bit-serial and digit-serial methods have been utilized. Finite field multiplier computes the product of two polynomials, then applies modular reduction with $f(x)$. Let $U(x)$ and $V(x)$ be two inputs and $Z(x)$ be their output, then

$$Z(x) = U(x).V(x) \text{ mod } f(x), \quad (3.4)$$

where $f(x)$ is a constant irreducible polynomial of degree m .

Bit-serial Multiplication

Bit-serial multiplication in $GF(2^m)$ creates one partial product for each bit of the input during multiplication. Algorithm 3.2 depicts the proposed bit-serial multiplication over

Algorithm 3.2: Bit-serial finite field multiplication.

Input: $U(x), V(x) \in \text{GF}(2^m)$, irreducible polynomial $f(x)$ of degree m

Output: $Z(x) = U(x) \cdot V(x) \bmod f(x)$

1. $Z_v = 0$; $P = f(x)$;
 2. **for** $j = m - 1$ **to** 0 **do**
 - 2.1 $U_v = \text{'0'}$ & $U(x)$; $Z_v = Z_v.x$ (left-shift operation) ;
 - 2.2 **for** $i = 0$ **to** $m - 1$ **do** $U_v(i) = U_v(i)$ and $V(j)$; **end for**
 - 2.3 $Z_v = Z_v \oplus U_v$;
 - 2.4 **for** $l = 0$ **to** m **do** $P_v(l) = P(l)$ and $Z_v(m)$; **end for**
 - 2.5 $Z_v = Z_v \oplus P_v$;
 3. **end for**
 4. **Return** $Z(x)$
-

$\text{GF}(2^m)$. In this algorithm, one partial product is generated and accumulated in each cycle in the product Z . The shift-and-add technique is used for this implementation because a vector shift can be performed in one clock cycle. As Step 2 in Algorithm 3.2, if $V_j = 1$ then $U(x)$ is added to the register Z_v and the product is left-shifted ($Z_v.x$). To obtain a result in $\text{GF}(2^m)$, the product Z_v also undergoes a reduction in each clock cycle. Therefore, a modular reduction is only needed when the result of $Z_v(m) = 1$. This demonstrates that reduction is achieved by checking each bit with a degree of m in the dividend to see whether or not it is a '1' or a '0'. If it is a '1' then the left-most '1' bit of the dividend and the divisor are aligned and undergo bit-wise XOR. This process is repeated until a remainder is determined.

The proposed hardware architecture based on Algorithm 3.2 is depicted in Fig. 3.3. Note that Algorithm 3.2 performs from the most-significant bit (MSB) first to the least-significant bit (LSB), where a multiplication over $\text{GF}(2^m)$ takes only m clock cycles. As

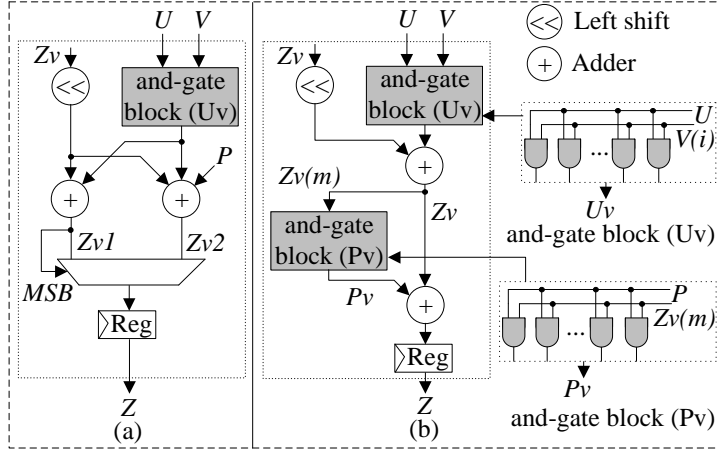


Figure 3.3: Proposed bit-serial finite field multiplication architecture in $GF(2^m)$.

can be seen from Fig. 3.3(a), two field additions are performed concurrently. This method requires one multiplexer which is more expensive than the and-gate (P_v) operation. On the other hand, Fig. 3.3(b) needs only two field additions, one left-shift operation, and two and-gate operations. Besides, the and-gate operation is the faster operation for hardware implementation. Therefore, Fig. 3.3(b) is implemented as a bit-serial multiplier. The area cost of bit-serial field multiplication is only $O(m)$, which is more efficient with the area than a parallel multiplier whose area cost is $O(m^2)$.

Traditional digit-serial Multiplication

Bit-serial finite field multiplication in the previous section has low area complexity, but is slow. A motivation for improving the performance of multiplication over $GF(2^m)$ is for the speed up of an ECC processor. By multiplying 2- bits at a time in digit-serial multiplication, the clock cycles required to perform binary field multiplication will be halved. Numerous techniques have been introduced in the literature [2, 43–45, 110, 111, 116, 117, 119–121] to achieve high-speed field multiplication. The digit-serial multiplier is one of them, and speeds up the multiplication process significantly compared to the bit-serial multiplier, as multiple partial products are generated and added per cycle, thus

greatly reducing the number of clock cycles. However, the area increases with the digit size. For this reason, different digit sizes were implemented so that their performance ($\frac{1}{Area \times Time} = \frac{1}{AT}$) can be compared. In this paper, 1, 2, 4, 8, 16, 32, and 64-bit versions of a digit-serial multiplier over $GF(2^m)$ are implemented to verify the performance. The best multiplier can then be chosen for the later stages of this paper in order to achieve the best area-time performance of the FFA unit.

Algorithm 3.3 depicts the basic (or traditional) digit-serial multiplier over $GF(2^m)$ and the corresponding hardware architecture is demonstrated in Fig. 3.4, where Fig. 3.4(a) represents the original digit-serial multiplier and Fig. 3.4(b) is the proposed multiplier. The detailed explanation of the registers utilized for the traditional digit-serial multiplier is shown in Table 3.3. In Fig. 3.4(b), the number of padding bits for register V_v is determined by first calculating $q = m/d$. As can be seen from Table 3.3, register U_v is used to store the new multiplier representing U from ‘*shiftModU*’ for the next cycle. Similarly, the product during each multiplication cycle is stored in the register Z_v . This approach is very efficient in hardware cost since it uses left-shift and add (xor) operations. In each cycle, $V(i)$ is multiplied by ‘*tempU*’ which contains $U.x^{di} \bmod f(x)$ of the current cycle. The computation of $U.x^{d(i+1)} \bmod f(x)$ is performed in parallel during the multiplication of $V(i)$ and ‘*tempU*’. ‘*tempU*’ is left-shifted by d bits followed by reduction with polynomial $f(x)$. Register Z_v contains $V_{q-1} \cdot (A.x^{dq-1}) \bmod f(x)$ at the final iteration, but with degree greater than m . For this reason, reduction of Z_v is also needed to obtain the final result Z . Therefore, the latency of a digit-serial multiplier is $O(m/d)$, where m is the field size over $GF(2^m)$ and d is the digit size.

Modified digit-serial Multiplication

Recall that for the traditional digit-serial multipliers implemented in the previous section, the number of clock cycles required to perform multiplication is defined by $q = \lceil m/d \rceil$.

Algorithm 3.3: Traditional digit-serial multiplication in $GF(2^m)$

Input: $U(x), V(x) \in GF(2^m)$, irreducible polynomial $f(x)$ of degree m

Output: $Z(x) = U(x) \cdot V(x) \bmod f(x)$

1. $Z_v = 0$;
 2. $V = V_0 + V_1x^d + \dots + V_{q-1}x^{d(q-1)}$, where $V_i = \sum_{j=0}^{d-1} v_{id}x^j$;
 - 3.1 **for** $i = 0$ to $q - 1$ **do**
 - 3.2 $Z_v = Z_v \oplus V_i U$; $U_v = U_v \cdot x^d \bmod f(x)$
 - 3.3 **end for**
 4. $Z = Z_v \bmod f(x)$
 5. **Return** $Z(x)$
-

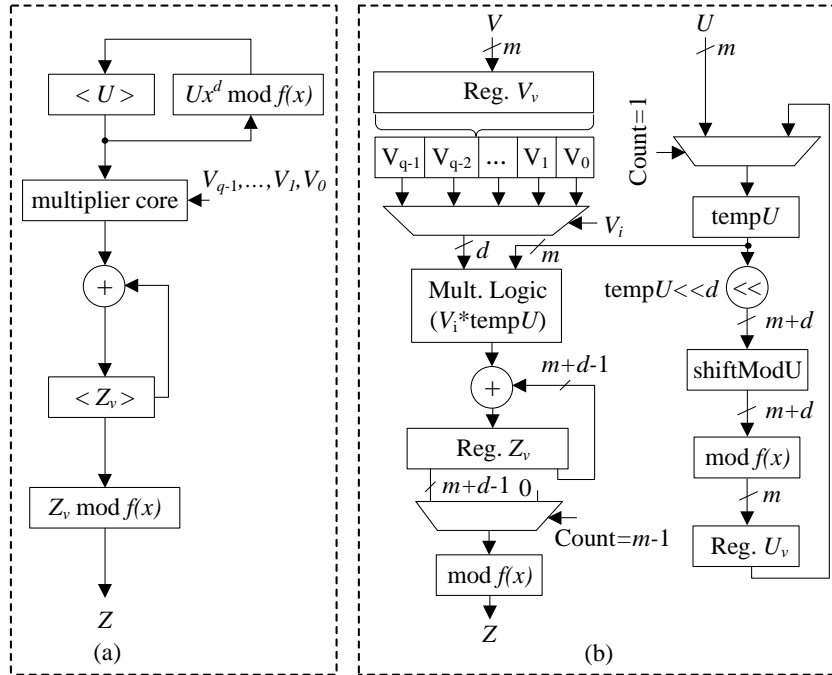


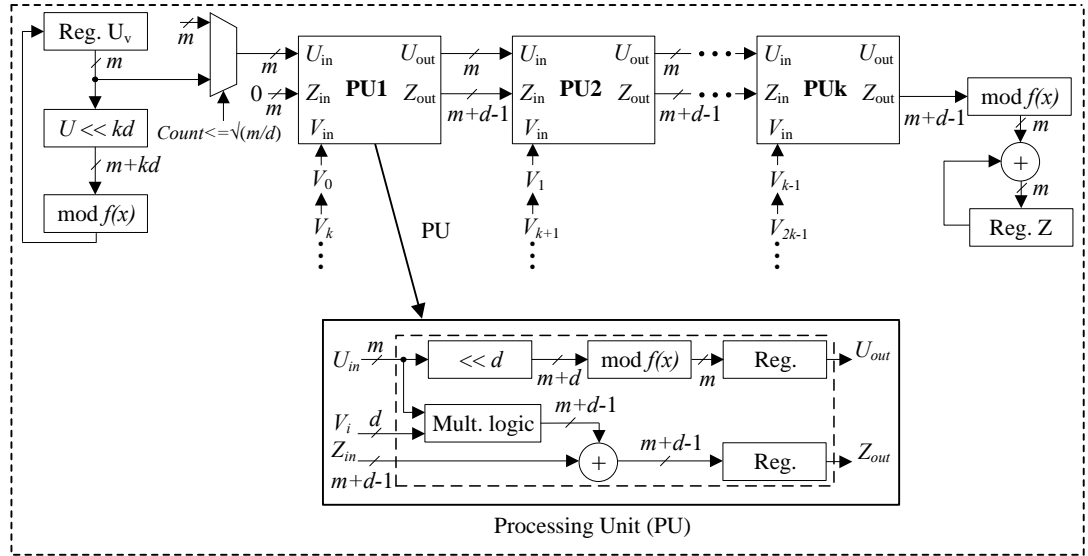
Figure 3.4: Traditional digit-serial multiplication architecture over $GF(2^m)$: (a) original [43, 46] and (b) proposed.

In this modified (or systolic) multiplier, a pair of integers p and k are now used such that $kp = q$ and $k = \sqrt{q}$. Zeros also need to be padded to the multiplicand V so that $q =$

Table 3.3: *Registers used for traditional digit-serial multiplication in $GF(2^m)$.*

Register Name	Description	Bit-size
U_v	Used to store the new multiplier representing U from 'shiftModU' for the next iteration	m
$f(x)$	Reduction polynomial	$m + 1$
tempU	Used to initialize and perform multiplication	m
shiftModU	Used to perform $U \cdot x^d \bmod f(x)$	$m + d$
V_v	Used for padding the V input by adding '0' bits to the left of the MSB of V .	$m + (qd - m)$
Z_v	Used to store the product during each multiplication iteration	$m + d - 1$

$\lceil m/d \rceil = kp$ can be satisfied. Fig. 3.5 illustrates the hardware architecture of the modified digit-serial multiplier based on Algorithm 3.4 [43]. The top-level architecture design uses k processing units (PUs) to perform the inner loop (lines 3.3 to 3.5) of Algorithm 3.4. The detailed mathematical formulas are discussed in [43].

**Figure 3.5:** *Proposed modified digit-serial field multiplication architecture in $GF(2^m)$.*

The modified multiplier is composed of three main parts consisting of the updating register U_v , the processing units (PUs), and the reduction & accumulation of the Z_{out} partial products. In each clock cycle, register U_v is updated with $Ux^{kd} \bmod f(x)$. The first PU has the value of register U_v as an input for $\sqrt{m/d}$ clock cycles followed by zeros until the multiplication is complete. PU1 also has a constant Z_{in} input of 0. The remaining PUs

Algorithm 3.4: Modified digit-serial multiplication in $GF(2^m)$

Input: $U = (u_{m-1}, \dots, u_1, u_0)$, $V = (v_{m-1}, \dots, v_1, v_0)$, $f(x) = x^m + r(x)$.

Output: $Z = U \cdot V \bmod f(x)$.

1. $Z_v = 0$;
 2. $V = \sum_{i=0}^{kp-1} V_i x^{id}$, where $V_i = \sum_{j=0}^{d-1} v_{(id+j)} x^j$
 - 3.1 **for** $i = 0$ to $p - 1$ **do**
 - 3.2 $W = V$; $U = x^{kd} U \bmod f(x)$
 - 3.3 **for** $j = 0$ to $k - 1$
 - 3.4 $Z_v = Z_v \oplus V_{ik+j} W$; $W = W \cdot x^d \bmod f(x)$;
 - 3.5 **end for**
 - 3.6 **end for**
 4. $Z = Z_v \bmod f(x)$
 5. **Return** $Z(x)$
-

have the U_{in} and Z_{in} inputs of the previous processing unit's U_{out} and Z_{out} , respectively. Fig. 3.5 depicts the internal hardware architecture of each PU. Given the inputs U_{in} , V_{in} , and Z_{in} , the processing element computes the outputs $U_{out} = U_{in} \cdot x^d \bmod f(x)$ and $Z_{out} = Z_{in} \oplus (U_{in} \cdot V_{in})$. Finally, register Z_v is used to add the partial products from the last processing unit's (PU_k) Z_{out} such that $Z_v = \sum_0^{2k} Z_{out}$.

The design for a systolic digit-serial multiplier in $GF(2^{233})$ with the digit size of 8 ($d = 8$) is described in Table 3.4. The number of PUs required is given by $k = \sqrt{\lceil m/d \rceil} = \sqrt{\lceil 233/8 \rceil} = 6$. The scheduling of V_{in} inputs for each processing unit is depicted in Table 3.4. The scheduling of U_{in} inputs for the first processing unit is also shown. The number of PUs and clock cycles for each digit size in $GF(2^{163})$, $GF(2^{233})$, $GF(2^{283})$, $GF(2^{407})$, and $GF(2^{571})$ are depicted in Table 3.5. The number of clock cycles for 233-bit finite field multiplication is given by $2k = 12$ with the addition of one clock cycle for initialization,

making a total of 13 clock cycles to complete a single multiplication.

Table 3.4: *The scheduling of V inputs for each processing unit in $GF(2^{233})$ when $d = 8$.*

Cycle	PU1_ U_{in}	PU1	PU2	PU3	PU4	PU5	PU6
0	U_0	0	0	0	0	0	0
1	U_1	$V[7:0]$	0	0	0	0	0
2	U_2	$V[55:48]$	$V[15:8]$	0	0	0	0
3	U_3	$V[103:96]$	$V[63:56]$	$V[23:16]$	0	0	0
4	U_4	$V[151:144]$	$V[111:104]$	$V[71:64]$	$V[31:24]$	0	0
5	U_5	$V[199:192]$	$V[159:152]$	$V[119:112]$	$V[79:72]$	$V[39:32]$	0
6	0	$V[247:240]$	$V[207:200]$	$V[167:160]$	$V[127:120]$	$V[87:80]$	$V[47:40]$
7	0	0	$V[255:248]$	$V[215:208]$	$V[175:168]$	$V[135:128]$	$V[95:88]$
8	0	0	0	$V[263:256]$	$V[223:216]$	$V[183:176]$	$V[143:136]$
9	0	0	0	0	$V[271:264]$	$V[231:224]$	$V[191:184]$
10	0	0	0	0	0	$V[279:272]$	$V[239:232]$
11	0	0	0	0	0	0	$V[287:280]$
12	0	0	0	0	0	0	0

Table 3.5: *The number of processing units (PUs) and clock cycles for each digit size implemented in $GF(2^{163})$, $GF(2^{233})$, $GF(2^{283})$, $GF(2^{407})$, and $GF(2^{571})$.*

Digit-size	$GF(2^{163})$		$GF(2^{233})$		$GF(2^{283})$		$GF(2^{409})$		$GF(2^{571})$	
	#PUs	Clock cycles	#PUs	Clock cycles	#PUs	Clock cycles	#PUs	Clock cycles	#PUs	Clock cycles
1	13	27	16	33	17	35	21	43	24	49
2	10	21	11	23	12	25	15	31	16	35
4	7	15	8	17	9	19	11	23	12	25
8	5	11	6	13	6	13	8	17	9	19
16	4	9	4	9	5	11	6	13	6	13
32	4	7	3	7	3	7	4	9	5	11
64	2	5	2	5	3	7	3	7	3	7

Squaring in $GF(2^m)$ is the field multiplication of two identical binary polynomials. A squaring operation is often used since it can be more efficient than field multiplication. This consideration depends on the performance of field multiplication utilized, since if multiplication becomes more efficient than squaring, then the squaring method can be replaced.

3.4.3 Finite Field Modular Reduction

A modular reduction needs to be performed on the result of the multiplication, squaring, and inversion to ensure that it exists within the binary field chosen. The reduction used in the architecture of traditional and systolic digit-serial multiplication to perform mod $f(x)$ is depicted in Fig. 3.6. As can be seen from Fig. 3.6, the reduction is performed by

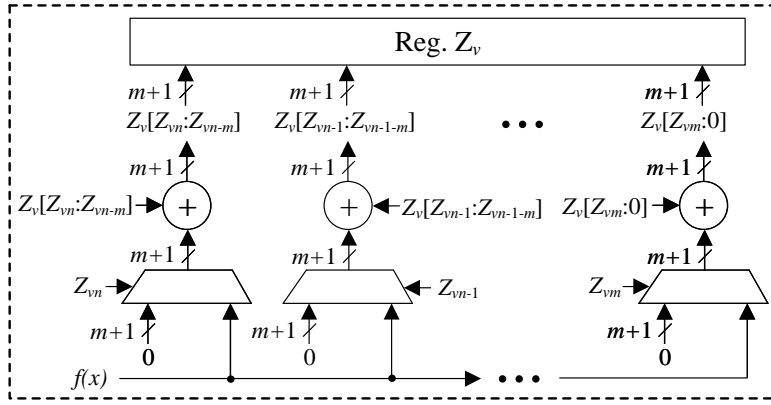


Figure 3.6: Hardware for modular reduction in $GF(2^m)$.

connecting all bit positions representing a binary polynomial with a degree more than m to be selectors to each multiplexer. For each ‘1’ bit of Z_v greater than Z_{m-1} , the contents of Z_v from $Z_v(C_n$ downto $C_{n-m})$ for the bit size of m will be added to the reduction polynomial such that $Z_v(Z_n$ downto $Z_{n-m}) = Z_v(Z_n$ downto $Z_{n-m}) \text{ xor } f(x)$, where $n > m$. Hence, it uses the reduction polynomial $f(x)$ of the corresponding binary field so that a binary polynomial $Z_v(x)$ is reduced to $Z(x)$, where $Z(x) = Z_v(x) \bmod f(x)$. Modular reduction in arbitrary polynomials can be done one bit at a time for bit-serial polynomial multiplication. Note that the irreducible polynomials recommended by NIST in the FIPS 186-2 standard for 163, 233, 283, 409 and 571 bits are $f(x) = x^{163} + x^7 + x^6 + x^3 + 1$, $f(x) = x^{233} + x^{74} + 1$, $f(x) = x^{283} + x^{12} + x^7 + x^5 + 1$, $f(x) = x^{409} + x^{87} + 1$, and $f(x) = x^{571} + x^{10} + x^5 + x^2 + 1$, respectively.

3.4.4 Finite Field Inversion

Inversion in $\text{GF}(2^m)$ is the most expensive (or time consuming) operation for FFA. However, it is required only once per ECPM to convert affine coordinates from Jacobian coordinates. There are mainly two kinds of algorithms available for inversion in $\text{GF}(2^m)$,

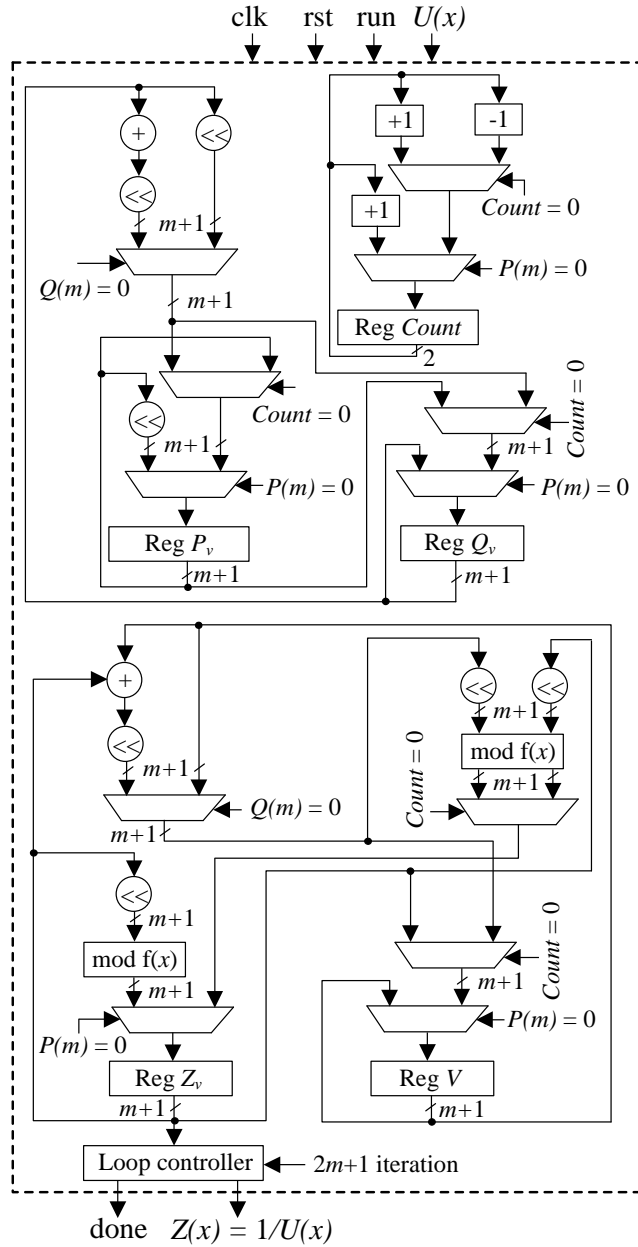


Figure 3.7: Hardware Architecture for field inversion in \mathbb{F}_2^m .

such as (1) Extended Euclidean-based algorithms (EEA) and (2) Fermat's Little Theorem (FLT) based algorithms. Based on FLT, for any a in a finite field with q elements $a^q = a$ or $a^{q-2} = a^{-1}$. For the binary field $\text{GF}(2^m)$ $q = 2^m$, hence $a^{-1} = a^{2^m-2}$. Therefore, a large exponentiation is the main difficulty in this method. On the other hand, EEA-based algorithms include: (a) the simplest EEA, (b) the almost inversion algorithm, and (c) the modified almost inversion algorithm. Also, other inversion algorithms exist in the literature, such as the Itoh-Tsujii inversion algorithm based on FLT [49] and the Montgomery inversion algorithm [50]. In this paper, we adopt the almost inversion algorithm which is the most commonly used algorithm for field inversion. Inversion of a non-zero field element $U(x) \in \mathbb{F}_2^m$ is $Z(x) \in \mathbb{F}_2^m$, where $UZ = 1 \bmod f(x)$. The basic inversion algorithm for $\text{GF}(2^m)$ was proposed in [51], then implemented in [52] and [53]. The almost inversion algorithm is depicted from [2, 52] and the corresponding hardware architecture is demonstrated in Fig. 3.7. A high-performance finite field inversion architecture is achieved using this algorithm. As can be seen from Fig. 3.7, all internal signals such as V , Q_v , Z_v , and P_v are $m + 1$ bits long because the reduction polynomial is also $m + 1$ bits long, and 'Count' has the size of 2 bits. Register P_v is initialized with $U(x)$ padded with a '0' bit. Register Q_v is initialized with the reduction polynomial of the binary field used. Register Z_v is initialized with the binary value of 1. Registers V and $Count$ are initialized with the value of zero. In this architecture, a series of finite field additions and shift operations are required to complete an inversion over $\text{GF}(2^m)$. Note that addition and shift operations are basic and faster operations for hardware implementation. The computation of division like Z_v/x or multiplication by x is performed by a shift operation. The modular reduction is performed by using the addition operation when the degree of the polynomial is m or higher than m . Finally, the field inversion $Z(x) = 1/U(x) \bmod f(x)$ is achieved using the modified EEA algorithm. The latency of this inversion over the binary field $\text{GF}(2^m)$ is $2m + 1$ clock cycles. The division $V(x)/U(x) \bmod f(x)$ is performed by first finding the

inverse $U^{-1}(x)$ and then performing the multiplication, where $V(x) \cdot U^{-1}(x) = V(x)/U(x)$.

Table 3.6: *Complexity of $GF(2^m)$ finite field arithmetic operations in terms of latency*

Operation	Complexity in terms of latency
Addition/subtraction	1
Bit-serial multiplier	$m + 1$
Traditional digit-serial multiplier	$\lceil m/d \rceil$
Modified digit-serial multiplier	$2 * \lceil \sqrt{m/d} \rceil + 1$
Inversion (bit-serial)	$2m + 1$

Table 3.6 illustrates the latency (clock cycles) for FFA. As shown in Table 3.6, addition is the most efficient operation, whereas inversion is the most time-consuming operation due to the large number of clock cycles required. As we can see in Table 3.6, addition and inversion take 1 and $2m + 1$ clock cycles, respectively. Also, three types of finite field multiplication are implemented, where the modified digit-serial approach takes fewer clock cycles than other finite field multiplication methods. Therefore, the latency for bit-serial, traditional digit-serial, and modified systolic digit-serial multiplication is $m + 1$, $\lceil m/d \rceil$, and $2 * \lceil \sqrt{m/d} \rceil + 1$, respectively. The detailed clock cycles required for FFA are tabulated in the next section.

3.5 Implementation Results and Comparison

This section presents the results and analysis of hardware implementations of FFA (e.g. multiplication and inversion) over the binary field $GF(2^m)$. The main focus of this research is to produce an efficient hardware implementation of finite field multiplication and inversion, reducing the latency through the use of efficient algorithms and better hard-

ware architectures. The proposed binary field arithmetic is implemented on both FPGA and ASIC platforms and the overall performance is compared to the most significant implementations in the literature.

3.5.1 FPGA Implementation Results

The presented finite field arithmetic is coded in synthesizable VHDL and implemented using Xilinx ISE 14.7 synthesis technologies. The target FPGAs selected are the Xilinx Virtex-7 and Virtex-6 which contain sufficient resources. The FPGA implementations were simulated using both ModelSim PE and Xilinx ISim. We first provide the results of bit-serial multiplication, then compare with related implementations in the literature.

Table 3.7: *Performance and comparison of FPGA-based implementation results of finite field multiplication using bit-serial approach over $GF(2^m)$.*

Work	Tech.	Field (Length)	Latency (CCs) ^a	CP ^b (ns)	Frequency f (MHz)	Time (ns)	Area (slices)	Area×Time (slices × μs)	Performance (1/AT)×10 ³	TR ^c (Gbps)
Fig. 3.3(b)	Virtex-7	163	164	1.771	564.65	290.4	98	28.5	35.0	0.56
		233	234	1.919	521.09	449.0	134	60.2	16.6	0.52
		283	284	2.334	428.43	662.9	189	125.3	8.0	0.43
		409	410	2.411	414.74	988.5	227	224.4	4.5	0.41
		571	572	2.682	372.88	1534.1	409	627.4	1.6	0.37
	Virtex-6	163	164	1.886	530.22	309.3	105	32.5	30.8	0.53
		233	234	2.236	447.14	523.2	287	150.2	6.7	0.45
		283	284	2.630	380.21	746.9	330	246.5	3.8	0.38
		409	410	2.699	370.48	1106.6	368	407.2	2.5	0.37
		571	572	2.973	336.38	1700.6	453	770.4	1.3	0.34
[108]	Virtex-6	163	-	3.727	268.30	-	461	-	-	-
		233	-	3.660	273.20	-	522	-	-	-
		283	-	4.216	237.20	-	677	-	-	-
		409	-	4.125	242.40	-	969	-	-	-
		571	-	3.512	284.70	-	1249	-	-	-
[109]	Virtex-6	163	-	-	-	9.1	8579*	78.1	12.8	17.91
[118]	Virtex-2	163	163	5.621	177.90	916.0	225	206.1	4.9	0.18

a. CCs = clock cycles, b. CP = clock period. *LUTs, c. TR = Throughput Rate

Table 3.7 depicts the FPGA-based implementation results for the proposed architec-

ture in Fig. 3.3, and the existing multiplications over $GF(2^m)$. The proposed bit-serial multiplication is implemented in a Xilinx Virtex-7 and Virtex-6 FPGA, which supports all five NIST fields. As can be seen from Table 3.7, the latency and space complexities increase with the increase of field size. The area/space complexity of the proposed multiplier in a Virtex-6 is compared with those in [108]. It can be seen that our proposed design is area-efficient, takes nearly one-third the number of slices than those [108]. The proposed hardware for multiplication in [109] is very efficient with timing, but requires more area to implement. As we can see in Table 3.7, the multiplication structure of [109] has the lowest latency among the other designs. However, the direct comparison is difficult because they provide only LUTs where we have given implementation results in terms of occupied slices. The proposed bit-serial structure of [118] is implemented in a Xilinx Virtex-2 FPGA, and their design needs three times as much time and two times as much area than those our design.

In this paper, digit-serial multipliers using a traditional approach (latency $\lceil m/d \rceil$) are implemented for all five NIST standards between 163-bit and 571-bit. Digit-serial multipliers with digit sizes of 1, 2, 4, 8, 16, 32 and 64 bits are implemented in Virtex-7 and Virtex-6 FPGAs. The FPGA-based implementation results for $GF(2^{163})$, $GF(2^{233})$, $GF(2^{283})$, $GF(2^{407})$, and $GF(2^{571})$ using a traditional digit-serial multiplier is depicted in Figs. 3.8 and 3.9. The results show that an increase in area from higher digit sizes for a reduction of time is able to achieve much better area-time (AT) product than bit-serial multiplication. As the digit size doubles, the number of clock cycles required to complete a single multiplication are halved due to the fact that the number of clock cycles is determined by $\lceil m/d \rceil$ for traditional digit-serial multipliers. The trade-off for this reduction is the number of clock cycles is the increase of the area. For this reason, the performance ($1/AT$) for all implementations is analyzed thoroughly, as shown in Figs. 3.8 and 3.9, then we choose which design gives better performance. These results reveal that

area-time is less improved from digit sizes 1 to 32, which the trade-off for more area and less time is better for the 64-bit digit-serial multiplier. As can be seen from the figure, a higher digit-level (or 64-bit version) multiplication gives better performance than other digit levels. Therefore, 64-bit digit-serial multiplication results are presented in Table 3.8, and compared with related work in the literature.

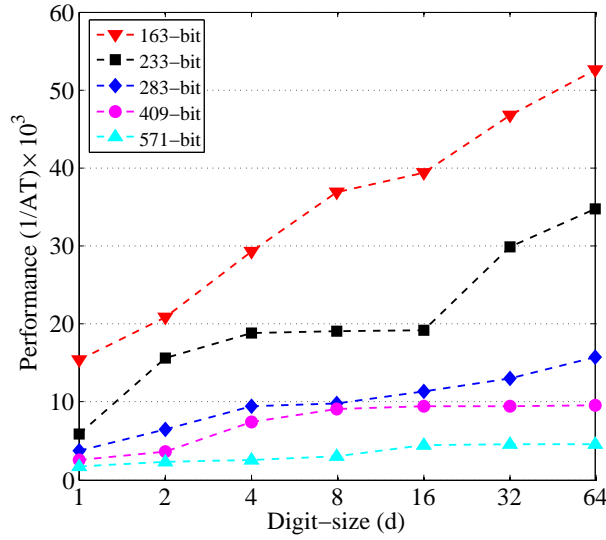


Figure 3.8: Performance analysis of traditional digit-serial multiplication with various digit sizes in Virtex-7 FPGA.

In [119], 64-bit word-level (digit-serial) multiplication over $GF(2^m)$ using a polynomial basis was proposed. 163-bit and 571-bit digit-serial finite field multiplications using different digit sizes were implemented in [35], where 11-bit and 24-bit digit sizes, respectively, provide a better results than their other implementations. Therefore, we have compared their best implementation results with our best implementations. The results of [115] shows implementation of 163-bit digit-serial multiplication only. As one can see from Table 3.8, our proposed implementations for Fig. 3.4(b) provide better results in terms of delay than [35], [115], and [119].

The modified digit-serial multiplier is implemented for $GF(2^{163})$, $GF(2^{233})$, $GF(2^{283})$, $GF(2^{407})$, and $GF(2^{571})$ with the digit sizes of 1, 2, 4, 8, 16, 32 and 64 bits. Note that a

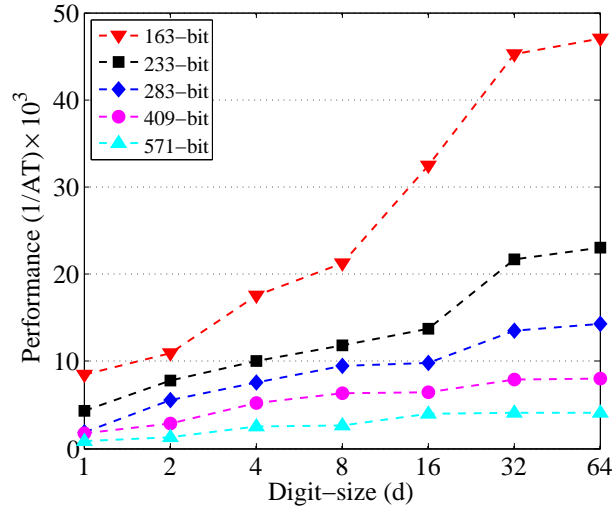


Figure 3.9: Performance analysis of traditional digit-serial multiplication with various digit sizes in Virtex-6 FPGA.

Table 3.8: Performance and comparison of FPGA-based implementation results of digit-serial finite field multiplication over $GF(2^m)$.

Work	Tech.	Field (Length)	Latency (CCs)	CP (ns)	Frequency f (MHz)	Time (ns)	Area (slices)	Area×Time (slices × μs)	Performnace (1/AT)×10 ³	TR (Gbps)
Fig. 3.4(b)[†]	Virtex-7	163	3	3.202	312.3	9.6	1975	19.0	52.6	17.0
		233	4	3.285	304.4	13.1	2218	28.8	34.7	17.9
		283	5	3.425	292.0	17.1	3697	63.2	15.8	16.6
		409	7	3.429	291.6	24.0	4335	104.0	9.6	17.0
		571	9	3.437	290.9	30.9	7046	217.7	4.6	18.5
	Virtex-6	163	3	3.541	282.4	10.6	2011	21.3	47.0	15.4
		233	4	3.607	277.2	14.3	3040	43.4	23.0	16.3
		283	5	3.717	269.0	18.6	3765	70.0	14.3	15.2
		409	7	3.752	266.5	26.3	4732	124.4	8.0	15.6
		571	9	3.788	263.9	34.0	7129	242.4	4.1	16.8
[35]	Virtex-5	163	15 ^a	2.360	423.7	35.0	1179 ^c	41.7	24.0	4.7
		571	24 ^b	2.780	359.7	67.0	8051 ^c	537.2	1.9	8.5
[115]	Virtex-4	163	43	4.032	248.0	173.0	1475 ^c	255	3.9	0.9
[119]	Virtex-2	163	48 ^d	6.667	150.0	320.0	3344 ^c	1070.1	0.9	0.5
		233	88 ^d	6.667	150.0	586.7	3344 ^c	1963.0	0.5	0.4
		283	110 ^d	6.667	150.0	733.4	3344 ^c	2451.1	0.4	0.4

[†] Traditional digit-serial multiplication (Fig. 3.4) (latency = $\lceil m/d \rceil$), 64-bit digit-serial multiplication results.

^a 11-bit digit-serial multiplication results, ^b 24-bit digit-serial multiplication results, ^c Area = LUTs, ^d 64-bit digit-serial multiplication results.

Table 3.9: *FPGA-based implementation results of modified digit-serial finite field multiplication over $GF(2^m)$.*

Work	Tech.	Field (Length)	digit size (bits)	Latency (CCs)	CP (ns)	Frequency f (MHz)	Time (ns)	Area (slices)	Area×Time (slices × μs)	Performance (1/AT)×10 ³	Throughput rate (Gbps)
	Virtex-7	163	8	11	1.560	641.0	17.2	1609	27.7	36.1	9.5
		233	16	9	1.828	547.2	16.5	3055	50.4	19.8	14.1
		283	8	13	1.538	650.1	20.0	3080	61.6	16.2	14.2
		409	32	9	1.980	505.0	17.8	6140	109.3	9.1	23.0
		571	16	13	1.834	545.4	23.8	8930	212.5	4.7	24.0
Fig. 3.5*											
	Virtex-6	163	8	11	1.715	583.1	18.9	1897	35.9	27.9	8.6
		233	16	9	1.831	546.2	16.5	3271	54.0	18.5	14.1
		283	8	13	1.701	587.9	22.1	3233	71.4	14.0	12.8
		409	32	9	2.249	444.7	20.2	7411	149.7	6.7	23.0
		571	16	13	2.058	486.0	26.8	9873	264.6	3.8	21.5

* Modified digit-serial multiplication (Fig. 3.5) (latency = $2 * \lceil \sqrt{m/d} \rceil + 1$).

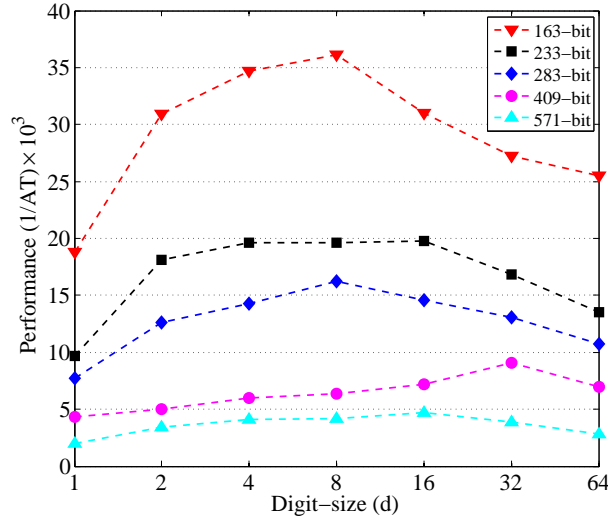


Figure 3.10: Performance analysis of modified digit-serial multiplication with various digit sizes in Virtex-7 FPGA.

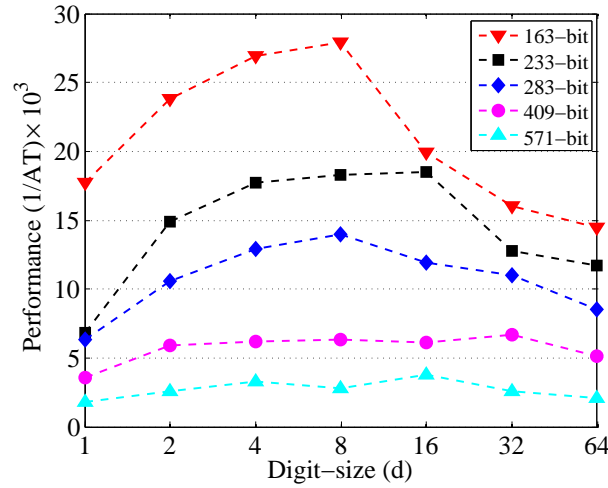


Figure 3.11: Performance analysis of modified digit-serial multiplication with various digit sizes in Virtex-6 FPGA.

digit size of 1 refers to the digit-serial multiplier and for the bit-serial multiplier a separate module is implemented. The Virtex-7 and Virtex-6 implementation results of these modified multipliers are demonstrated in Figs. 3.10 and 3.11, respectively. The number of clock cycles required to complete a single multiplication using this method is given by

$2 * \lceil \sqrt{m/d} \rceil + 1$. The modified multiplication with the best performance is for the digit size of 16 for the 163-bit, 233-bit, and 571-bit implementations, whereas the best performance for the 283-bit and 409-bit is the digit size of 8 and 16, respectively. Both the modified (or systolic) digit-serial multipliers and the traditional digit-serial multipliers have their own benefits. The multiplier with the best area (slices) is achieved by the traditional bit-serial multipliers, but this takes more computation time. For high-speed implementations, the modified digit-serial multiplier should be used for the best computation time at the expense of few more slices. However, an ASIC-based implementation of the modified digit-serial multiplier offers better performance than the traditional digit-serial multiplier. In order to achieve a high-performance ECC processor, the multiplier with the most optimal area-time can be used to implement elliptic curve group operations.

In this part, the FPGA-based inversion implementation results are presented in a bit-serial approach for the NIST recommended fields $GF(2^{163})$, $GF(2^{233})$, $GF(2^{283})$, $GF(2^{407})$, and $GF(2^{571})$. We have proposed an inverter architecture based on the modified Euclidean algorithm, and implemented it in Xilinx Virtex-7 and Virtex-6 FPGAs with a speed grade of -2. Implementation results and performance comparisons are introduced in Table 3.10. The performance for all FPGA implementations is computed because of the clear representation of the trade-off between speed and area. Inversion over $GF(2^m)$ based on the modified Itoh-Tsujii algorithm was proposed in [124]. They implemented inversion for all NIST fields on a Xilinx Virtex-4 FPGA. As shown in Table 3.10, their proposed design needs fewer clock cycles than our design, but has a longer critical path, hence takes more computation time than our bit-serial inversion. Besides, their design requires more area (LUTs) to implement. In [35], the inversion algorithm based on the digit-serial (digit size between 1 and 9) approach was implemented in a Virtex-5 FPGA. According to their recommendation, the digit sizes 1 and 5 provide better results. For this reason, we have compared our implementation results with their 1-bit digit-serial

Table 3.10: *Performance and comparison of FPGA-based implementation results of finite field inversion over $GF(2^m)$.*

Work	Tech.	Field (Length)	Latency (CCs) ^b	CP ^b (ns)	Frequency f (MHz)	Time (ns)	Area (slices)	Area×Time (slices × μ s)	Performance (1/AT) × 10 ³	Throughput rate (Mbps)
Fig. 3.7	Virtex-7	163	327	3.338	299.54	1092	362	395	2.53	149
		233	467	3.347	298.78	1563	597	933	1.07	149
		283	567	3.347	298.78	1898	599	1137	0.88	149
		409	819	3.726	268.38	3052	830	2533	0.40	134
		571	1143	3.726	268.38	4259	963	4101	0.24	134
	Virtex-6	163	327	3.709	269.64	1213	480	582	1.72	134
		233	467	3.709	269.64	1732	542	939	1.07	134
		283	567	3.718	268.96	2108	618	1303	0.77	134
		409	819	3.726	268.38	3052	975	2976	0.34	134
		571	1143	3.726	268.38	4259	1162	4949	0.20	134
[124]	Virtex-4	163	154	8.030	124.53	1237	6104 ^c	7550	0.13	132
		233	181	8.623	115.97	1561	9076 ^c	14165	0.07	149
		283	232	10.878	91.93	2524	13526 ^c	341360	0.34	112
		409	474	10.216	97.89	4842	13608 ^c	65894	0.02	85
		571	599	11.823	84.58	7082	30509 ^c	216062	0.005	81
[35]	Virtex-5	163	163	1.730	578.03	282	836 ^c	235	4.26	578
		571	571	1.780	561.80	1016	2878 ^c	2925	0.34	562
[118]	Virtex-2	163	-	5.584	179.10	26000	1505	39130	0.03	6.3
[115]	Virtex-4	163	327	6.803	147.00	2221	1169	2596	0.39	73

^aCCs = clock cycles, ^bCP = clock period, ^cLUTs

inversion. In [118] and [115], 163-bit finite field inversion was presented. As can be seen from Table 3.10, our proposed inversion architecture provides better performance (e.g. time, area) than [118] and [115].

3.5.2 ASIC Implementation Results and Comparison

All FFAs are also synthesized on the ASIC 65-nm CMOS STMicroelectronics standard cell library using Synopsys Design Compiler. The standard logic-cell library for normal case analysis assumes 1.2 V and 25°C for this implementation. ASIC implementations of binary field arithmetic such as multiplication and inversion were simulated using ModelSim/Quarta Sim. The proposed FFA implementations are also compared with the most

significant ASIC-based implementations in the literature.

Bit-serial multiplication and inversion are implemented in ASIC 65-nm CMOS technology and related work compared with our implementations. Table 3.11 demonstrates the proposed ASIC-based synthesis results of bit-serial multiplication and inversion. As shown in Table 3.11, the time and area required for finite field arithmetic (e.g. multiplication or inversion) increases with increasing field size. In[107], an ASIC-based implementation of finite field inversion with normal basis was proposed. They used the same 65-nm CMOS technology as we have used for this implementation. They implemented four NIST fields except $GF(2^{233})$ out of the five. According to their discussion, inversion for $GF(2^{233})$ is not suitable using their proposed hybrid-double multiplier method. Their design supports a digit-size approach, which reduces the latency with increasing digit size. However, their design requires a much larger area than our proposed design with the same technology. For fair comparison, we have calculated $\text{area} \times \text{time}$ (AT) for both designs, then compared their AT value with our AT. We find that our bit-serial inversion gives 2-3 times better performance in terms of AT than their digit-level inversion. Both multiplication and inversion over $GF(2^{163})$ using the bit-serial approach for all NIST fields were implemented in [114]. However, they need at least 50% more computation time and around 11 times more area than our design. Besides, the power consumption is not given in [114], whereas energy dissipation for both multiplication and inversion is computed for our designs from the power consumption and latency. As can be seen from the implementation results in Table 3.11, the proposed bit-serial multiplier and inverter consume very low power and this means that they dissipate much less energy. Moreover, we have computed $\text{area} \times \text{time} \times \text{energy}$ (ATE) values for all of our ASIC-based designs.

In this paper, both traditional and modified versions of digit-serial multiplication are implemented on ASIC platforms and their performance compared. Table 3.12 demonstrates the ASIC-based implementation results for all NIST fields with digit levels of 1,

Table 3.11: *Performance and comparison of ASIC-based synthesis results of finite field arithmetic over* \mathbb{F}_2^m .

Work/Technology	Field (Length)	Latency (Cycles)	CP (ns)	Frequency f (MHz)	Time (ns)	Area ($\mu\text{m}^2/\text{KGs}$) ¹	Area \times Time ² (AT)	Power (mW)	Energy ³ (nJ)	ATE ⁴
Multiplication Fig. 3.3(b) (65-nm CMOS)	163	164	0.83	1205	136	9098.0/4.4	1.24/598	4.67	0.63	0.78/0.38
	233	234	0.83	1205	194	11119.7/5.3	2.16/1028	6.52	1.26	2.72/1.30
	283	284	0.83	1205	236	13774.3/6.6	3.25/1557	7.60	1.79	5.82/2.79
	409	410	0.84	1190	344	23104.2/11.1	7.95/3818	10.91	3.75	29.81/14.32
	571	572	0.84	1190	480	35189.1/16.92	16.89/8112	15.28	7.34	123.97/59.54
Inversion Fig. 3.7 (65-nm CMOS)	163	327	0.83	1205	271	22779.1/11.0	6.17/2981	11.66	3.16	19.50/9.42
	233	467	0.83	1205	388	34685.6/16.7	13.46/6480	17.07	6.62	89.10/42.9
	283	567	0.83	1205	471	38145.1/18.3	17.97/8619	19.52	9.19	165.14/79.2
	409	819	0.83	1205	680	57692.4/27.7	39.23/18836	28.55	19.41	761.45/365.60
	571	1143	0.83	1205	949	75600.2/36.3	71.74/34449	39.31	37.30	2676.0/1284.90
Inversion [107] (65-nm CMOS)	163	38	3.49	287	133	136100.0/65.4 [†]	18.05/8698	-	-	-/-
	283	101	2.77	361	280	197657.0/95.0 [†]	55.30/26600	-	-	-/-
	409	156	1.65	606	258	152820.0/73.5 [†]	39.35/18963	-	-	-/-
	571	158	2.33	429	368	407086.0/195.7 [†]	149.89/72018	-	-	-/-
	163	164	1.23	811	202	123800.0/12.4 [*]	25.00/2501	-	-	-/-
Multiplication [114] (0.18- μm CMOS)	233	234	1.27	785	298	171000.0/17.1 [*]	50.96/5096	-	-	-/-
	283	284	1.29	774	367	209200.0/20.9 [*]	76.77/7678	-	-	-/-
	409	410	1.33	754	544	295700.0/29.6 [*]	160.86/16086	-	-	-/-
	571	572	1.32	758	755	416900.0/41.7 [*]	314.76/31476	-	-	-/-
	163	325	1.23	811	401	123800.0/12.4 [*]	49.64/4964	-	-	-/-
Inversion [114] (0.18- μm CMOS)	233	465	1.27	785	592	171000.0/17.1 [*]	101.23/10123	-	-	-/-
	283	565	1.29	774	730	209200.0/20.9 [*]	152.72/15272	-	-	-/-
	409	817	1.33	754	1083	295700.0/29.6 [*]	320.24/32024	-	-	-/-
	571	1141	2.32	758	1506	416900.0/41.7 [*]	627.85/62785	-	-	-/-
	163	325	1.23	811	401	123800.0/12.4 [*]	49.64/4964	-	-	-/-

1. KGs = Kilo-Gates, 2. AT = Area \times Time = ($\text{mm}^2 \times \text{ns}$)/(Kilo-Gates \times ns), 3. Energy (nJ/operation) = power \times time, 4. ATE = Area \times Time \times Energy ($\text{mm}^2 \times \text{ns} \times \text{nJ}$)/(KGs \times ns $\times \mu\text{J}$). The gate count was computed from the required area divided by the NAND gate area.
[†]NAND2x1 gate area = 2.08 μm^2 for 65-nm CMOS Technology. *NAND2x1 gate area = 5x2 = 10 μm^2 for 0.18- μm CMOS Technology.

Table 3.12: Performance and comparison of ASIC-based digit-serial finite field multiplication over \mathbb{F}_2^m .

Work	Tech.	Field (Length)	Digit size (bits)	Latency (Cycles)	Frequency f (MHz)	Time (ns)	Area ¹ (μm^2 /KGs)	Area \times Time ² (AT)	Energy ³ (nJ)	ATE ⁴
		163	1	163	549	297	10421/5.0	3.10/1485	0.89	2.76/1.32
			2	82	549	149	13047/6.3	1.94/939	0.57	1.11/0.53
			4	41	549	75	18758/9.0	1.41/675	0.40	0.56/0.27
			8	21	549	38	31756/15.3	1.21/581	0.41	0.50/0.24
			16	11	549	20	99574/47.9	1.99/958	0.71	1.41/0.68
			32	6	500	12	203647/97.9	2.44/1175	0.84	2.05/0.99
			64	3	196	15	333133/160.2	5.00/2403	2.15	10.75/5.17
			1	233	549	424	14758/7.1	6.26/3010	2.28	14.27/6.86
			2	117	549	213	18622/9.0	3.97/1916	1.13	4.49/2.17
			4	59	549	107	26442/12.7	2.84/1364	0.79	2.24/1.08
		233	8	30	549	55	44516/21.4	2.43/1168	0.83	2.02/0.97
			16	15	549	27	102206/49.1	2.79/1340	0.78	2.18/1.05
			32	8	498	16	267115/128.4	4.30/2064	1.48	6.36/3.05
			64	4	183	22	475815/228.8	10.41/5006	3.95	41.12/19.77
			1	283	546	518	17877/8.6	9.26/4453	2.64	24.45/11.76
			2	142	546	260	22618/10.9	5.88/2832	1.66	9.76/4.70
			4	71	546	130	32618/15.7	4.24/2039	1.21	5.13/2.48
			8	36	546	66	57953/27.9	3.82/1838	1.33	5.08/2.44
			16	18	546	33	182811/87.9	6.02/2895	2.17	13.06/6.28
			32	9	463	19	440376/211.7	8.56/4115	2.82	24.14/11.60
Fig. 3.4(b) ⁵	65-nm	283	64	5	168	30	526432/253.1	15.71/7555	6.37	100.00/9.48

1. KGs = Kilo-Gates, 2. AT = Area \times Time = ($\text{mm}^2 \times \text{ns}$)/(Kilo-Gates \times ns) 3. Energy (nJ/operation) = power \times time, 4. ATE = Area \times Time \times Energy ($\text{mm}^2 \times \text{ns} \times \text{nJ}$)/(KGs \times ns \times μ s). ⁵Traditional digit-serial, [†]Implemented in [106]

Table 3.12: Performance and comparison of ASIC-based digit-serial finite field multiplication over \mathbb{F}_2^m

Work	Tech.	Field (Length)	Digit size (bits)	Latency (Cycles)	Frequency f (MHz)	Time (ns)	Area ¹ (μm^2 /KGs)	Area \times Time ² (AT)	Energy ³ (nJ)	ATE ⁴
			1	409	546	748	25685/12.3	19.21/9200	5.51	105.85/50.69
			2	205	546	375	32415/15.6	12.16/5850	3.49	42.44/20.42
			4	103	546	188	48239/23.2	9.07/4362	3.10	28.12/13.52
		409	8	52	546	95	82492/39.7	7.84/3771	3.00	23.52/11.31
			16	26	546	48	242485/116.6	11.64/5597	4.53	52.73/25.35
			32	13	463	28	467447/224.7	13.09/6292	4.74	62.05/29.82
			64	7	131	53	721793/347.0	38.25/18391	17.04	651.78/313.38
			1	571	546	1045	35996/17.3	37.62/18078	10.76	404.79/194.52
			2	286	546	523	45263/21.8	23.67/11401	6.86	162.38/78.21
			4	143	546	262	65524/31.5	17.17/8253	6.40	109.88/52.82
		571	8	72	546	132	118946/57.2	15.70/7550	6.01	94.36/45.38
			16	36	500	72	358884/172.5	25.84/12420	9.28	239.79/115.26
			32	18	444	41	655589/315.2	26.88/12923	9.79	263.15/126.51
			64	9	118	76	853741/410.4	64.88/31190	33.64	2182.56/1049.23
		163	4	43	140	307	-/8.5	-/2610	1.2	-/3.13
[106]	65-nm	233	4	61	140	436	-/9.5	-/4142	2.7	-/11.18
		283	4	81	140	578	-/10.4	-/6011	3.9	-/23.44
		163	4	41	140	293	-/10.2	-/2989	1.9	-/5.68
[108] [†]	65-nm	233	4	59	140	421	-/14.0	-/5894	4.0	-/23.58
		163	4	163	140	1164	-/4.9	-/5703	5.0	-/28.5
[112] [†]	65-nm	233	4	233	140	1664	-/7.0	-/11648	10.5	-/122.3
[113]	0.18- μm	283	-	-	-	-	195712/-	-/-	5.61	-/-

1. KGs = Kilo-Gates, 2. AT = Area \times Time = ($\text{mm}^2 \times \text{ns}$)/(Kilo-Gates \times ns) 3. Energy (nJ/operation) = power \times time, 4. ATE = Area \times Time \times Energy ($\text{mm}^2 \times \text{ns} \times \text{nJ}$)/(KGs \times ns $\times \mu\text{s}$). [†]Implemented in [106]

2, 4, 8, 16, 32 and 64 bits. As can be seen from Table 3.12, as the digit size increases between 1 and 32, the computation time decreases, however the area required increases with the digit size. The energy dissipation also increases with the digit size. The area \times time \times energy (ATE) is the best indicator to say which digit-serial version provides better performance. For better analysis of different digit-serial multipliers, ATE is computed for all implementations and presented in Fig. 3.12. As we can see from Fig.3.12, an 8-bit digit-serial multiplier gives better performance in terms of ATE than other digit-serial versions. There only a few ASIC-based digit-serial multiplication results available in the literature, most giving only analytical results. In [106], ASIC implementations for the NIST fields $GF(2^{163})$, $GF(2^{233})$, and $GF(2^{283})$ are presented using the 4-bit digit-serial approach. They have synthesized their design using the 65-nm TSMC CMOS standard cell library, whereas we have used 65-nm STMicroelectronics technology. ASIC-based multiplications for [108] and [112] were introduced in [106]. Implementation results show that our 4-bit version provides better performance than [106] in terms of timing, although their design requires a similar number of clock cycles and area to our design. In addition, we have achieved an energy-efficient design, which dissipates one-third energy than their implementations. A 283-bit digit-level field multiplier was implemented in [113] using 0.18- μm technology. However, our design delivers better performance than their design in terms of area and energy dissipation. Therefore, our design provides better performance (ATE value) than related work in the literature.

Recall the ASIC-based multiplication results using the traditional digit-serial approach in the previous section; the latency (clock cycles), hence the computation time required for this method is far more than the systolic digit-serial approach. For both cases, the computation time decreases and the area increases as the digit size increases. However, the ATE value computed from the modified digit-serial approach is much less than for the traditional digit-serial approach. The hardware implementation results of finite field

Table 3.13: *Performance analysis of ASIC-based modified digit-serial finite field multiplication over \mathbb{F}_2^m .*

Work	Tech.	Field (Length)	Digit size (bits)	Latency (Cycles)	Frequency f (MHz)	Time (ns)	Area ¹ (μm^2 /KGs)	Area \times Time ² (AT)	Energy ³ (nJ)	ATE ⁴
Fig. 3.5 [†] 65-nm		163	1	27	1205	22	65189/31.3	1.46/701	1.66	2.42/1.16
			2	21		17	64610/31.1	1.13/542	1.10	1.24/0.60
			4	15		12	63529/30.5	0.79/380	0.64	0.51/0.24
			8	11		9	71808/34.5	0.66/315	0.39	0.26/0.12
			16	9		7	95293/45.8	0.71/342	0.29	0.21/0.10
			32	7		6	131406/63.2	0.76/367	0.25	0.19/0.09
			64	5		4	198426/95.4	0.82/396	0.24	0.20/0.10
			233	1		33	1205	27	109807/52.8	3.00/1446
		2		23	19	97579/46.9		1.86/895	1.38	2.57/1.24
		4		17	14	99175/47.7		1.40/673	1.15	1.61/0.77
		8		13	11	118118/56.8		1.27/613	0.73	0.93/0.45
		16		9	7	134086/64.5		1.00/482	0.43	0.43/0.21
		32		7	6	203954/98.1		1.18/570	0.34	0.40/0.19
		64		5	4	229250/110.2		0.95/457	0.30	0.29/0.14
		283		1	35	1205		29	163367/78.5	4.75/2280
			2	25	21		128490/61.8	2.67/1282	2.66	7.10/3.41
			4	19	16		133964/64.4	2.11/1016	1.75	3.69/1.78
			8	13	11		143839/69.2	1.55/747	0.91	1.41/0.69
			16	11	9		199950/96.1	1.83/877	0.73	1.34/0.64
			32	7	6		349469/168.0	2.03/976	0.48	0.97/0.47
			64	7	6		459496/220.9	2.67/1283	0.47	1.25/0.60

Fig. 3.5[†]

1. KGs = Kilo-Gates, 2. AT = Area \times Time = ($\text{mm}^2 \times \text{ns}$)/(Kilo-Gates \times ns) 3. Energy (nJ/operation) = power \times time, 4. ATE = Area \times Time \times Energy ($\text{mm}^2 \times \text{ns} \times \text{nJ}$)/(KGs \times ns \times μ s). [†]Modified digit-serial.

Table 3.13: Performance analysis of ASIC-based modified digit-serial finite field multiplication over \mathbb{F}_2^m

Work	Tech.	Field (Length)	Digit size (bits)	Latency (Cycles)	Frequency f (MHz)	Time (ns)	Area ¹ (μm^2 /KGs)	Area \times Time ² (AT)	Energy ³ (nJ)	ATE ⁴
		409	1	43		36	240854/115.8	8.67/4169	9.71	84.19/40.48
			2	31		26	222674/107.1	5.80/2785	5.69	33.00/15.85
			4	23		19	224630/108.0	4.27/2054	3.43	14.65/7.05
			8	17	1205	14	260400/125.2	3.65/1753	2.12	7.74/3.72
			16	13		11	331950/159.6	3.65/1756	1.47	5.37/2.58
			32	9		7	464316/223.2	3.25/1562	0.95	3.09/1.48
			64	7		6	602818/289.8	3.62/1739	0.90	3.26/1.57
			1	49		41	383958/184.6	15.74/7569	17.26	271.67/130.6
			2	35		29	347514/167.1	10.08/4846	9.86	99.39/47.78
			4	25		21	345025/165.9	7.25/3484	5.76	41.76/20.07
		571	8	19	1205	16	409945/197.1	6.56/3154	3.71	24.34/11.70
			16	13		11	470852/226.4	5.18/2490	2.22	11.50/5.53
			32	11		9	568698/273.4	5.19/2470	1.73	8.85/4.27
			64	7		6	827612/397.9	4.97/2387	1.09	5.42/2.60

1. KGs = Kilo-Gates, 2. AT = Area \times Time = ($\text{mm}^2 \times \text{ns}$)/(Kilo-Gates \times ns) 3. Energy (nJ/operation) = power \times time, 4. ATE = Area \times Time \times Energy
($\text{mm}^2 \times \text{ns} \times \text{nJ}$)/(KGs \times ns $\times \mu\text{s}$). [†]Modified digit-serial.

Fig. 3.5[†]

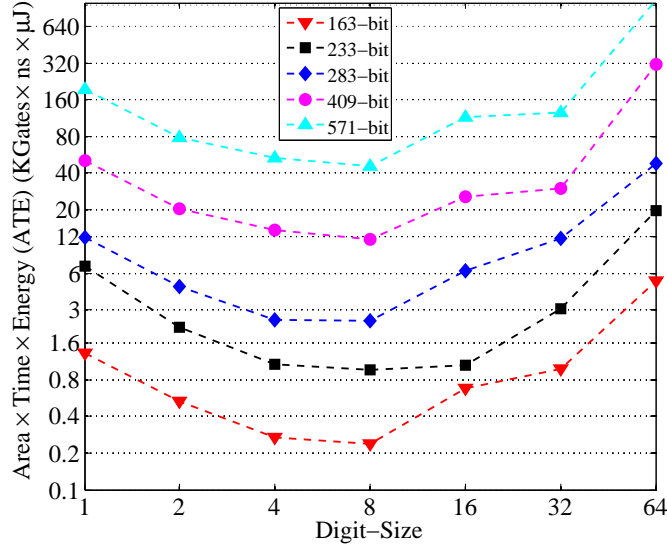


Figure 3.12: *Area × time × energy (ATE) comparison of traditional digit-serial multiplication with digit sizes in \mathbb{F}_2^m .*

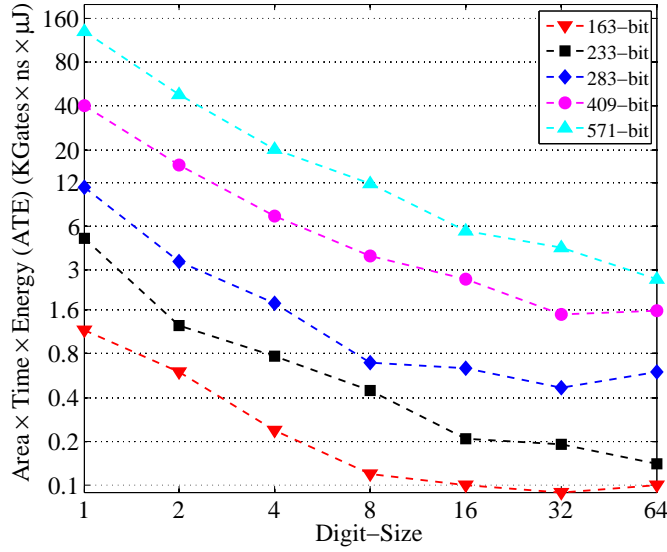


Figure 3.13: *Area × time × energy (ATE) comparison of modified digit-serial multiplication with digit sizes in \mathbb{F}_2^m .*

multiplication with digit sizes of 1, 2, 4, 8, 16, 32 and 64 bits is demonstrated in Table 3.13. Fig. 3.13 illustrates the ATE performance for all NIST fields with the digit sizes. As shown in Table 3.13, the energy dissipation decreases with increasing digit size. For

this reason, higher digit size (e.g. 32-bit or 64-bit) gives better results in terms of ATE performance, as depicted in Fig. 3.13. On the other hand, using the traditional approach the 8-bit digit size delivers better performance. We have compared the performance in terms of ATE for both ASIC-based implementations. We find that the modified version provides 2 to 17 times better ATE performance than the traditional digit-serial approach.

3.6 Conclusion

In this paper, we have presented a fast, area- and energy-efficient FFA unit that can be used for an ECC processor over the binary field $GF(2^m)$. The proposed FFA hardware has been synthesized using Xilinx Virtex-7 and Virtex-6 FPGAs and the ASIC 65-nm CMOS STMicroelectronics standard cell library. A low area complexity multiplication and inversion architecture over $GF(2^m)$ was implemented using a bit-serial approach. In addition, we have implemented a digit-serial multiplication over $GF(2^m)$ with a low area complexity and high speed. Moreover, a structurally improved version of the systolic digit-serial multiplier architecture was presented whose time complexity is much better than basic digit-serial multiplication. Note that the presented multipliers and inverters support all five binary fields $GF(2^{163})$, $GF(2^{233})$, $GF(2^{283})$, $GF(2^{407})$, and $GF(2^{571})$, recommended by NIST. The time and area was computed for all the designs. Furthermore, the energy dissipation was also calculated from the power consumption and latency for the ASIC-based FFA. We found that the bit-serial multiplication/inversion is better with area, whereas the digit-serial version is better in terms of latency and energy. From the comparison analysis of different finite field multiplications and inversions, we have achieved a better performance, either in FPGA implementation or ASIC implementation, than the most significant work in the literature. Finally, we can say that the proposed FFA architecture is well suited for a high-performance ECC processor.

Chapter 4

High-Performance FPGA

Implementation of Elliptic Curve

Cryptography Processor over Binary

Field $\text{GF}(2^{163})$ ¹

4.1 Abstract

Elliptic curve cryptography (ECC) plays a vital role in passing secure information among different wireless devices. This paper presents a fast, high-performance hardware implementation of an ECC processor over binary field $\text{GF}(2^m)$ using a polynomial basis. A high-performance elliptic curve point multiplier (ECPM) is designed using an efficient finite-field arithmetic unit

¹Published as: Md Selim Hossain, Ehsan Saeedi and Yinan Kong, “High-Performance FPGA Implementation of Elliptic Curve Cryptography Processor over Binary Field $\text{GF}(2^{163})$,” *Proceedings of the 2nd International Conference on Information Systems Security and Privacy (ICISSP)*, Rome, Italy, pp. 415-422, 19-21 February, 2016, DOI:10.5220/0005741604150422.

in affine coordinates, where ECPM is the key operation of an ECC processor.

It has been implemented using the National Institute of Standards and Technology (NIST) recommended curves over the field $GF(2^{163})$. The proposed design is synthesized in field-programmable gate array (FPGA) technology with the VHDL. The delay of ECPM in a modern Xilinx Kintex-7 (28-nm) technology is 1.06 ms at 306.48 MHz. The proposed ECC processor takes a small amount of resources on the FPGA and needs only 2253 slices without using any DSP slices. The proposed design provides nearly 50% better delay performance than recent implementations.

4.2 Introduction

With the swift growth of mobile devices and computer applications, cryptography has become a vital tool to ensure the security of data communications and network services. Secret-key cryptography and public-key cryptography (PKC) are two main families of cryptography used for different data-security purposes. ECC [10, 11] and the RSA cryptosystem [8] are the most popular PKCs. The elliptic curve system as applied to cryptography was first proposed in the mid 80s by Koblitz and Miller. This cryptosystem became popular because it offers equivalent security to the traditional RSA with significantly smaller keys. For instance, 163-bit ECC provides equivalent security to 1024-bit RSA [1, 54]. This feature makes ECC very popular for resource-constrained environments such as pagers, PDAs, cellular phones, smart cards and so on [35]. The IEEE [23] and National Institute of Standards and Technology (NIST) [22] have standardized elliptic curve (EC) parameters for $GF(p)$ and $GF(2^m)$. Certicom has provided NIST-recommended EC domain parameters, standard for efficient cryptography in SEC2 [54].

Several FPGA-based efficient ECC hardware architectures and elliptic curve cryp-

tographic processors have been presented in the literature [35, 118, 125–131]. In [128], Ghanmy proposed ECC processor over $GF(2^{163})$ on a FPGA platform for wireless sensor networks (WSN). Reaz’s design [126] can perform ECC over $GF(2^{131})$ and $GF(2^{163})$ on Altera FPGAs. Hasan and Benaissa [127] implemented their ECC processor using the μ -coding technique on Xilinx Spartan-3 FPGAs over $GF(2^{131})$, $GF(2^{163})$, $GF(2^{283})$ and $GF(2^{571})$. A coupled FPGA/ASIC implementation of an elliptic curve crypto-processor over $GF(2^{163})$ is presented in [118], and they used Xilinx Virtex Pro FPGAs and ASIC CMOS 45 nm technology as a hardware platform. Shieh [129], Park et al. [131] also proposed their ECC processor over a binary field using Xilinx FPGAs. An ASIC-based ECC processor is presented over $GF(2^m)$ in [130].

The optimization aim is generally to reduce the latency of an ECPM in terms of the number of required clock cycles. For this, we have concentrated on efficient algorithms and mathematical reformulations for improving finite-field arithmetic operations which are required for ECPM [35, 125, 132, 133]. The arithmetic includes operations defined in finite (Galois) fields, namely $GF(p)$ and $GF(2^m)$ [2]. To the best of the authors’ knowledge, there have been few high-speed hardware implementations of an ECC processor in the literature. Thus an efficient design of an ECC processor is still mandatory for modern cryptographic applications.

In this paper an efficient ECC processor is developed in which ECPM operations are achieved in a very low area (around 2.25K slices without using any DSP slices) and latency (almost 50% less than recent implementations). For this, efficient algorithmic reformulations underlying binary finite field and architectural optimization schemes are explored to improve the operating speed. We propose a data-flow architecture of elliptic curve point doubling (ECPD) and elliptic curve point addition (ECPA) that are required for the ECC processor. An efficient field inversion and multiplication algorithms over $GF(2^m)$ are employed to implement high-performance ECPD and ECPA. Finally, an FPGA-based

high-performance hardware implementation over $\text{GF}(2^{163})$ is proposed, which is the fastest implementation in an affine coordinate system.

The rest of this paper is organized as follows. Section 4.3 introduces a background of groups and fields, Galois finite fields ($\text{GF}(p)$ and $\text{GF}(2^m)$), and ECC theories related to this work. Section 4.4 describes an efficient finite-field algorithm over $\text{GF}(2^m)$, elliptic curve group operations (PD and PA) and hardware architectures. An elliptic curve point multiplication algorithm and cryptographic processor are given in Section 4.5. FPGA implementation results and comparisons with related designs are given in Section 4.6. Finally this paper is summarized in Section 4.7.

4.3 Background

In this section, a brief introduction to abstract algebra, field and group theories relevant to ECC designs used in our hardware implementation is presented.

4.3.1 Groups and Fields

An abelian group $(G, *)$ consists of a set of elements together with a binary operation $*$ which satisfies the following properties :

1. (Associativity) $a * (b * c) = (a * b) * c$ for all $a, b, c \in G$.
2. (Identity) There is an element $e \in G$ such that $a * e = e * a$ for all $a \in G$.
3. (Inverse) for each $a \in G$, there is an element $b \in G$, called the inverse of a , such that $a * b = b * a = e$.

The group operation is generally called addition (+) or multiplication (.). The group is finite if G is a finite set, in which case the number of elements in G is called the order of G .

Fields are abstractions of familiar number systems and their essential properties. A field

$(\mathbb{F}, +, \times)$ is a set of numbers \mathbb{F} with two operations, addition and multiplication, satisfying the following properties:

1. $(\mathbb{F}, +)$ is an abelian group with (additive) identity 0.
2. $(\mathbb{F} \setminus \{0\})$ is an abelian group with (mult.) identity 1.

Division of field elements is represented in terms of multiplication (mult.): for $a, b \in F$ with $b \neq 0$, $a/b = a.b^{-1}$ where $b.b^{-1} = 1$. (b^{-1} is called the inverse of b) [2].

4.3.2 Elliptic Curve Cryptography (ECC)

ECC is the most popular public-key encryption technique. To encrypt data in ECC, it is denoted as a point on an elliptic curve (EC) over a Galois field. A Galois field denoted normally as $\text{GF}(q = p^m)$ is said to be a binary field or characteristic-two finite field if $q = 2^m$. A elliptic curve defined over a Galois field provides a group structure that is used to implement cryptographic systems. The group operations are EC point addition (ECPA) and EC point doubling (ECPD).

There are various coordinate systems to represent elliptic curve points. They vary in the number and type of field operations required to implement PA/PD. In our work, we implement all elliptic curve operations in an affine coordinate system. A non-supersingular elliptic curve E over $\text{GF}(2^m)$ in affine coordinates is the set of solutions to the equation

$$y^2 + xy = x^3 + ax^2 + b \quad (4.1)$$

where $x, y, a, b \in \text{GF}(2^m)$, $b \neq 0$. The coefficients $a, b \in \mathbb{F}_2^m$ specifying an elliptic curve $E(\mathbb{F}_2^m)$ are defined by the NIST standard and then the elliptic curve is defined by (4.1). The number of points on an elliptic curve E is represented by $\#E(\mathbb{F}_2^m)$. It is defined over \mathbb{F}_2^m as nh , where n is the prime order of the curve, and h is an integer called the co-factor.

If $P = (x_1, y_1) \in E$ and $Q = (x_2, y_2) \in E$ (points on the EC), then summing PA and

PD can be respectively derived as

$$\begin{aligned} R(x_3, y_3) &= P(x_1, y_1) + Q(x_2, y_2) \in E, \\ x_3 &= \lambda^2 + \lambda + x_1 + x_2 + a, \\ y_3 &= \lambda(x_1 + x_3) + x_3 + y_1, \end{aligned} \tag{4.2}$$

where $\lambda = (y_2 + y_1)/(x_2 + x_1)$ and $P \neq Q$;

$$\begin{aligned} R(x_3, y_3) &= 2P(x_1, y_1) \in E, \\ x_3 &= \lambda^2 + \lambda + a = x_1^2 + b/x_1^2, \\ y_3 &= x_1^2 + \lambda x_3 + x_3, \end{aligned} \tag{4.3}$$

where $\lambda = x_1 + y_1/x_1$ and $P = Q$;

where $R = 0$ when $x_1 = x_2$ and $y_2 \neq y_1$, or $x_1 = x_2 = 0$. Hence, when $P \neq Q$ we have the PA operation in (4.2) and when $P = Q$ we have the PD operation in (4.3). Using these operations, EC point multiplication kP will be implemented using an ECC- based algorithm [2, 10, 11, 35].

Table 4.1: *comparison of Key length for equivalent security of Symmetric-key and public-key Cryptography [2]*

Symmetric key	Example-algorithm	RSA/DH	ECC in $\text{GF}(p)$	ECC in $\text{GF}(2^m)$
80	SKIPJACK	1024	160	163
112	Triple-DES	2048	224	233
128	AES Small	3072	256	283
192	AES Medium	8192	384	409
256	AES Large	15360	521	571

In 2000, FIPS-2 was recommended with 10 finite fields: 5 prime fields, and 5 binary fields. The binary fields are $\mathbb{F}_2^{163}, \mathbb{F}_2^{233}, \mathbb{F}_2^{283}, \mathbb{F}_2^{409}$ and \mathbb{F}_2^{571} [22]. Prime fields $\text{GF}(p)$ and binary fields $\text{GF}(2^m)$ of similar size are considered to provide almost the same level

of security [1]. Table 4.1 compares symmetric cipher key length, and key lengths for PKC such as RSA, Diffie-Hellman (DH), and ECC (both prime and binary fields). It demonstrates that smaller field sizes can be used in ECC than in RSA and DH systems at a given security level. ECC is many times more efficient than RSA and DH for either private-key operations (such as signature generation and decryption) or public-key operations (such as signature verification and encryption). This makes ECC a promising branch of public-key cryptography [2].

4.4 Hardware Implementation for Finite Field

This section presents all arithmetic algorithms and operations for hardware implementation which are important for ECC. All parameters for NIST elliptic curves over $\text{GF}(2^{163})$ are listed in Table 4.2. The irreducible polynomial is $f(x) = x^{163} + x^7 + x^6 + x^3 + 1$ given for the field $\text{GF}(2^{163})$. A modern Xilinx Kintex-7 (XC7K325T-2FFG900) FPGA with VHDL (VHSIC Hardware Description Language) is used for this hardware implementation. The main components in this ECC design are: polynomial-basis modular addition or field addition, field multiplication, field squaring, field inversion, and elliptic curve group operations (PD and PA).

4.4.1 Polynomial Basis Representation

A polynomial basis (or standard basis) is an extension field used to represent field elements and is very popular. PB is used in our hardware design for the representation of numbers. For the PB representation, the elements \mathbb{F}_2^m are the binary polynomials of degree at most $m - 1$, i.e.

$$\mathbb{F}_2^m = u_{m-1}.x^{m-1} + u_{m-2}.x^{m-2} + \cdots + u_1.x + u_0 = \sum_{i=0}^{m-1} u_i x^i : u_i \in \{0, 1\}$$

For instance, $x^3 + x + 1$ is a polynomial-basis representation for the 4-bit number 1011_2 . For a reduction polynomial or irreducible polynomial, ($f(x)$ be an irreducible binary polynomials of degree m), and $f(x) = x^m + G(x) = x^m + \sum_{i=0}^{m-1} g_i x^i$ where $g_i \in \{0, 1\}$ for $i = 1, \dots, m-1$ and $g_0 = 1$ [2]. For example, $f(x) = x^4 + x + 1 = 10011_2$ is an irreducible polynomial of the finite field $\text{GF}(2^4)$.

Table 4.2: *NIST-recommended elliptic curves over \mathbb{F}_2^{163}*

K-163: $m = 163$, $f(x) = x^{163} + x^7 + x^6 + x^3 + 1$, $a, b = 1, h = 2$							
n=0x	4	00000000	00000000	00020108	A2E0CC0D	99F8A5EF	
x=0x	2	FE13C053	7BBC11AC	AA07D793	DE4E6D5E	5C94EEEE8	
y=0x	2	89070FB0	5D38FF58	321F2E80	0536D538	CCDAA3D9	

4.4.2 Addition in $\text{GF}(2^m)$

Addition is the simplest operation in $\text{GF}(2^m)$. It is simply a bit-wise exclusive-or (xor (\oplus)) in either hardware or software. Addition in \mathbb{F}_2^m can be achieved as shown in (4.4) [41]:

$$Z(x) = U(x) + V(x) = \sum_{i=0}^{m-1} u_i x^i + \sum_{i=0}^{m-1} v_i x^i = \sum_{i=0}^{m-1} (u_i + v_i) x^i = \sum_{i=0}^{m-1} z_i x^i \quad (4.4)$$

where $z_i = (u_i + v_i) \bmod 2 = u_i \oplus v_i$. The subtraction operation in $\text{GF}(2^m)$ is the same as addition because the additive inverse of an element is its identity : $U(x) + U(x) = 0$.

For example, if $U = 1100_2$ and $V = 0110_2$ over the finite field $\text{GF}(2^4)$ then $Z = U + V = U \oplus V = (1100_2 \oplus 0110_2) = 1010_2$.

4.4.3 Multiplication in $\text{GF}(2^m)$

Polynomial multiplication or multiplication in $\text{GF}(2^m)$ with the interleaved modular reduction algorithm is a well-known algorithm for hardware implementation [41]. It computes the product of two polynomials then applies modular reduction, and its operation

is different from simple integer multiplication. Multiplication in \mathbb{F}_2^m can be achieved as shown in (4.5):

$$Z(x) = U(x).V(x) = U(x). \sum_{i=0}^{m-1} v_i.x^i = \sum_{i=0}^{m-1} (U(x).v_i).x^i \quad (4.5)$$

Multiplication by x^i can easily be calculated by the binary left-shift operation. From polynomial multiplication in algorithm 4.1, we check whether the result is an element of $\text{GF}(2^m)$ with degree $< m$. A modular reduction step is only necessary if the polynomial multiplication result Z_v has degree m or higher. This condition is checked by the $Z_v(m) = 1$ command. The result of polynomial multiplication $Z(x) = U(x).V(x) \bmod f(x)$, is

Algorithm 4.1: Multiplication in $\text{GF}(2^m)$ with interleaved modular reduction

Input: $U(x), V(x) \in \text{GF}(2^m)$, irreducible polynomials of degree m

Output: $Z(x) = U(x) . V(x) \bmod f(x)$

$Z_v = 0$; $U_v = '0'$ & $U(x)$;

for $i = m - 1$ **to** 0 **do**

if $V(i) = '1'$ **then** $Z_v = Z_v . x + U_v$; **else** $Z_v = Z_v . x$; **end if**

if $Z_v(m) = '1'$ **then** $Z_v = Z_v + f(x)$; **end if**

end for

Return ($Z(x) = Z_v(m-1 \text{ downto } 0)$) (At this instance,

$Z(x)$ is the result of $U(x) . V(x) \bmod f(x)$)

achieved after m iterations. Algorithm 4.1 [41], named multiplication (Mult.) in $\text{GF}(2^m)$ with interleaved modular reduction, takes just four steps to find the solution of polynomial multiplication over $\text{GF}(2^4)$. The polynomial multiplication result should be reduced to a degree < 4 by irreducible polynomial $f(x) = x^4 + x + 1$.

4.4.4 Squaring in $\text{GF}(2^m)$

A PB squarer is simpler than and closely related to multiplication. But squaring in $\text{GF}(2^m)$ has less difficulty than polynomial multiplication because $U(x)^2 \bmod f(x)$ is a linear operation. It can be computed as shown in (4.6):

$$Z(x) = U(x)^2 = u_{m-1}.x^{2m-2} + \dots + u_2.x^4 + u_1.x^2 + u_0 = \sum_{i=0}^{m-1} u_i x^{2i} \quad (4.6)$$

The squaring operation in $\text{GF}(2^m)$ of $Z(x) = U(x)^2$ is achieved by setting a 0 bit between consecutive bits of the binary representation of $U(x)$ as shown in Fig. 4.1 [2, 41].

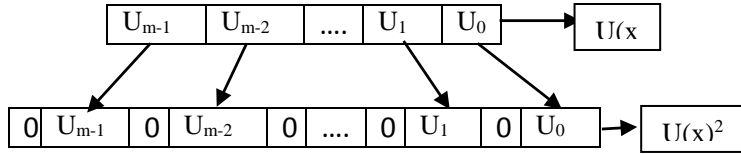


Figure 4.1: Squaring a binary polynomial $U(x)$.

4.4.5 Inversion in $\text{GF}(2^m)$

Inversion in $\text{GF}(2^m)$ is the most expensive operation for implementing ECC over a binary field. Algorithm 4.2 computes the field inversion of a non-zero field element $U(x) \in \mathbb{F}_2^m$ using the modified Euclidean algorithm [52]. We used this inversion algorithm for our hardware implementation because it is easy to implement on a FPGA. The result of field inversion $Z(x) = 1/U(x) \bmod f(x)$ or multiplicative inversion of $U(x)$ is achieved after $2m$ iterations ($i = 1$ to $2m$) and the value of cnt is always equal to zero at the end of the last iteration [52].

Algorithm 4.2: Inversion in $\text{GF}(2^m)$ with Modified Euclidean Algorithm

Input: $U(x) \in \text{GF}(2^m)$, irreducible polynomial of degree m
Output: $Z(x) = 1/U(x) \bmod f(x)$

```

 $P_v = '0'$  &  $U(x)$  ;  $Q_v = f(x)$ ;  $Z_v = 00001$ ;  $V = 0$  ;  $cnt = 0$  ;
for  $i = 1$  to  $2m$  do
    if  $P_v(m) = '0'$  then  $P_v = x \cdot P_v$  ;  $Z_v = x \cdot Z_v$  ;
        if  $Z_v(m) = '1'$  then  $Z_v = Z_v + f(x)$  ; end if
         $cnt = cnt + 1$  ;
    else
        if  $Q_v(m) = '1'$  then  $Q_v = Q_v + P_v$  ;  $V = V + Z_v \bmod f(x)$  ; end if
         $Q_v = x \cdot Q_v$  ;
        if  $cnt = 0$  then
             $P_v = Q_v$  ;  $Q_v = P_v$  ; ( $P_v \leftrightarrow Q_v$ )
             $Z_v = V$  ;  $V = Z_v$  ; ( $Z_v \leftrightarrow V$ , exchange operations)
             $Z_v = x \cdot Z_v \bmod f(x)$ ;  $cnt = cnt + 1$  ;
        else
             $Z_v = Z_v/x \bmod f(x)$  ;  $cnt = cnt - 1$  ;
        end if
    end if
end for

Return ( $Z(x) = Z_v(m-1 \text{ downto } 0)$ ) ( $Z(x)$  is the result of  $1/U(x) \bmod f(x)$ )

```

4.4.6 Proposed EC Group Operations

The elliptic curve group operations in $\text{GF}(2^m)$ are the PD and PA operations. These are the building blocks of finite-field arithmetic operations such as addition, multiplication, squaring and inversion. Figs. 4.2 and 4.3 show the data-flow architecture of the proposed ECPD and ECPA operations, corresponding to (4.2) and (4.3) respectively. The ECPD

operation in affine coordinates requires one field inversion, five field additions, two field multiplications, and two field squarings. Similarly, the ECPA operation in affine coordinates requires one field inversion, eight field additions, two field multiplications, and one field squaring.

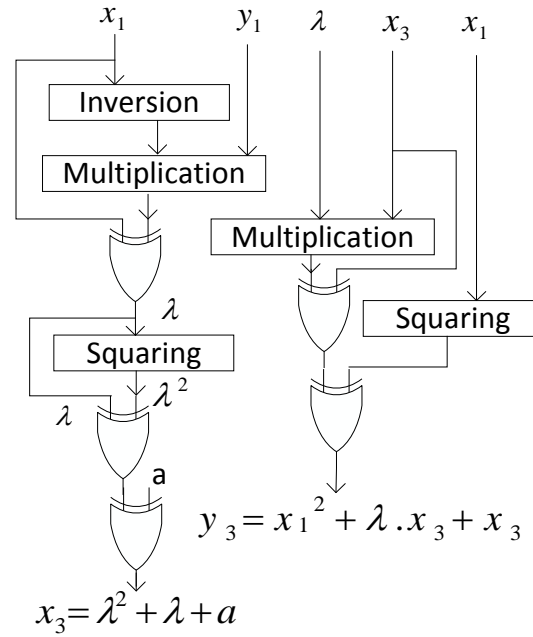


Figure 4.2: Hardware architecture of the elliptic curve point doubling (ECPD) operation.

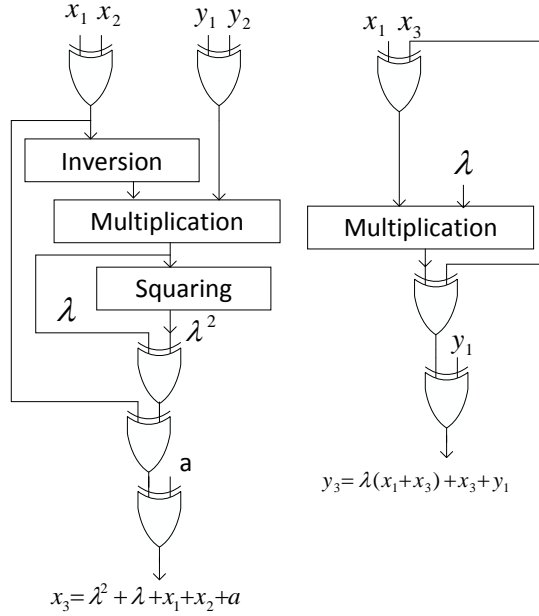


Figure 4.3: Hardware architecture of the elliptic curve point addition (ECPA) operation.

4.5 Proposed ECPM

Elliptic curve point multiplication (ECPM) is the main operation of an ECC processor; it is computationally the most expensive. However, we have designed a high-performance ECPM using efficient group operations and FFMA units. The building block of an elliptic curve cryptosystem contains ECC protocols such as ECDH (elliptic curve Diffie-Hellman) key exchange, ECDSA (EC digital signature algorithm) at the top level, point multiplication in the second level, group operations in the third level, and field arithmetic operations in the bottom level. The basic operation of ECPM is defined as kP , where k is a positive integer and P is a point on the elliptic curve E defined over a field \mathbb{F}_2^m . The proposed ECPM architecture over $\text{GF}(2^m)$ is presented in Fig. 4.4. Various methods exist for implementing ECPM: the binary method, the Non-adjacent form (NAF) method, and the Montgomery method. The easiest way to implement ECPM is the binary method (left to right) [2]. Finally, we present the ECPM Algorithm 4.3 using the binary method. It is

implemented using the “Double-and-Add” algorithm concept.

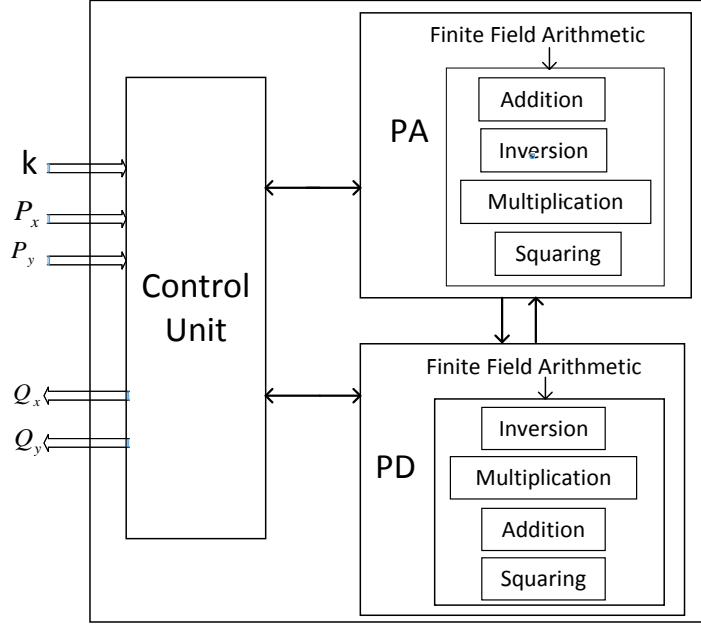


Figure 4.4: Hardware architecture of Elliptic Curve Point Multiplication (ECPM) processor.

Algorithm 4.3: Binary method (Left to right) for point multiplication

Input: $k = (k_{m-1}, \dots, k_1, k_0)_2$, $P(x, y) \in E(\mathbb{F}_2^m)$

Output: $Q(x, y) = k.P(x, y)$, where $Q(x, y), P(x, y) \in E(\mathbb{F}_2^m)$

```

 $Q = 0$  ;
for  $i = m - 1$  to  $0$  do
     $Q = 2Q$ ;
    if  $k(i) = '1'$  then  $Q = Q + P$  ; end if
end for
Return ( $Q(x, y)$ )
    
```

4.6 FPGA Implementation Results and Performance Analysis

This section presents the hardware implementation results of this design. We have implemented and tested our design on a modern 28-nm Xilinx Kintex-7 (XC7K325T-2FFG900) FPGA. All VHDL modules are extensively simulated using both Isim and ModelSim, and synthesized using Xilinx ISE 14.7 synthesis technologies.

Table 4.3: *Synthesis Results of the finite-field arithmetic for $GF(2^{163})$ in Kintex-7*

Arithmetic Opn	FF	LUTS	LUT-FF Pairs	CC	Freq. (MHz)	Time (μs)
Mult./SQ	335	385	335	163	388.83	0.419
Inversion	1479	2007	1315	327	431.71	0.757

Table 4.4: *Elliptic curve Group Operation Results for $GF(2^m)$ in Kintex-7.*

Group Opn	FF	LUTs	LUT-FF Pairs	CC	Frequency (MHz)	Time (μs)
PD	3484	4264	3037	1636	331.76	4.930
PA	6587	8347	5664	1636	331.58	4.934

Table 4.3 depicts the synthesis results of the finite-field arithmetic operations such as field multiplication/squaring and field inversion over $GF(2^{163})$. Multiplication or squaring over $GF(2^{163})$ takes the same area (FF and LUTS), the same number of clock cycles and the same computation time. On the other hand, the clock cycles, flip-flops (FFs), and LUTs (look-up tables) ratio of inversion to multiplications are about 2, 4.45, and 5.2 respectively. Only inversion consumes more clock cycles, area, and timing. The multiplication/squaring (SQ) over $GF(2^{163})$ is performed in Xilinx Kintex-7 in 419 ns but inversion takes 757 ns. From our implementation results, we notice that field inversion is the most time-consuming operation over the binary field because an inversion takes the

same number of clock cycles as 2 multiplications.

The hardware implementation results of proposed elliptic curve group operations are presented in Table 4.4. The major building block of the elliptic curve group operations (PD and PA) contains addition, multiplication, squaring and inversion. These operations were defined over the binary finite field $\text{GF}(2^m)$. The PA operation occupies almost double the area of the PD operation, but the number of clock cycles and the computation time are identical for both operations. The ECPM results for the NIST-recommended field ($\text{GF}(2^{163})$) is shown in Table 4.6. We achieve a point multiplication in 1.06 ms at a frequency of 306.48 MHz in Xilinx Kintex-7 (XC7K325T-2FFG900) FPGA.

Table 4.5: *Synthesis results for elliptic curve point multiplication (ECPM) over $\mathbb{F}_{2^{163}}$.*

Slice Logic Utilization	Used	Available	Used (%)
Numbers of Slice Registers	6620	407600	1
Number of Slice LUTs	7963	203800	3
Numbers of Fully Used LUT-FF Pairs	5712	8871	66
Numbers of BUFG/BUFGCTRLs	2	32	6
Numbers of Bonded IOBs	330	500	66
Numbers of occupied Slices	2253	50950	4

Table 4.5 represents the summary of estimated values of device utilization. The implemented design over the binary field $\mathbb{F}_{2^{163}}$ takes a small amount of resources on the FPGA. The synthesis report shows that our design is area-efficient as it contains only 2253 slices (4% utilization of total available resources).

The hardware implementation results and performance comparisons with related cryptographic processors are listed in Table 4.6, which tries to give all the frequencies, number of clock cycles, and the computation time of the designs to make a fair comparison on the performance between them. An ECC processor over $\text{GF}(2^{163})$ for wireless sensor networks

Table 4.6: *Comparison between our ECC design and related work over $GF(2^{163})$*

References	Technology	Frequency (MHz)	Clock cycles	Time (ms)
This work	Kintex-7	306.48	325564	1.06
Ghanmy [128]	Virtex-II	24	54138	2.26
Reaz [126]	FLEX10KE	43	640700	14.9
Hasan [127]	Spartan-3	76	205200	2.7
Machhout [118]	Virtex-II	167.84	347425	2.07
Shieh [129]	V1000E	-	-	2.55
Park [131]	V1000E	44	134090	3.05
Smyth [130]	0.13 μ mASIC	166	526280	3.17

(WSN) is proposed in [128], and it requires 2.26 ms to achieve a point multiplication. The ECC processor proposed by Reaz [126] provides a result for the field $GF(2^{163})$, and their design takes 14.9 ms to compute a point multiplication. Our implemented result is almost 14 times the speed of Reaz [126] but our presented result is not in the same platform. Hasan [127], Machhout [118], and Sheih [129] implemented ECC processors over $GF(2^{163})$, and their designs require 2.7 ms, 2.07 ms, and 2.55 ms respectively. Our implemented result is almost double the speed of that of Hasan, Maccout, and Sheih. Park [131] and Smyth [130] developed ECC processors over $GF(2^{163})$ in different platforms but their cryptographic processors require more computation time than our design. Our ECC processor over $GF(2^{163})$ takes 1.06 ms to accomplish a point multiplication. We have also achieved a higher frequency than other cryptographic processors. From the comparison and performance analysis in Table 4.6, our ECC processor over $GF(2^{163})$ provides better performance than others.

4.7 Conclusions

A high-performance ECC processor over $\text{GF}(2^{163})$ has been implemented using FPGA technology. The binary method (double-and-add) point-multiplication algorithm using an affine coordinate system was used for this hardware implementation. An efficient polynomial-basis multiplication and inversion algorithm was developed for performing elliptic curve PD and PA operations and hence ECC processor. The implemented design is optimized by using different optimization techniques such as balancing the PD and PA architecture, parallelization in operations, and pre-computations for obtaining high performance on an FPGA compared to other designs. In $\text{GF}(2^{163})$, we can achieve a point multiplication in 1.06 ms at 306.48 MHz in Kintex-7 (28-nm) devices, which is the fastest hardware implementation result. The proposed design provides nearly 50% better delay performance than recent implementations. Our implemented design is also area-efficient as it contains only 2253 slices without using any DSP slices. Based on the overall performance analysis and comparisons of different ECC processors over the binary field \mathbb{F}_{163} , it can be concluded that this design provides better performance than others in terms of the area and the timing.

Chapter 5

High-Speed, Area-Efficient, FPGA-Based Elliptic Curve Cryptographic Processor over NIST Binary Fields¹

5.1 Abstract

In this paper we propose a high-performance FPGA-based implementation of an elliptic curve cryptographic (ECC) processor over binary field $GF(2^m)$ for modern cryptographic applications. A high-speed elliptic curve scalar multiplier (ECSM) is designed using an efficient finite-field arithmetic unit, where ECSM is the main operation of an ECC processor. It has been implemented

¹Published as: Md Selim Hossain, Ehsan Saeedi and Yinan Kong, “High-Speed, Area-Efficient, FPGA-Based Elliptic Curve Cryptographic Processor over NIST Binary Fields,” *2015 IEEE International Conference on Data Science and Data Intensive Systems (DSDIS)*, UTS, Sydney, Australia, pp. 175-181, 11-13 December, 2015, DOI: 10.1109/DSDIS.2015.44.

in an affine coordinate system using a polynomial basis. The implemented design is synthesized in field-programmable gate array (FPGA) technology. The ECSCM time in a modern Xilinx Kintex-7 FPGA is 2.66 ms at 255.66 MHz and 5.54 ms at 251.98 MHz for the field size of $GF(2^{233})$ and $GF(2^{283})$ respectively. Simulation results show that the implemented design is area-efficient, as it contains only 3016 slices for the field $\mathbb{F}_{2^{233}}$ and 4625 slices for the field $\mathbb{F}_{2^{283}}$. To the best of the authors' knowledge, the proposed ECC processor shows better performance than the available hardware implementations.

5.2 Introduction

Public-key cryptography (PKC) and Private-key cryptography are two main families of cryptography used for different wireless security purposes. With the rapid growth of communication systems, the demand for data security in wireless communication and associated appliances has increased rapidly in recent days. For these applications, PKC plays a vital role to pass secured information among the different wireless devices. A cryptographic algorithm such as elliptic curve cryptography should be designed in such a way that it requires minimal available resources with the assurance of high security and throughput. The Ron Rivest, Adi Shamir and Leonard Adleman cryptosystem called the RSA cryptosystem [8] and the Elliptic Curve Cryptosystem (ECC) [10, 11] are the two most popular public-key cryptosystems. ECC is a comparatively new cryptosystem, first proposed in 1985 independently by Neal Koblitz and Victor S. Miller [10, 11]. However, ECC can provide the highest security per bit compared to other traditional public-key cryptosystems such as RSA, DH. ECC typically has a inferior throughput rate and most complex computation. This attractive feature makes ECC very popular for resource-constrained applications such as smart cards, credit cards, pagers, PDAs (Personal Digi-

tal Assistants), and cellular phones [35]. The IEEE has standardized P1363-2000 [23] for the use of ECC-based key-agreement and digital-signature algorithms (DSA). Certicom has provided NIST-recommended elliptic curve domain parameters, standard for efficient cryptography in SEC2 [54]. The U.S. Government organization called the National Institute of Standards and Technology (NIST) recommends elliptic curve parameters for $\text{GF}(p)$ and $\text{GF}(2^m)$ [22]. There are different security organizations such as ISO, NSA and ANSI who also have been working for standardization of the use of ECC. Field-programmable gate-array (FPGA) technology is used for this hardware implementation due to greater flexibility and ability to update the cryptographic algorithm.

In [128], Ghanmy et al. implemented their ECC processor over $\text{GF}(2^{163})$ on a Xilinx Virtex-2 FPGA for wireless sensor networks (WSN). Hasan and Benaissa [127] implemented their ECC processor over $\text{GF}(2^{131})$, $\text{GF}(2^{163})$, $\text{GF}(2^{283})$ and $\text{GF}(2^{571})$ using the μ -coding technique on a Xilinx Spartan-3 FPGA. A coupled FPGA/ASIC implementation of an EC crypto-processor over $\text{GF}(2^{163})$ is presented in [118], using both FPGA and ASIC as a hardware platform. Wang [134], Smyth [130], Zeng [135], and Nguyen [136] et al. also implemented their ECC processor over a binary field. The optimization aim is generally to reduce the latency of an elliptic curve scalar multiplication. For this, we have concentrated on efficient algorithms to improve finite-field arithmetic operations using repeating blocks [35]. A dedicated hardware implementation of an ECC processor is required for real-time applications to speed up the overall computation of cryptosystems.

To the best of our knowledge, there have been few high-performance hardware implementations of an ECC processor in the literature. Thus a fast, high-performance implementation of an ECC processor is still needed for modern cryptographic applications. In this paper, efficient algorithmic reformulations underlying binary finite field and architectural optimization schemes are explored to improve the operating speed [132, 133]. An FPGA-based hardware implementation for an ECC processor over $\text{GF}(2^{233})$ and $\text{GF}(2^{283})$

is presented and is the fastest implementation in an affine coordinate system.

This paper is organized as follows. Section 5.3 describes a background of groups and fields, Galois finite fields, and ECC theories related to this work. Section 5.4 presents an efficient finite-field algorithm over $\text{GF}(2^m)$, EC group operations (PD and PA) and hardware architectures. An ECSM algorithm and cryptographic processor are given in Section 5.5. FPGA implementation results and comparisons with related designs are given in Section 5.6. Finally, Section 5.7 summarizes our work.

5.3 Background

In this section, a brief introduction to the Galois field, mathematics and theories of ECC used in this hardware implementation is presented. The implementation hierarchy of the ECC operations over the binary field $\text{GF}(2^m)$ is presented in Fig. 5.1. From this figure, elliptic curve cryptographic schemes such as ECDSA and ECDH are the building blocks of ECSM and elliptic curve group operations (ECPA and ECPD). These are the series of finite-field arithmetic operations such as field addition, subtraction, multiplication, squaring, and inversion. Finite-field arithmetic units are most crucial for the overall performance of an ECC processor.

5.3.1 Finite Field

If the field consists of a finite number of elements, it is called a finite field or Galois field (GF). It is a set of elements denoted normally as $\text{GF}(q = p^m)$ where p is a prime number called the characteristic of \mathbb{F} , and m is a positive integer. The field is said to be a prime field if $m = 1$ and an extension field [34] if m is greater than 2. A Galois field is said to be a binary field or characteristic-two finite field if $q = 2^m$. A binary field is quite simple for hardware implementation using more-efficient modulo-2 arithmetic. This can

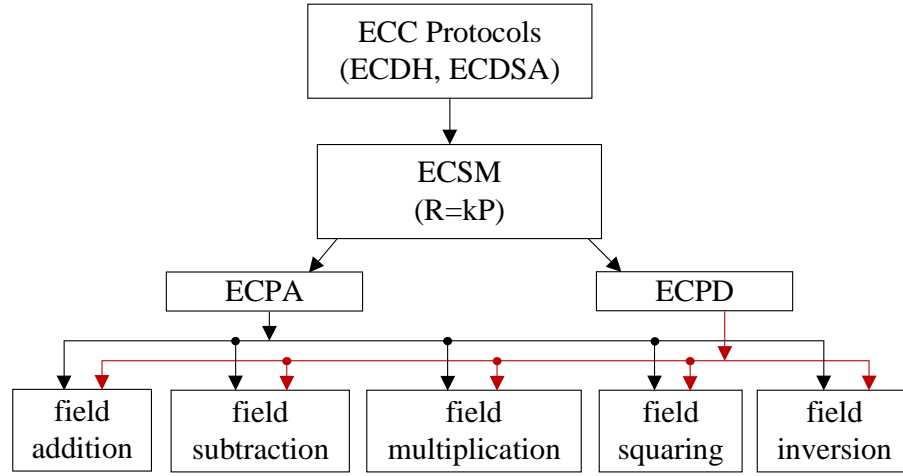


Figure 5.1: *Implementation hierarchy of the ECC operations over $GF(2^m)$.*

be represented using an optimal normal basis (ONB) and a polynomial basis (PB). A PB is beneficial for hardware implementation, as a field element can be characterized by m binary bits. All hardware architecture for finite-field operations is implemented modulo a degree m irreducible polynomial $f(x)$. Irreducibility means $f(x)$ cannot be factored as a product of binary polynomials each of degree $< m$. In Section 5.4, all mathematical operations, algorithms, and hardware architectures over $GF(2^m)$ will be discussed [2, 35].

5.3.2 Coordinate Systems for Elliptic Curve Point Representation

There are various coordinate systems to represent elliptic curve points, either binary field or prime field, but two well-known coordinate systems are often used for ECC: Affine coordinate systems and projective coordinate systems. A point on the elliptic curve E for affine coordinates can be represented by using two elements $x, y \in \mathbb{F}_2^m$, i.e. $P(x, y)$. In this coordinate system, the elliptic curve group operations like elliptic curve point doubling and elliptic curve point addition require a field inversion, a time-consuming operation. The field inversion over a binary field for each group operation can be reduced

by using projective coordinate systems. But we have implemented a high-performance modular inversion that is well suited for an ECC processor. There is plenty on projective coordinates in the available literature; a detailed coordinate system is discussed in [2].

5.3.3 Elliptic Curve Cryptography

Elliptic curve cryptography (ECC) is performed in either prime fields $GF(p)$ or binary fields $GF(2^m)$. But EC over $GF(2^m)$ will be the emphasis of this work because it is very efficient for hardware implementation due to the use of modulo-2 arithmetic. An elliptic curve defined over a finite field provides a group structure that is used to implement the cryptographic systems. The group operations are elliptic curve point addition (ECPA) and elliptic curve point doubling (ECPD). There have been different coordinate systems to represent elliptic curve points. They vary in the number and type of field operations required to implement PA/PD. In our work, we implement all elliptic curve operations in an affine coordinate system. A non-supersingular elliptic curve E over $GF(2^m)$ in affine coordinates is the set of solutions to the equation

$$y^2 + xy = x^3 + ax^2 + b \tag{5.1}$$

where $x, y, a, b \in GF(2^m), b \neq 0$. The coefficients $a, b \in \mathbb{F}_2^m$ specifying an elliptic curve $E(\mathbb{F}_2^m)$ are defined by (5.1). The number of points on an elliptic curve E is represented by $\#E(\mathbb{F}_2^m)$. It is defined over \mathbb{F}_2^m as nh , where n is the prime order of the curve and h is an integer called the co-factor.

If $P = (x_1, y_1) \in E$ and $Q = (x_2, y_2) \in E$ (points on the EC), then summing PA and PD can be respectively derived as

$$R(x_3, y_3) = P(x_1, y_1) + Q(x_2, y_2) \in E,$$

$$x_3 = \lambda_1^2 + \lambda_1 + x_1 + x_2 + a, \quad (5.2)$$

$$y_3 = \lambda_1(x_1 + x_3) + x_3 + y_1,$$

where $\lambda_1 = (y_2 + y_1)/(x_2 + x_1)$ and $P \neq Q$;

$$R(x_3, y_3) = 2P(x_1, y_1) \in E,$$

$$x_3 = \lambda^2 + \lambda + a, \quad (5.3)$$

$$y_3 = x_1^2 + \lambda x_3 + x_3,$$

where $\lambda = x_1 + y_1/x_1$ and $P = Q$;

where $R = 0$ when $x_1 = x_2$ and $y_2 \neq y_1$, or $x_1 = x_2 = 0$. Hence, when $P \neq Q$ we have the PA operation in (5.2) and when $P = Q$ we have the PD operation in (5.3). Using these operations, ECSM kP will be implemented using an ECC-based algorithm [2, 10, 11, 35].

Table 5.1: *comparison of Key length for equivalent security of Symmetric-key and public-key Cryptography [2, 33]*

Symmetric key	Example algorithm	RSA/DH	ECC in $\text{GF}(2^m)$
80	SKIPJACK	1024	163
112	Triple-DES	2048	233
128	AES Small	3072	283
192	AES Medium	8192	409
256	AES Large	15360	571

In 2000, FIPS-2 was recommended with 10 finite fields: 5 prime fields, and 5 binary fields. The binary fields are \mathbb{F}_2^{163} , \mathbb{F}_2^{233} , \mathbb{F}_2^{283} , \mathbb{F}_2^{409} and \mathbb{F}_2^{571} [22]. Both prime fields $\text{GF}(p)$ and $\text{GF}(2^m)$ are considered to provide almost the same level of security [1]. Table 5.1 compares symmetric cipher key length, and key lengths for public-key cryptography like

RSA, Diffie-Hellman (DH), and ECC in a binary field. It demonstrates that smaller field sizes can be used in ECC than in RSA and DH systems at a given security level. For instance, 283-bit ECC gives equivalent security to 3072-bit RSA with significantly smaller keys and area. This makes ECC a promising branch of public-key cryptography [2, 33].

5.4 Implementation of Finite-Field Arithmetic

Finite-field arithmetic units are most important for overall performance of an ECC processor. The proposed hardware architecture, which is important for an ECC processor, was implemented by using NIST-recommended binary fields $\text{GF}(2^{233})$ and $\text{GF}(2^{283})$. All parameters for NIST-recommended elliptic curves over $\text{GF}(2^{233})$ and $\text{GF}(2^{283})$ are listed in Table 5.2. A modern Xilinx Kintex-7 (XC7K325T-2FFG900) FPGA with VHDL as hardware language is used for our hardware implementation. The main components in this ECC design are: polynomial-basis (PB) modular addition or field addition, field multiplication, field squaring, field inversion, and group operations.

5.4.1 Polynomial Basis Representation

A polynomial basis is a very popular extension field used to represent field elements. It is used in our hardware design for the representation of numbers. For the PB representation, the elements \mathbb{F}_2^m are binary polynomials of degree at most $m - 1$, i.e.

$$\begin{aligned}\mathbb{F}_2^m &= u_{m-1}.x^{m-1} + u_{m-2}.x^{m-2} + \cdots + u_2.x^2 + u_1.x + u_0 \\ &= \sum_{i=0}^{m-1} u_i x^i : u_i \in \{0, 1\}\end{aligned}$$

For instance, $x^3 + x + 1$ is a polynomial-basis representation of the 4-bit number 1011_2 . For a reduction polynomial or irreducible polynomial, let $f(x)$ be an irreducible binary polynomial of degree m , and $f(x) = x^m + G(x) = x^m + \sum_{i=0}^{m-1} g_i x^i$ where $g_i \in \{0, 1\}$ for

Table 5.2: *NIST-recommended elliptic curves over \mathbb{F}_2^{233} and \mathbb{F}_2^{283} [22]*

B-233:	$m = 233$,	$f(x) = x^{233} + x^{74} + 1$,	$a = 1$,	$h = 2$
S=0x	74D59FF0	7F6B413D	0EA14B34	4B20A2DB 049B50C3
b=0x	00000066	647EDE6C	332C7F8C	0923BB58 213B333B
	20E9CE42	81FE115F	7D8F90AD	
n=0x	00000100	00000000	00000000	00000000 0013E974
	E72F8A69	22031D26	03CFE0D7	
x=0x	000000FA	C9DFCBAC	8313BB21	39F1BB75 5FEF65BC
	391F8B36	F8F8EB73	71FD558B	
y=0x	00000100	6A08A419	03350678	E58528BE BF8A0BEF
	F867A7CA	36716F7E	01F81052	

B-283:	$m = 283$,	$f(x) = x^{283} + x^{12} + x^7 + x^5 + 1$,	$a = 1$,	$h = 2$
S=0x	77E2B073	70EB0F83	2A6DD5B6	2DFC88CD 06BB84BE
b=0x	027B680A	C8B8596D	A5A4AF8A	19A0303F CA97FD76
	45309FA2	A581485A	F6263E31	3B79A2F5
n=0x	03FFFFFF	FFFFFFFF	FFFFFFFF	FFFFFFFF FFFFEF90
	399660FC	938A9016	5B042A7C	EFADB307
x=0x	05F93925	8DB7DD90	E1934F8C	70B0DFEC 2EED25B8
	557EAC9C	80E2E198	F8CDBECD	86B12053
y=0x	03676854	FE24141C	B98FE6D4	B20D02B4 516FF702
	350EDDB0	826779C8	13F0DF45	BE8112F4

$i = 1, \dots, m - 1$ and $g_0 = 1$ [2, 40]. For example,

$$\left\{ f(x) = x^4 + x + 1 = (10011)_2 \right.$$

is an irreducible polynomial of the finite field $\text{GF}(2^4)$.

5.4.2 Addition in $\text{GF}(2^m)$

Field addition is the simplest operation in binary field $\text{GF}(2^m)$. It is simply a bit-wise exclusive-or (xor (\oplus)) in either hardware or software. Addition in \mathbb{F}_2^m can be achieved as shown in (5.4) [41]:

$$Z(x) = U(x) + V(x) = \sum_{i=0}^{m-1} u_i x^i + \sum_{i=0}^{m-1} v_i x^i = \sum_{i=0}^{m-1} (u_i + v_i) x^i = \sum_{i=0}^{m-1} z_i x^i \quad (5.4)$$

where $z_i = (u_i + v_i) \bmod 2 = u_i \oplus v_i$. The subtraction operation in $\text{GF}(2^m)$ is the same as addition because the additive inverse of an element is its identity : $U(x) + U(x) = 0$.

For example, if

$$\begin{cases} U = 1100_2, \text{ and } V = 0110_2 \text{ over the field } \text{GF}(2^4) \\ \text{then } Z = (U + V) = (U \oplus V) = (1100_2 \oplus 0110_2) = 1010_2. \end{cases}$$

5.4.3 Multiplication in $\text{GF}(2^m)$

Polynomial multiplication or field multiplication in $\text{GF}(2^m)$ with the interleaved modular reduction algorithm, which is a well-known algorithm for hardware implementation [41]. It computes the product of two polynomials then applies modular reduction, and its operation is different from simple integer multiplication. Multiplication in \mathbb{F}_2^m can be achieved as shown in (5.5):

$$Z(x) = U(x).V(x) = U(x). \sum_{i=0}^{m-1} v_i . x^i = \sum_{i=0}^{m-1} (U(x).v_i) . x^i \quad (5.5)$$

Multiplication by x^i can easily be done with a binary left-shift operation. Polynomial multiplication in algorithm 5.1, we check whether the result is an element of $\text{GF}(2^m)$ with degree $< m$. A modular reduction step is only necessary if the polynomial multiplication result Z_v has degree m . This condition is checked by the $Z_v(m) = 1$ command. The result of polynomial multiplication, $Z(x) = U(x).V(x) \bmod f(x)$, is achieved after m iterations. Algorithm 5.1 [41], named multiplication (Mult.) in $\text{GF}(2^m)$ with interleaved

Algorithm 5.1: Multiplication in $\text{GF}(2^m)$ with interleaved modular reduction

Input: $U(x), V(x) \in \text{GF}(2^m)$, irreducible polynomials of degree m

Output: $Z(x) = U(x) \cdot V(x) \bmod f(x)$

$Z_v = 0$; $U_v = '0'$ & $U(x)$;

for $i = m - 1$ **to** 0 **do**

if $V(i) = '1'$ **then** $Z_v = Z_v \cdot x + U_v$; **else** $Z_v = Z_v \cdot x$; **end**

if $Z_v(m) = '1'$ **then** $Z_v = Z_v + f(x)$; **else** $Z_v = Z_v$; **end**

end for

Return ($Z(x) = Z_v(m-1 \text{ downto } 0)$) (At this instant,

$Z(x)$ is the result of $U(x) \cdot V(x) \bmod f(x)$)

Table 5.3: *Example of Computing Multiplication in $\text{GF}(2^4)$ Based on Algorithm 5.1*

($f(x) = x^4 + x + 1 = 10011$, $U(x) = x^3 + x^2 + x + 1 = 1111$, $V(x) = x^3 + x^2 + x = 1110$)

i	U_v	Z_v	Z
	01111	00000	0000
3	01111	01111	1111
2	01111	00010	0010
1	01111	01011	1011
0	01111	00101	0101

modular reduction, takes just four steps to perform polynomial multiplication over $\text{GF}(2^4)$.

The detailed operation of a step-by-step solution is shown in Table 5.3. As an example of polynomial multiplication over $\text{GF}(2^4)$, assume that $f(x) = x^4 + x + 1 = (10011)_2$,

$U(x) = x^3 + x^2 + x + 1 = (1111)_2$, and $V(x) = x^3 + x^2 + x = (1110)_2$, then

$$\left\{ \begin{array}{l} Z(x) = U(x).V(x) = (x^3 + x^2 + x + 1).(x^3 + x^2 + x) \\ \quad = x^6 + 2.x^5 + 3.x^4 + 3.x^3 + 2.x^2 + x \\ \quad = x^6 + x^4 + x^3 + x \text{ (applying mod 2 operation)} \\ \quad = (x^2 + 1)(x^4 + x + 1) + x^2 + 1 = x^2 + 1 = (0101)_2. \end{array} \right.$$

The polynomial multiplication result should be reduced to a degree < 4 by the irreducible polynomial $f(x) = x^4 + x + 1$.

5.4.4 Squaring in $\text{GF}(2^m)$

PB squaring is simpler than and closely related to multiplication. But squaring in $\text{GF}(2^m)$ has less difficulty than polynomial multiplication because $U^2(x) \bmod f(x)$ is a linear operation. It can be computed as shown in (5.6).

$$Z(x) = U^2(x) = u_{m-1}.x^{2m-2} + \dots + u_2.x^4 + u_1.x^2 + u_0 = \sum_{i=0}^{m-1} u_i x^{2i} \quad (5.6)$$

The squaring operation in $\text{GF}(2^m)$ of $Z(x) = U^2(x)$ is achieved by setting a 0 bit between consecutive bits of the binary representation of $U(x)$ as shown in Fig. 5.2 [2,40,41]. A basic example of polynomial squaring is as follows

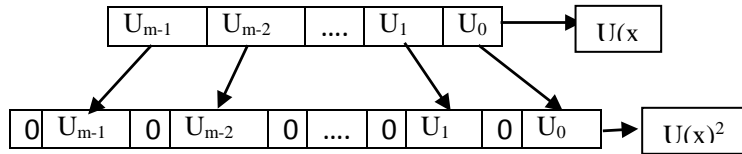


Figure 5.2: Squaring a binary polynomial $U(x)$.

$$\left\{ \begin{array}{l} f(x) = x^4 + x + 1, \quad U(x) = x^3 + x^2 + x + 1, \text{ then} \\ Z(x) = U^2(x) \bmod f(x) = (x^3 + x^2 + x + 1).(x^3 + x^2 + x) \\ \quad = (x^3 + x^2 + x + 1).(x^3 + x^2 + x + 1) = (x^3 + x) = (1010)_2. \end{array} \right.$$

5.4.5 Inversion in $\text{GF}(2^m)$

Inversion in $\text{GF}(2^m)$ is the most expensive operation for implementing ECC over a binary field. Algorithm 5.2 computes the field inversion of a non-zero field element $U(x) \in \mathbb{F}_2^m$ using the modified Euclidean algorithm [52]. We used this inversion algorithm for our hardware implementation because it is easy to implement on a FPGA. The result of field

Algorithm 5.2: Inversion in $\text{GF}(2^m)$ with Modified Euclidean Algorithm

Input: $U(x) \in \text{GF}(2^m)$, irreducible polynomial of degree m

Output: $Z(x) = 1/U(x) \bmod f(x)$

```

 $P_v = '0' \ \& \ U(x) ; Q_v = f(x) ; Z_v = 00001 ; V = 0 ; cnt = 0 ;$ 
for  $i = 1$  to  $2m$  do
    if  $P_v(m) = '0'$  then  $P_v = x \cdot P_v ; Z_v = x \cdot Z_v ;$ 
    if  $Z_v(m) = '1'$  then  $Z_v = Z_v + f(x) ;$  end
     $cnt = cnt + 1 ;$ 
else
    if  $Q_v(m) = '1'$  then  $Q_v = Q_v + P_v ; V = V + Z_v \bmod f(x) ;$  end
     $Q_v = x \cdot Q_v ;$ 
    if  $cnt = 0$  then
         $P_v = Q_v ; Q_v = P_v ; (P_v \leftrightarrow Q_v)$ 
         $Z_v = V ; V = Z_v ; (Z_v \leftrightarrow V, \text{exchange operations})$ 
         $Z_v = x \cdot Z_v \bmod f(x) ; cnt = cnt + 1 ;$ 
    else
         $Z_v = Z_v / x \bmod f(x) ; cnt = cnt - 1 ;$ 
    end
end
end for

Return  $(Z(x) = Z_v(m-1 \text{ downto } 0))$  ( $Z(x)$  is the result of  $1/U(x) \bmod f(x)$ )

```

Table 5.4: *Example of Computing Inversion in $GF(2^4)$ Based on Algorithm 5.2 ($f(x) = x^4 + x + 1 = 10011$, $U(x) = x^3 + x^2 + x + 1 = 1111$)*

i	cnt	P_v	Q_v	V	Z_v	Z
	0	01111	10011	00000	00001	0001
1	1	11110	10011	00000	00010	0010
2	0	11110	11010	00010	00001	0001
3	1	01000	11110	00001	00110	0110
4	2	10000	11110	00001	01100	1100
5	1	10000	11100	01101	00110	0110
6	0	10000	11000	01101	00011	0011
7	1	10000	10000	00011	00011	0011
8	0	10000	00000	00000	01000	1000

inversion $Z(x) = 1/U(x) \bmod f(x)$ or multiplicative inversion of $U(x)$ is achieved after $2m$ iterations ($i = 1$ to $2m$) and the value of cnt is always equal to zero at the end of the last iteration [52]. The detailed computation steps are shown in Table 5.4. For example, assume

$$\left\{ \begin{array}{l} f(x) = x^4 + x + 1 = (10011)_2, \\ U(x) = x^3 + x^2 + x + 1 = (1111)_2, \text{ for inversion over } GF(2^4), \text{ then} \\ Z(x) = Z(x) = 1/U(x) \bmod f(x) \\ \quad = 1/(1111)_2 = 1/g^{12} = g^{-12} = g^{(15-12)} = g^3 = x^3 = (1000)_2. \end{array} \right.$$

To check that this is the multiplicative inverse of $U(x)$ their multiplicative identity should be one, e.g.

$$\left\{ \begin{array}{l} Z(x) = 1/U(x) \bmod f(x) \\ \quad = 1/(1111)_2 = 1/g^{12} = g^{-12} = g^{(15-12)} \\ \quad = (x^3 + x^2 + x + 1)x^3 = x^6 + x^5 + x^4 + x^3 = (x^4 + x + 1)(x^2 + x + 1) + 1 = 1. \end{array} \right.$$

5.4.6 Elliptic Curve Group Operations (ECPD and ECPA)

The elliptic curve group operations in $\text{GF}(2^m)$ are the PD and PA operations. These are the building blocks of finite-field arithmetic operations such as field addition, multiplication, squaring and inversion. Fig. 5.3 shows the data-flow architecture of the ECPD and ECPA operations, corresponding to (5.2) and (5.3) respectively. The ECPD operation in affine coordinates requires one inversion, five additions, two multiplications, and two squarings. Similarly, the ECPA operation in affine coordinates requires one inversion, eight additions, two multiplications, and one squaring.

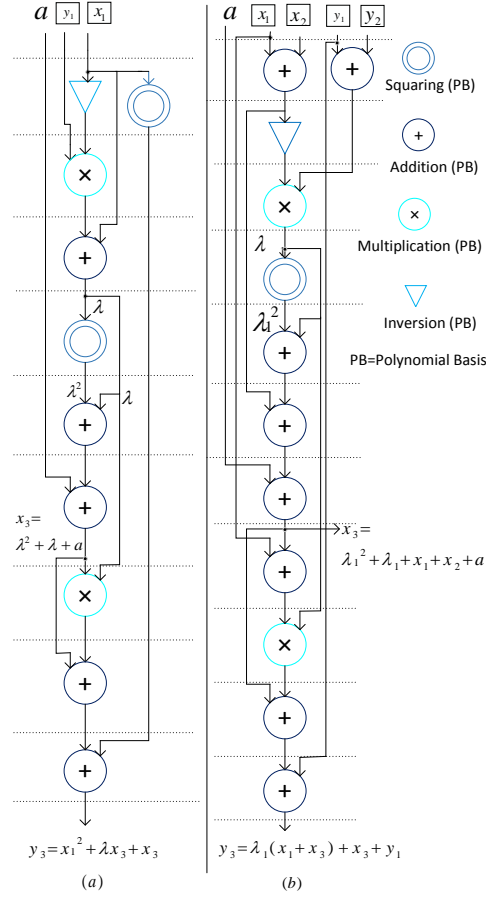


Figure 5.3: Proposed hardware architecture of the elliptic curve (a) point doubling (PD) and (b) point addition (PA).

5.5 Elliptic Curve Scalar Multiplication (ECSM)

ECSM is the main operation of an ECC processor; it is computationally the most expensive operation. The overall performance of an ECC processor rely on ECSM. The detailed implementation hierarchy of the ECSM over the binary field $\text{GF}(2^m)$ is presented in Fig. 5.1. The building block of an elliptic curve cryptosystem contains ECC protocols like ECDH key exchange, ECDSA at the top level, point multiplication in the second level, group operations in the third level, and field arithmetic operations in the bottom level. The ECSM is the building block of ECPD and ECPA. These are the building blocks of

finite-field arithmetic such as field addition, field multiplication, field squaring, and field inversion or division. The basic operation of ECSM is defined as kP , where k is a positive integer and P is a point on the elliptic curve E defined over a field \mathbb{F}_2^m . An ECC processor architecture over $\text{GF}(2^m)$ is presented in Fig. 5.4. A control unit is designed for interconnecting between point addition and point doubling units. There are different methods to implement an elliptic curve scalar multiplication: the binary method, the Non-adjacent form (NAF) method, and the Montgomery method. The easiest way to implement ECC is the binary method (left to right). Finally, we present the EC point multiplication algorithm using the binary method. It is implemented using the "Double-and-Add" algorithm concept.

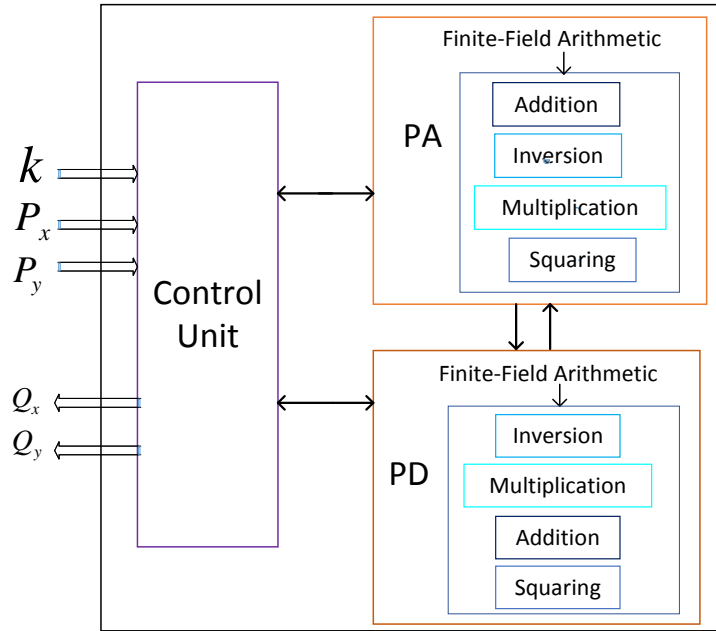


Figure 5.4: Hardware architecture of elliptic curve scalar multiplier (ECSM).

Algorithm 5.3: Binary method (Left to right) for point multiplication

Input: $k = (k_{m-1}, \dots, k_1, k_0)_2$, $P(x, y) \in E(\mathbb{F}_2^m)$

Output: $Q(x, y) = k.P(x, y)$, where $Q(x, y), P(x, y) \in E(\mathbb{F}_2^m)$

$Q = 0$;

for $i = m - 1$ **to** 0 **do**

$Q = 2Q$;

if $k(i) = '1'$ **then** $Q = Q + P$; **end**

end for

Return $(Q(x, y))$

5.6 Results and Performance Analysis

This section presents the hardware implementation results of this design. We have implemented and tested our design on a Xilinx Kintex-7 (XC7K325T-2FFG900) FPGA device using VHDL. All implemented modules are simulated using ModelSim, and synthesized using Xilinx ISE 14.7 synthesis technologies. All the simulation results are verified using high-level Maple software.

Table 5.5: *Synthesis Results of the finite-field arithmetic for $GF(2^m)$ in Kintex-7*

Arithmetic Opn	$GF(2^m)$	Cycles	FF	LUTs	Freq. (MHz)	Time (μs)
Mult.	233-bit	233	474	540	353.09	0.659
	283-bit	283	575	674	319.52	0.885
Squaring	233-bit	233	474	540	353.09	0.659
	283-bit	283	575	674	319.52	0.886
Inversion	233-bit	467	2111	2856	429.87	1.086
	283-bit	567	2562	3468	413.92	1.369

Table 5.6: *Elliptic Curve Group Operation Results for $GF(2^m)$ in Kintex-7*

Group Opn	$GF(2^m)$	FF	LUTs	LUT-FF Pairs	Freq. (MHz)	Time (μs)
PD	233-bit	4959	6052	4311	305.43	7.65
	283-bit	6011	7491	5230	280.19	10.16
PA	233-bit	4484	5689	3957	340.25	7.65
	283-bit	5436	6976	4746	279.42	10.15

Table 5.5 presents the synthesis results of the finite-field arithmetic operations such as multiplication, squaring and inversion over $GF(2^{233})$ and $GF(2^{283})$. According to Table 5.5, multiplication and squaring take almost identical area (FF and LUTs), the same number of clock cycles and the same computation time. On the other hand, the clock cycles, flip-flops (FFs), and LUTs (look-up tables) ratio of inversion to multiplications are about 2, 4.45, and 5.2 respectively. Only inversion consumes more clock cycles, area, and timing. The multiplication over $GF(2^{233})$ and $GF(2^{283})$ is performed in Xilinx Kintex-7 in $0.659 \mu s$ and $0.885 \mu s$ but inversion takes $1.086 \mu s$ and $1.369 \mu s$ respectively. From our implementation results, we notice that field inversion is the most time-consuming operation over the binary field because an inversion takes the same number of clock cycles as two multiplications.

Table 5.7: *Elliptic Curve Scalar Multiplication Results for different GFs in Kintex-7 (XC7K325T-2FFG900)*

$GF(2^m)$	FF (%)	LUTs (%)	LUT-FF Pairs (%)	Slices (%)	Freq.	Time
233-bit	9407 (2%)	9151 (4%)	7022 (71%)	3016 (6%)	255.66	2.66
283-bit	11419 (2%)	14440 (7%)	10028 (62%)	4625 (9%)	251.98	5.54

The hardware implementation results of elliptic curve group operations are presented

Table 5.8: *Comparison between our ECC design and related work over $GF(2^m)$*

Circuits	Bit Length	Technology	Freq. (MHz)	Cycles	Time (ms)
This work	233	Kintex-7	255.66	679776	2.66
	283	Kintex-7	251.98	1395312	5.54
Ghanmy [128]	163	Virtex-II	24	54138	2.26
Hasan [127]	163	Spartan-3	76	205200	2.7
	283	Spartan-3	76	630800	8.3
Machhout [118]	163	Virtex-II	167.84	347425	2.07
Wang [134]	233	Spartan-3	80	183000	2.29
Smyth [130]	163	0.13 μ mASIC	166	526280	3.17
Zeng [135]	233	0.35 μ mASIC	100	466000	4.66
Nguyen [136]	233	Virtex-II	100	335000	3.35

in Table 5.6. The major building block of the elliptic curve group operations (ECPD and ECPA) depends on finite-field arithmetic units. These operations were defined over the binary finite field $GF(2^m)$. The ECPA operation occupies almost the same area as the ECPD operation, but the number of clock cycles and the computation time are slightly different for both operations. Jacobian coordinate systems can be considered for further improvement because then we can avoid costly field inversions. But one field inversion is still required for converting Jacobian coordinates to an affine coordinate system.

The ECSM results for NIST-recommended fields $GF(2^{233})$ and $GF(2^{283})$ are shown in Table 5.7. We achieve a scalar multiplication in 2.66 ms and 5.528 ms respectively in a Xilinx Kintex-7 FPGA over the mentioned fields.

The hardware implementation results and performance comparisons with related cryptographic processors are listed in Table 5.8, which tries to give all the frequencies and number of clock cycles of the designs to make a fair comparison on the performance be-

tween them. It is to be noted that the results provided in the available literature are implemented on different FPGA and ASIC technologies from our implemented design. In this case, a straightforward comparison is difficult. Ghanmy et al. [128] presented an ECC processor over $\text{GF}(2^{163})$ for WSN, and cryptographic processors require 2.255 ms on a Xilinx Virtex-II FPGA to achieve a ECSM. Hasan and Benaissa [127] implemented an ECC processor over $\text{GF}(2^{163})$ and $\text{GF}(2^{283})$ on a Xilinx Spartan-3 FPGA, but their cryptographic processor requires more computation time than our ECC processor. Machhout [118], and Smyth [130] et al. present an ECC processor over the binary field $\text{GF}(2^{163})$, and the computation time of their 163-bit ECC processor is similar to that of our 233-bit ECC processor. Wang [134], Zeng [135], and Nguyen [136] et al. employed a reconfigurable ECC Co-processor for $\text{GF}(2^{233})$, where they used different hardware platforms for their implementation. We achieve a scalar multiplication over $\text{GF}(2^{233})$ in 2.66 ms which is faster than the results provided in Zeng et al. [135] and Nguyen et al. [136] but very similar to the results presented in Wang et al. [134]. For $\text{GF}(2^{283})$, our implementation is almost 1.5 times the speed of that of Hasan [127] but our presented result is not in the same platform. Our cryptographic processor does not support ECC over $\text{GF}(p)$ because the arithmetic operations are completely different. From the comparison and performance analysis in Table 5.8, our ECC processor over $\text{GF}(2^m)$ provides better performance than others.

5.7 Conclusion

In brief, a fast, high-performance ECC processor over \mathbb{F}_{2^m} on a Xilinx Kintex-7 FPGA is proposed. An efficient polynomial-basis multiplication, squaring, and inversion algorithm was developed for performing elliptic curve group operations efficiently. In $\mathbb{F}_{2^{233}}$, we can achieve a scalar multiplication in 2.66 ms at 255.66 MHz in Kintex-7 devices, which

is the fastest hardware implementation result. Additionally, this paper has provided implementation results of scalar multiplication for $\mathbb{F}_{2^{283}}$ in 5.54 ms, showing the good performance of this hardware. This design on a Xilinx Kintex-7 FPGA can achieve a maximum clock frequency of 251.98 MHz for the field $\mathbb{F}_{2^{283}}$. The design was extensively simulated using ModelSim PE and all the results verified using high-level Maple software. From the overall performance analysis and comparisons of different implementations over the binary field \mathbb{F}_{2^m} , our implemented design is faster than recent implementations.

Chapter 6

Parallel Point-Multiplication Architecture using Combined Group Operations for High-Speed Cryptographic Applications¹

6.1 Abstract

In this paper, we propose a novel parallel architecture for fast hardware implementation of elliptic curve point multiplication (ECPM), which is the key operation of an elliptic curve cryptography processor. The point multiplication over binary fields is synthesized on both FPGA and ASIC technology by designing fast elliptic curve group operations in Jacobian projective coordi-

¹In review as: Md Selim Hossain, Ehsan Saeedi and Yinan Kong, “Parallel Point-Multiplication Architecture using Combined Group Operations for High-Speed Cryptographic Applications,” *PLOS ONE*, vol. 12, pp. 1-18, May, 2017, DOI: 10.1371/journal.pone.0176214.

nates. A novel combined point doubling and point addition (PDPA) architecture is proposed for group operations to achieve high speed and low hardware requirements for ECPM. It has been implemented over the binary field which is recommended by the National Institute of Standards and Technology (NIST). The proposed ECPM supports two Koblitz and random curves for the key sizes 233 and 163 bits. For group operations, a finite-field arithmetic operation, e.g. multiplication, is designed on a polynomial basis. The delay of a 233-bit point multiplication is only 3.05 and 3.56 μ s, in a Xilinx Virtex-7 FPGA, for Koblitz and random curves, respectively, and 0.81 μ s in an ASIC 65-nm technology, which are the fastest hardware implementation results reported in the literature to date. In addition, a 163-bit point multiplication is also implemented in FPGA and ASIC for fair comparison which takes around 0.33 and 0.46 μ s, respectively. The area-time product of the proposed point multiplication is very low compared to similar designs. The performance ($\frac{1}{Area \times Time} = \frac{1}{AT}$) and Area \times Time \times Energy (ATE) product of the proposed design are far better than the most significant studies found in the literature.

6.2 Introduction

With the swift growth of secure transactions over the network, the demand for cryptography to ensure security has increased rapidly in recent times. Public-key cryptography (PKC) and secret-key cryptography are the two main types of cryptography used for different data-security purposes. Various PKC techniques exist in the literature; among them elliptic curve cryptography (ECC) [10, 11] and the Rivest-Shamir-Adleman (RSA) cryptosystem [8, 137] are the most popular. However, ECC became popular for resource-

constrained environments because it offers the same level of security as the traditional RSA cryptosystem with a significantly shorter key. For example, a 233-bit ECC over a binary field provides equivalent security to 2048-bit RSA [2,22,23]. The National Institute of Standards and Technology (NIST) [22] and IEEE [23], have standardized elliptic curve parameters for prime fields as well as binary fields. The proposed point multiplication hardware is implemented using the NIST standard on an FPGA, which provides higher flexibility of hardware design than an application-specific integrated circuit (ASIC), and means that the cryptographic algorithm can easily be updated if using FPGAs as hardware devices. Also, FPGAs are cheaper for prototype design or in small volumes since they do not incur any fabrication cost. However, bulk production (e.g. in high volumes) of ASICs, after the first run, is much cheaper than the corresponding production based on FPGA devices. Besides, ASIC-based implementation is needed for faster and low-power customized applications.

Elliptic curve point multiplication (ECPM), also called point multiplication, is defined as $Q = k.P$, where the multiplication of an elliptic curve point P by a scalar k provides the resultant point Q [2]. Numerous FPGA implementations of point multiplication over a binary field $GF(2^m)$ have been proposed in the literature, e.g.[35, 45, 53, 110, 111, 115, 125, 138–148]. In the literature, most of the implementations of ECPM over $GF(2^{163})$ are not secure based on today's security level requirements. For this reason, a 233-bit point multiplication is implemented both in FPGA and ASIC. In addition, a 163-bit ECPM is implemented for a fair comparison purpose. In [110, 111, 138], a scalable elliptic curve cryptosystem processor in $GF(2^m)$ is proposed which reduces the latency of ECPM by improving finite-field arithmetic blocks. A Xilinx Virtex-5 FPGA is used in [138] and a Xilinx Virtex-4 FPGA is used in [110, 111] as a hardware platform. However, they have not focused on optimization of elliptic curve group operations in their design. An FPGA implementation of ECPM based on the Montgomery ladder method over binary

fields is proposed in [139] and [146]. They designed the point multiplication using elliptic curve point addition (PA) and point doubling (PD). An efficient FPGA implementation of ECPM over binary finite fields is proposed in [35, 45, 140, 142, 143]. Among them [35] produces better results using digit-serial binary field operations. In [140], a point multiplication was designed in $GF(2^{163})$ for Koblitz curves only. In [141, 144] and [147], a parallel architecture for scalar point multiplication was implemented on a Xilinx Virtex-4 FPGA using the Lopez-Dahab method and separate PA and PD. A practical hardware implementation of point multiplication over $GF(2^{163})$ is proposed using polynomial residue arithmetic in [145]. Several ASIC-Based ECC processors have been proposed over the binary fields in the literature [35, 73, 131, 144, 149–153]. ECC can be used for modern practical applications like mobile services [154], authentication for identity protection for smart grid, wireless sensor and mesh networks [155–157], biometric-based authentication [158], identity-based cryptography [159], and session initiation protocol [160].

Various techniques are introduced, using either FPGA or ASIC implementation, to improve the performance of point multiplication, such as algorithm optimization and improved finite-field arithmetic architectures. Besides, most point multiplication architectures were implemented using separate group operations, which may increase the latency of group operations, hence reduce the speed of point multiplication. Although a few high-speed point-multiplication techniques for an ECC processor have been presented in the literature, most are only area-efficient. Our proposed architecture has a trade-off between speed and area which is suitable for modern faster cryptographic applications.

Contributions: This paper proposes a parallel hardware architecture for point multiplication using combined point doubling and point addition (PDPA) in Jacobian projective coordinates. The proposed point multiplication is synthesized both in FPGA and ASIC. A novel optimized data-flow architecture of the PDPA is introduced to develop high-performance point multiplication. The designed PDPA module is highly parallel, which

means that it takes only one clock cycle to complete. In addition, a parallel hardware architecture using separate group operations (PD and PA) for the ECPM is designed and implemented, and compared with the performance of point multiplication using our combined PDPA. The point multiplication using the combined PDPA provides almost 13 times better performance than using separate group operations. To implement efficient group operations, hence point multiplication, a parallel architecture for field multiplication on a polynomial basis is introduced. The proposed point multiplication requires less time and a smaller area-time (AT) and area-time-energy (ATE) product, providing almost 50% better performance or efficiency than recent implementations.

This paper is organized as follows. Section 6.3 gives an introduction and the mathematical background of ECC over the binary field \mathbb{F}_2^m . The proposed point multiplication architecture is described in Section 6.4. Section 6.5 describes elliptic curve group operations, namely PD, PA, and PDPA. Finite-field arithmetic, e.g. field multiplication, for \mathbb{F}_2^m is given in Section 6.6. Section 6.7 discusses the FPGA and ASIC implementation results and compares our work to the state of the art. Section 6.8 summarizes our work.

6.3 ECC Background

ECC is a popular and powerful public-key encryption technique for cryptographic applications, and nowadays it is very popular due to the smaller field size, in either prime fields or binary fields. An elliptic curve over a binary field will be the emphasis of this work because it is very efficient for hardware implementation due to the use of modulo-2 arithmetic. An elliptic curve defined over a finite field provides a group structure that is used to implement the cryptographic system. The group operations are PD and PA. We have combined these two group operations into a compact hardware implementation and called it PDPA. Two well-known coordinate systems are often used for elliptic curve

group operations: Affine coordinate systems and projective coordinate systems. A point on the elliptic curve E for affine coordinates can be represented by using two elements $x, y \in \mathbb{F}_2^m$, i.e. $P(x, y)$, whereas in projective coordinates, a point P on the EC needs three elements $X, Y, Z \in \mathbb{F}_2^m$, i.e. $P(X, Y, Z)$. In this work, we have implemented all elliptic curve operations in a Jacobian projective coordinate system, avoiding costly modular inversion.

An elliptic curve E over the binary field $GF(2^m)$ (or \mathbb{F}_2^m) in affine coordinates is the set of solutions to the equation

$$y^2 + xy = x^3 + ax^2 + b \quad (6.1)$$

where $x, y, a, b \in GF(2^m), b \neq 0$. The coefficients $a, b \in \mathbb{F}_2^m$ are defined by the NIST standard, which is listed in [2, 22]. In our design, the value of m is 163 which means that we have implemented a 163-bit ECC system.

Let $P = (x, y)$ be a point in an affine coordinate system; the Jacobian projective coordinates $P = (X, Y, Z)$ are given by

$$X = x; Y = y; Z = 1. \quad (6.2)$$

The Jacobian projective point $P = (X, Y, Z)$, $Z \neq 0$ corresponding to the affine point $P = (x, y)$ is given by

$$x = X/Z^2; y = Y/Z^3. \quad (6.3)$$

Using (6.1) and (6.3), the projective form of the Weierstrass equation of the elliptic curve becomes

$$Y^2 + XYZ = X^3 + aX^2Z^2 + bZ^6 \quad (6.4)$$

where the point at infinity is defined as $(1, 1, 0)$. Let $P = (X_1, Y_1, Z_1)$ and $Q = (X_2, Y_2, Z_2)$ be two points on the elliptic curve, then the PD and PA formulae in Jacobian projective

coordinates are given below, for doubling (6.5) and adding (6.6)

$$\begin{aligned}
 R(X_3, Y_3, Z_3) &= 2P(X_1, Y_1, Z_1) \in E(\mathbb{F}_2^m), \\
 Z_3 &= X_1 Z_1^2, \\
 X_3 &= (X_1^4 + b Z_1^8), \\
 Y_3 &= X_1^4 Z_3 + (X_1^2 + Y_1 Z_1 + Z_3) X_3;
 \end{aligned} \tag{6.5}$$

$$\begin{aligned}
 R(X_3, Y_3, Z_3) &= P(X_1, Y_1, Z_1) + Q(X_2, Y_2, Z_2) \in E(\mathbb{F}_2^m), \\
 Z_3 &= Z_1 Z_2 W, \\
 X_3 &= a Z_3^2 + R(R + Z_3) + W^3, \\
 Y_3 &= (R + Z_3) X_3 + Z_1^2 W^2 (R X_2 + Y_2 Z_1 W),
 \end{aligned} \tag{6.6}$$

where $W = (X_1 Z_2^2 + X_2 Z_1^2)$ and $R = (Y_1 Z_2^3 + Y_2 Z_1^3)$.

Hence when $P = Q$, then $R = 2P$ is the PD operation corresponding to (6.5) and when $P \neq Q$, then $R = P + Q$ is the PA operation corresponding to (6.6) [87]. The implementation hierarchy of the ECC system over the binary field $\text{GF}(2^m)$ is presented in Fig. 6.1. From this figure, elliptic curve cryptographic schemes such as ECDSA and

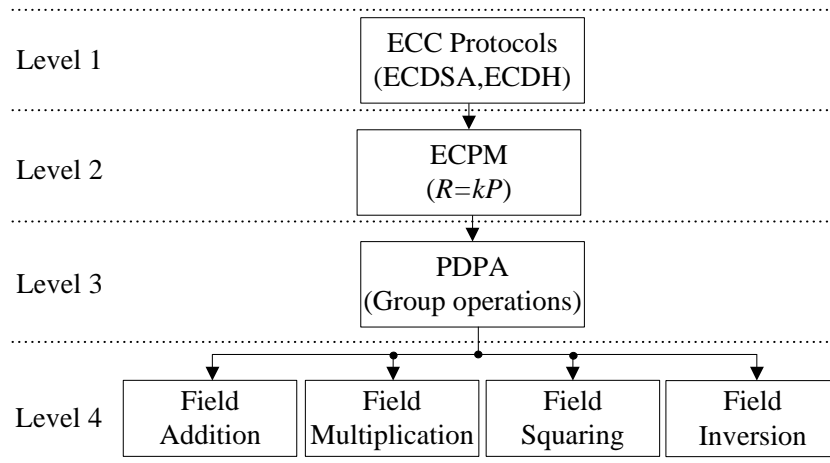


Figure 6.1: Implementation hierarchy of the ECC operations over \mathbb{F}_{2^m} .

ECDH are the building blocks of ECPM and elliptic curve group operations e.g., PDPA. This is the series of finite-field arithmetic operations such as field addition, multiplication, squaring, and inversion. The bottom level is finite-field arithmetic units, which are crucial for the overall performance of an ECC processor. Details of the algorithm, and a hardware architecture for ECPM, are discussed in Section 6.4.

6.4 Proposed Point Multiplication in Projective Coordinates

Point multiplication is the core operation of an ECC processor. It is computationally the most expensive operation throughout the entire processor. However, we have designed a novel parallel architecture for ECPM using our developed PDPA and finite-field arithmetic units. Details of group operations and finite-field arithmetic algorithms and the corresponding architectures which are essential for ECC are discussed in Section 6.5 and Section 6.6, respectively.

6.4.1 Point Multiplication Algorithm

The three most-used algorithms for implementing point multiplication are (1) double-and-add, (2) non-adjacent form (NAF) addition-subtraction chain, and (3) Montgomery ladder product. The easiest to implement is the double-and-add method, shown in Algorithm 6.1. In this approach, the scalar k (which is the private/secret key) is represented in binary, and iterates through each bit. Generally, a PD operation performs on every iteration, and a PA operation only performs when the particular bit of k is one. However, we have implemented a combined PDPA operation which produces PD and PA results simultaneously on each cycle. Then m iterations are required to compute the final result of ECPM, but each iteration needs only one clock cycle (CC) (CC for PDPA).

Algorithm 6.1: Double-and-add (Left to right) method for ECPM

Input: $k = (k_{m-1}, \dots, k_1, k_0)_2$, $P(X, Y, Z) \in E(\mathbb{F}_2^m)$

Output: $Q(X, Y, Z) = k.P(X, Y, Z)$, where $Q(X, Y, Z) \in E(\mathbb{F}_2^m)$

1. $Q = 0$;
 2. **for** $i = m - 1$ **to** 0 **do** $Q = 2Q$;
 - 2.1 **if** $k(i) = '1'$ **then** $Q = Q + P$; **end**
 - 2.2 **end for**
 3. **Return** $(Q(X, Y, Z))$
-

6.4.2 Architecture for ECPM

A novel point multiplication architecture is proposed in Jacobian projective coordinates using our designed PDPA architecture, which is highly parallel. Note that most ECC implementations in the literature have used separate PD and PA modules, and require more computation time. The proposed ECPM architecture using PDPA is shown in Fig. 6.2. Our proposed ECPM consists of PDPA, counter, select logic, multiplexer, and register modules. In Fig. 6.2, the PDPA architecture generates the PD and PA results at the same time because it performs the group operations in parallel. For example, when $1P(X1, Y1, Z1)$ is an input, this architecture generates the $2P(2PX, 2PY, 2PZ)$ and $3P(3PX, 3PY, 3PZ)$ results concurrently. In this architecture, the outputs of PDPA are $X3_PD$, $Y3_PD$, $Z3_PD$, which stand for the outputs of PD, and $X3_PA$, $Y3_PA$, and $Z3_PA$, which stand for the outputs of PA. In this approach, the PDPA module is the main component to make a faster point multiplication. As can be seen from Fig. 6.2, a two-bit 'sel2s' signal is generated from the select logic unit which is based on PD outputs. When PD results are zero, 'sel2s = 01', when PD results are equal to $1P(X1, Y1, Z1)$, then 'sel2s = 10', otherwise 'sel2s = 00' is produced from the select logic unit. Thus, 'sel2s' is a control signal for the MUX1 module that decides which output passes to the

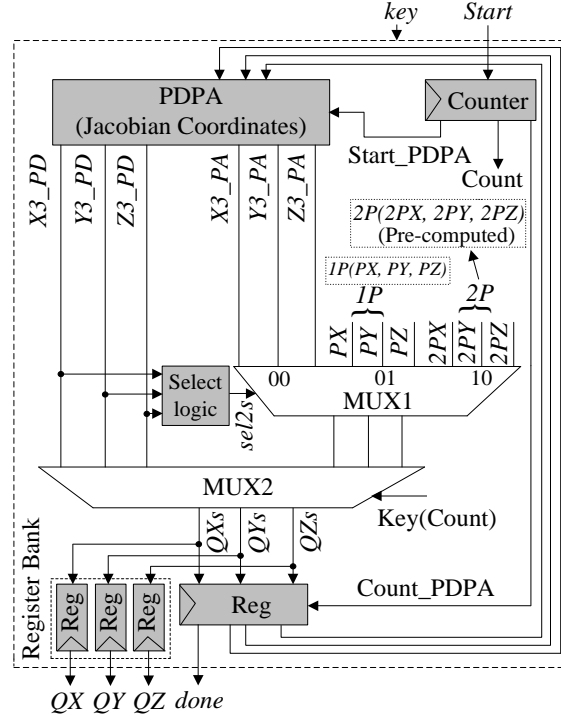


Figure 6.2: Hardware architecture of proposed ECPM in Jacobian coordinates.

MUX2 module. As one can see from MUX1 in Fig. 6.2, are of the $1P(X1, Y1, Z1)$, $2P(2PX, 2PY, 2PZ)$, and PA results, based on the 'sel2s' signal, goes to the MUX2 module, which means that when 'sel2s = 00', then PA results, when 'sel2s = 01', then $1P(X1, Y1, Z1)$, and when 'sel2s = 10', then $2P(2PX, 2PY, 2PZ)$ results are selected. The PA result from the PDPA module goes to the output when the particular bit of 'key' is one. Similarly, the PD result goes to the output when 'key' is zero. Hence, the PD and PA results are stored in the register bank to get the output. A counter module is used to decide when the results will be passed to the next input of the PDPA module. Note that, the combined PDPA module needs only one clock cycle to compute the PD and PA results concurrently, although it looks to need many logic stages. In this method, only 233 and 163 CCs are needed to compute a 233-bit and 163-bit point multiplication, respectively in projective coordinates due to the highly parallel PDPA architecture, which will be discussed in the next section.

6.4.3 Security Analysis

A combined PDPA architecture is designed and implemented which performs the PD and PA operations concurrently, as demonstrated in Fig 6.3C. For this reason, the power consumption pattern for the PDPA hardware will be symmetric in nature. As shown in Fig 6.2, an ECPM hardware is developed using this combined PDPA architecture. A uniform power consumption profile may be measured throughout the point multiplication computation. From the analysis, we can say that any ‘key’ information is difficult to observe from this hardware. Besides, the double-and-add algorithm is secure against timing and simple power analysis (SPA) attacks [90].

6.5 Proposed Group Operations

A separate PD and PA architecture as well as a combined PDPA architecture have been designed in Jacobian projective coordinates for point multiplication. To decrease the latency of the group operations in Fig. 6.3, different techniques have been used such as balancing the architecture, parallelization in operations, and pre-computations. In this work, we have utilized Koblitz curve K-163 for implementing group operations. Also, our proposed group operations are supports for a random curve. For doing this, the coefficients $a, b \in \mathbb{F}_2^m$ defined by NIST [22] have been changed. Fig. 6.3 depicts the proposed architecture of group operations in projective coordinates, corresponding to (6.5) and (6.6). From Fig. 6.3(a), the cost of PD is $4A + 5M + 5S$, where A, M, and S are the costs of field addition, multiplication, and squaring, respectively. Field addition is the simplest operation in the binary field $\text{GF}(2^m)$, being simply a bit-wise exclusive-or (xor (\oplus)). Field multiplication is one of the most complex operations in $\text{GF}(2^m)$. However, we have proposed an efficient architecture for field multiplication. A field squarer is similar to a field multiplier. As can be seen from Fig. 6.3(a), only 7 levels are required to implement

the PD operation, and it is fully parallel. The hardware architecture for PA corresponding to (6.6) is shown in Fig. 6.3(b). This architecture is also fully parallel, and the cost of this architecture is $7A + 15M + 5S$. Fig. 6.3(b) demonstrates that 11 levels are required for computing PA. Fig. 6.3(c) illustrates the combined architecture for a group operation in Jacobian projective coordinates named PDPA. There are 18 levels (7 for PD and 11 for PA) required for group operations using separate architectures, whereas the combined architecture needs only 14 levels. Using this parallel combined architecture, the number of levels in the data path is reduced, which means that the number of logic stages can be minimized, and the overall performance is improved.

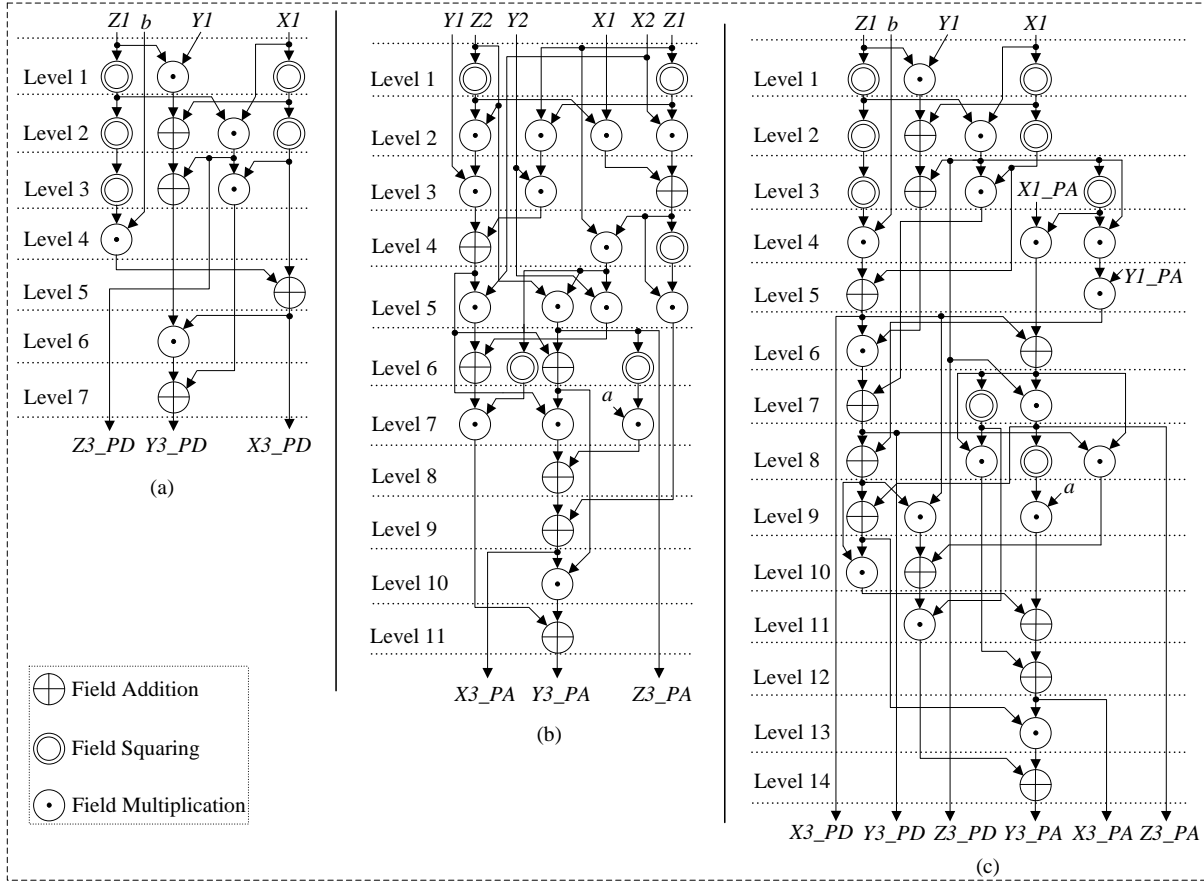


Figure 6.3: Proposed data-flow architecture for parallel computation of elliptic curve:
(a) PD, (b) PA, and (c) PDPA.

6.6 Proposed Field Multiplication for \mathbb{F}_2^m

This section presents a field multiplication algorithm and a corresponding hardware architecture using a polynomial basis. It is the most crucial operation in implementing point multiplication, because the overall latency of ECPM in projective coordinates mostly depends on the field multiplication. The irreducible polynomials $f(x) = x^{163} + x^7 + x^6 + x^3 + 1$ and $f(x) = x^{233} + x^{74} + 1$ have been used for the field $\text{GF}(2^m)$ (163-bit and 233-bit ECC). Field multiplication computes the product of two polynomials then applies modular reduction, as shown in (6.7):

$$Z(x) = U(x) \cdot V(x) \bmod f(x) \quad (6.7)$$

Algorithm 6.2 presents field multiplication over binary field \mathbb{F}_2^m . The proposed paral-

Algorithm 6.2: Field multiplication in $\text{GF}(2^m)$

Input: $U(x), V(x) \in \text{GF}(2^m)$, an irreducible polynomial $f(x)$ of degree m

Output: $Z(x) = U(x) \cdot V(x) \bmod f(x)$

1. $Z_v = 0$; $P = f(x)$;
 2. **for** $j = m - 1$ **to** 0 **do**
 - 2.1 $U_v = '0'$ & $U(x)$; $Z_v = Z_v \cdot x$ (left-shift operation) ;
 - 2.2 **for** $i = 0$ **to** $m - 1$ **do** $U_v(i) = U_v(i)$ and $V(j)$; **end for**
 - 2.3 $Z_v = Z_v \text{ xor } U_v$;
 - 2.4 **for** $l = 0$ **to** m **do** $P_v(l) = P(l)$ and $Z_v(m)$; **end for**
 - 2.5 $Z_v = Z_v \text{ xor } P_v$;
 3. **end for**
 4. **Return** $Z(x)$
-

lel architecture corresponding to Algorithm 6.2 is shown in Fig. 6.4. As can be seen from Fig. 6.4(a), two field additions are performed at the same time. However, this method

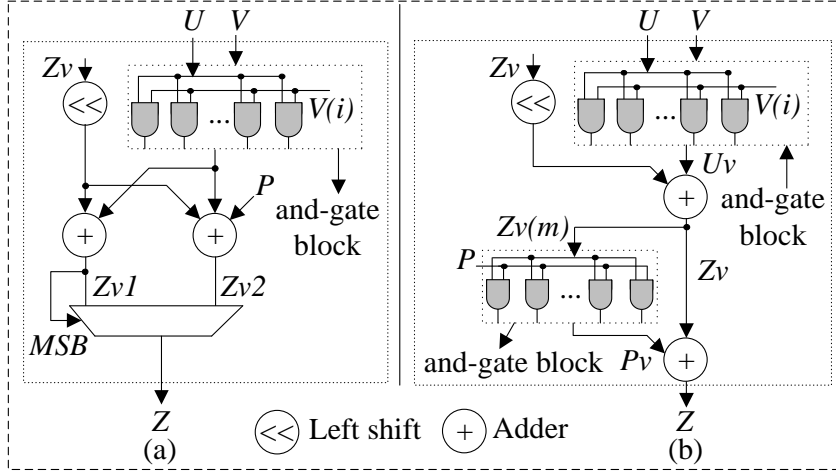


Figure 6.4: Proposed parallel field multiplication architecture in $GF(2^m)$.

requires one multiplexer module, is a more expensive operation than the and-gate block (P_v). On the other hand, Fig. 6.4(b) (Algorithm 6.2) needs only two field additions, one left-shift operation, and two and-gate blocks. Multiplication by x can easily be computed by the binary left-shift operation. The and-gate operation is also straightforward as well as time efficient both on FPGA and ASIC. From Algorithm 6.2, we check whether the result is an element of $GF(2^m)$ with degree $< m$. Only when the multiplication result Z_v has degree m or higher is a modular reduction step necessary. This condition is checked by $Z_v(m)$. When the particular bit of $Z_v(m)$ is zero, then P_v from the and-gate block generates zero results. Otherwise, P_v generates some result which depends on the modulus $f(x)$ ($P = f(x)$). The proposed polynomial-basis multiplication algorithm is better for ASIC-based implementation due to the efficient and-gate block. This architecture is performed fully in parallel. A parallel group operation has been designed using this efficient field multiplication.

6.7 Comparisons and Performance Analysis

In this section, a performance comparison of various hardware implementations of point multiplication is discussed. The proposed point multiplication has been implemented using synthesizable VHDL code, and synthesized, placed and routed using Xilinx ISE 14.7 with an optimized goal of ‘speed’. It was simulated using both ModelSim PE and ISim. The target FPGA selected is the Xilinx Virtex-7 (XC7VX485T-2FFG1761). We have also implemented our design on a Xilinx Virtex-6 FPGA. In addition, we have synthesized our design using Synopsys Design Compiler with the 65-nm United Microelectronics (UMC) standard logic-cell library. The synthesis results provide better performance in terms of speed and energy than other similar designs in the literature.

In the literature, most of the point multiplications were implemented over $GF(2^{163})$, but it is of no practical interest to test the algorithm for $GF(2^{163})$, since this curve is no longer approved by NIST to generate digital signatures. For a fair comparison, we have implemented 233-bit as well as 163-bit ECPM for both random and Koblitz curves. Table 6.1 depicts the performance and a comparison of FPGA implementations of point multiplication over $GF(2^{233})$. The AT value and performance of this design is comparable with other designs in the literature as shown in Fig. 6.5. As can be seen from Table 6.1 and Fig. 6.5, the point multiplication for a 233-bit random curves takes a little bit more delay and area than with the Koblitz curve. The combined group operation (PDPA) is used to implement ECPM instead of separate PD and PA operations, because the proposed combined PDPA provides better performance than separate group operations. In addition, 163-bit point multiplication is also implemented using both combined and separate group operations for fair comparison.

The proposed point multiplication over $GF(2^{233})$ is synthesized using a Xilinx Virtex-7 (XC7V980T-2FFG1930) FPGA; results are demonstrated in Table 6.1. As we can see in Table 6.1, the latency of 233-bit point multiplication is almost 3 μs for a Koblitz curve

and $3.56 \mu\text{s}$ for a random curve with the maximum frequency of 76.50 and 65.48 MHz, respectively. Moreover, the proposed design takes very few clock cycles to implement, which is much better than other comparable work in the literature, but it takes more than 100K slices without using any DSP slices. From the results, we can say that the design provides high speed, but it takes a huge area to implement. However, we have a trade-off between speed and area. Note that the proposed parallel architecture is not suitable for lower versions of the FPGA due to resource (e.g. slices) limitations. On the other hand, our proposed point multiplication over $\text{GF}(2^{233})$ provides a higher throughput rate than other related work. As one can see in Fig. 6.5, the AT and performance of our design is similar to [138], but better than [110] and [111]. The point multiplication proposed in [35] and [45] provides a little bit better performance than our proposed design. However, our proposed design is almost six times as fast as [35] and almost four times as fast as [45], making it suitable for cryptographic applications that require high throughput rate.

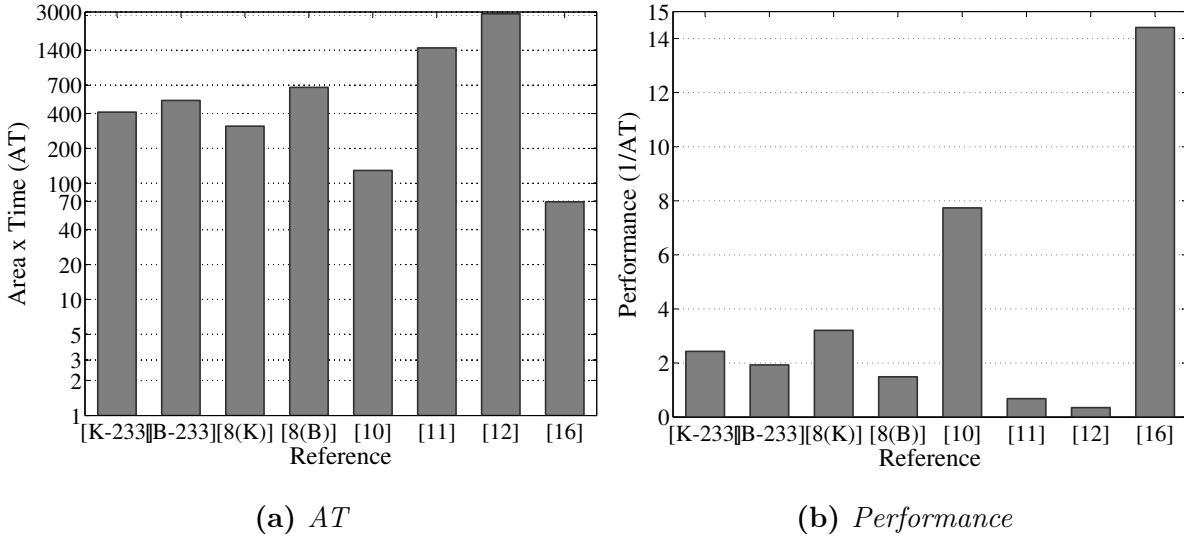


Figure 6.5: Comparison ((a) *AT* and (b) *performance*) of point multiplication over $\text{GF}(2^{233})$.

Table 6.2 shows a performance comparison of point multiplication over the last few

Table 6.1: *Performance analysis of point multiplication on FPGA over $GF(\mathcal{Q}^{233})$*

Work	Platform	Field	Length	Reported Area (slices)	Cycles	Time (μs /ECPM) @f (MHz)	Area \times Time (AT) (Kilo-slices $\times\mu s$)	Performance (1/AT)	TR ² (Mbps)
This work ¹	Virtex-7	K-233		134.68K	233	3.05@76.50	410.78	2.43	76.39
		B-233		145.42K	233	3.56@65.48	517.70	1.93	65.45
[138] Loi (2014)	Virtex-5	K-233		7.43K	6792	41.91@162.07	311.39	3.21	5.56
		B-233		7.98K	12955	84.19@154.35	671.83	1.49	2.77
[35] Sutter (2013)	Virtex-5	233		6.49K	3825	19.89@192.30	129.10	7.74	11.71
[110] Loi (2013)	Virtex-4	233		2.43K	93825	604@155.38	1467.72	0.68	0.39
[111] Loi (2013)	Virtex-4	233		2.65K	155785	1093@142.53	2894.26	0.35	0.21
[45] Rebeiro (2012)	Virtex-5	233		5.64K	1919	12.3@156.00	69.42	14.41	18.94

1. Using Virtex-7 (XC7VX980T-2FFG1930) and 2. TR = Throughput rate

Table 6.2: Performance comparison of point multiplication on FPGA over $GF(2^{163})$

Work	Platform	Reported Area (slices)	Cycles	Time (μs /ECPM) @f (MHz)	Area \times Time (AT) (Kslices $\times\mu s$)	Performance (1/AT)	TR (Mbps)
[a]	This work ¹ Virtex-7*	57.39K	163	0.31@528.58	17.70	56.50	528.58
[b]		72.43K	163	0.31@528.58	22.45	44.54	528.58
[c]		71.96K	163	0.33@500.16	23.45	42.64	500.16
[d]		73.47K	163	0.33@500.16	24.25	41.24	500.16
[e]	This work ² Virtex-7*	64.80K	163	3.51@46.41	227.59	4.39	46.41
[f]		77.54K	163	3.82@42.71	296.19	3.40	42.71
[138] Loi (2014)	Virtex-5	7.43K ^{8a} 7.98K ^{8b}	4745 9130	29.28@162.07 59.15@154.35	217.46 471.90	4.60 2.12	5.57 2.76
[139] Liu (2014)	Virtex-4	10.42K	1091	9.00@121.00	93.75	10.70	18.11
[35] Sutter (2013)	Virtex-5	6.15K	1371	5.50@249.27	33.83	29.56	29.64
[110] Loi (2013)	Virtex-4	2.43K	42419	273@155.38	663.39	1.51	0.60
[111] Loi (2013)	Virtex-4	2.65K	68842	483@142.53	1279.95	0.78	0.34

[a], [c], [e] for Koblitz curve and [b], [d], [f] for random curve. *using Virtex-7 (XC7VX485T-2FFG1761) and Virtex-6 (XC6VLX760-2ff1760).

1. ECPM using PDPA. 2. ECPM using separate PD and PA.

Table 6.2: *Performance comparison of point multiplication on FPGA over $GF(2^{163})$*

Work	Platform	Reported Area (slices)	Cycles	Time (μ s/ECPM) @f (MHz)	Area \times Time (AT) (Kslices $\times\mu$ s)	Performance (1/AT)	TR (Mbps)
[140] Reza (2013)	Stratix II	23.08K ALMs	1721	9.15@188.71	-	-	17.81
[141] Mahdizadeh (2013)	Virtex-4	17.93K	2751	9.60@250.00	172.12	5.81	16.97
[142] Reza (2012)	Virtex-5	5.79K	3880	14.50@267.10	83.93	11.92	11.24
[45] Rebeiro (2012)	Virtex-5	3.5K	1436	8.60@167.00	29.64	33.74	18.95
[143] Roy (2012)	Virtex-5	12.43K	552	12.10@45.62	150.40	6.65	13.47
[144] Zhang (2010)	Virtex-4	20.81K	1428	7.70@185.00	160.21	6.24	21.17
[145] Dimitrios (2009)	Virtex-2	12245 LUTs	9107	39.00@233.50	-	-	4.18
[125] Chelton (2008)	Virtex-4	16.21K	3010	19.55@153.90	316.89	3.16	8.33
[146] Ansari (2008)	Virtex-2	3.42K	4075	41.00@100.00	140.06	7.14	3.98
[147] Kim (2008)	Virtex-4	24.36K	1446	10.00@143.00	243.63	4.11	16.30
[115] Antao (2008)	Virtex-4	10.49K	14256	144.00@99.00	1510.27	0.66	1.13

[a], [c], [e] for Koblitz curve and [b], [d], [f] for random curve. *using Virtex-7 (XC7VX485T-2FFFG1761) and Virtex-6 (XC6VLX760-2FF1760).

1. ECPM using PDPA. 2. ECPM using separate PD and PA.

years in FPGA technology as compared with our proposed parallel design over $\text{GF}(2^{163})$. In the available literature, most point multiplication architectures were implemented using separate PD and PA (group operations) modules. We have proposed a novel ECPM hardware in Jacobian coordinates using PDPA (combined group operations). Our design takes m clock cycles for m -bit point multiplication, which is much less than other designs. As can be seen from Table 6.2, the point multiplication for a 163-bit random curve takes the same time as with the Koblitz curve, but it takes a little bit more area than the random curve. The proposed ECPM using PDPA architecture takes less time than all other similar designs on FPGA. We have achieved a point multiplication in $0.31 \mu\text{s}$ and $0.33 \mu\text{s}$ in a Virtex-7 and Virtex-6 FPGA, respectively. In addition, an ECPM is designed and implemented using separate group operations which take $3.51 \mu\text{s}$ for a Virtex-7 FPGA and $3.82 \mu\text{s}$ for a Virtex-6 FPGA. As can be seen from Table 6.2, ECPM using a combined PDPA architecture performs 13 times as fast as separate modules in either a Virtex-7 or a Virtex-6 FPGA device.

In Table 6.2, the results of [35, 45, 138, 140, 142, 143, 145, 146] show FPGA implementations of point multiplication in $\text{GF}(2^{163})$. They used trivial group operations (PD and PA) for implementing ECPM. Their proposed designs require fewer slices than our design, but they need more clock cycles, hence more computation time, to complete. Point multiplication schemes over the binary field $\text{GF}(2^{163})$ are presented in [110, 111, 115, 125, 139, 141, 144, 147]. Their proposed point multiplication schemes were implemented in a Virtex-4 FPGA device. Of them, the result provided in [139] shows the best result in terms of performance as shown in Table 6.2. On the other hand, our proposed point multiplication using the PDPA architecture delivers 5 times the performance ($1/\text{AT}$) of those in [139]. Besides, the throughput rate of our design is far better than the others.

The AT and performance or efficiency metric are the best indicators to say which design is better. The performance or efficiency of point multiplication is defined in (6.8),

in ECPM operations per sec per slice. The area-time (AT) comparison of point multiplication over $\text{GF}(2^{163})$ with similar designs is shown in Fig. 6.6. As can be seen from the graph, our design provides a lower AT value than all other designs. Fig. 6.7 compares the performance of point multiplication with similar work in Table 6.2. Note that [a], [b], [c], [d], [e], and [f] of Figs 6.6 and 6.7 represent our implementation results and [35, 45, 110, 111, 115, 125, 138, 139, 141–144, 146, 147] illustrate reference FPGA implementations over $\text{GF}(2^{163})$. The AT and performance metric demonstrates that we have achieved a higher efficiency than most of the similar designs in the available literature. Note that, of all the available designs, in terms of AT value the designs proposed in [35] and [45] perform better. However, we have achieved a 50% better performance than their designs. The point multiplication techniques proposed in the literature need fewer slices but require more computation time than our design. From the comparison of various ECPMs over the binary field $\text{GF}(2^{163})$ in Table 6.2, our novel parallel point multiplication using combined PDPA in Jacobian coordinates is the fastest hardware implementation result reported in the literature to date.

$$\text{Performance} = \text{Efficiency} = \frac{1}{\text{Area} \times \text{Time}} = \frac{1}{\text{AT}} \quad (6.8)$$

In the state of the art, few implementations are targeted on ASIC, being mostly FPGA implementations. Both technologies (FPGA and ASIC) have been utilized for this paper. Table 6.3 depicts the ASIC-based performance analysis and comparison of elliptic curve point multiplication over $\text{GF}(2^{233})$ and $\text{GF}(2^{163})$. The proposed high-speed parallel point-multiplication architecture is synthesized using 65-nm CMOS technology, a more advanced version of ASIC technology than 0.13 μm , 0.18 μm , and 0.35 μm CMOS technology. Besides, we have optimized our design for Koblitz (K-233 and K-163) curves as well as random (B-233 and B-163) curves to compare with those of other similar studies. We find that the NIST random curve takes more area than the NIST Koblitz curve

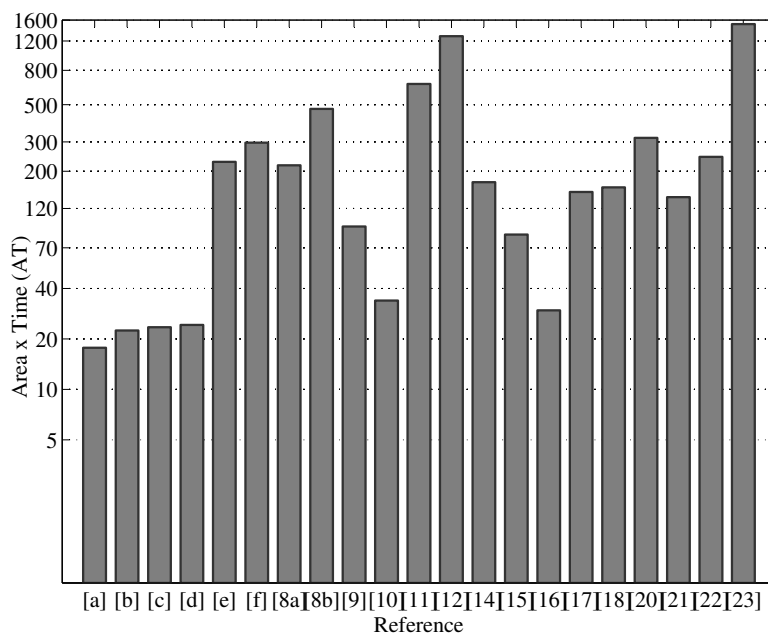


Figure 6.6: *AT comparison of point multiplication ([a], [b], [c], [d], [e], and [f] represent our work) over $GF(2^{163})$.*

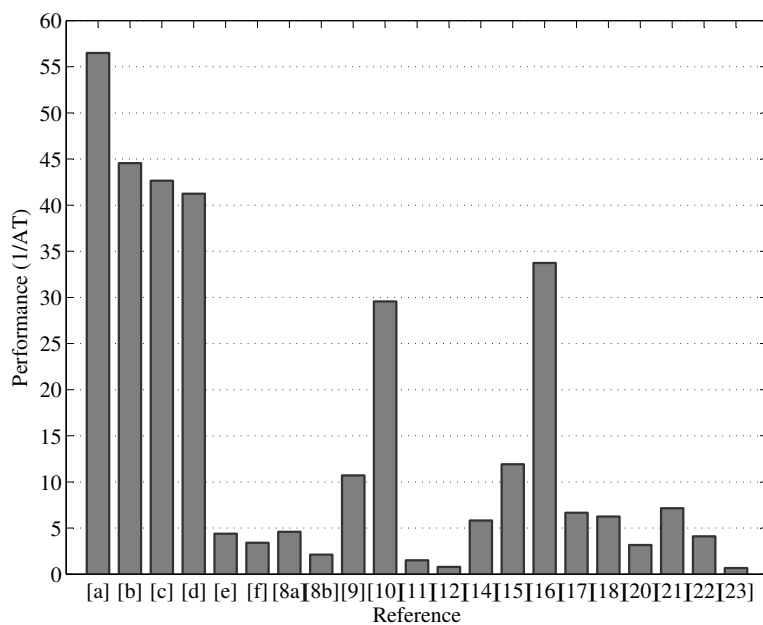


Figure 6.7: *Performance comparison of point multiplication ([a], [b], [c], [d], [e], and [f] represent our work) over $GF(2^{163})$.*

Table 6.3: *Performance analysis of ASIC-based point multiplication over binary fields*

	Technology	Field	Area ¹	KCycles	Time (μ s) @f (MHz)	AT ²	Energy (μ J)	ATE ³	TR ⁴ (Mbps)
[g]		K-233	7.56/3635	0.233	0.81@289	6.12/2.94	0.88	0.0054/2.59	288
[h]	This work 65-nm	B-233	8.42/4048	0.233	0.81@289	6.82/3.28	0.98	0.0067/3.21	288
[i]		K-163	3.43/1649	0.163	0.46@353	1.58/0.76	0.22	0.00035/0.17	354
[j]		B-163	3.47/1668	0.163	0.46@353	1.60/0.77	0.24	0.00038/0.18	354
		B-233 ^{26a}	1.12/313	124.3	520@238	582/162.8	34.0	19.8/5534	0.45
[73]	90-nm	B-163 ^{26b}	0.24/65	62.5	220@277	53/14.3	8.0	0.43/117	0.74
[35]	180-nm	B-163	-/138	1.70	9.5@179	-/1.3	-	-/-	17
[149]	130-nm	163	2.34/332	182.6	440@415	1030/146.1	61.0	63/8911	0.37
[144]	180-nm	163	-/218	1.43	5.4@263	-/1.2	-	-/-	30
[150]	180-nm	B-163	2.10/69	228.1	1890@181	3969/130.4	257.0	1020/33515	0.09
[151]	130-nm	163	-/12.5	275.8	244000@0.001	-/3050	9.0	-/27450	0.0007
[152]	130-nm	163	-/393	22.0	70@292	-/27.5	-	-/-	2.33
[153]	350-nm	163	-/16	376.8	27900@14	-/446.4	-	-/-	0.006
[131]	350-nm	163	-/46	134.0	3050@44	-/140.3	-	-/-	0.05

1. Area = (mm^2 /Kilo-Gates¹), 2. AT = Area \times Time ($\text{mm}^2 \times \mu\text{s}$)/(KGs \times ms), 3. ATE = Area \times Time \times Energy ($\text{mm}^2 \times \mu\text{s} \times \text{mJ}$)/(KGs \times ms $\times \mu\text{J}$), and 4. Throughput rate.

for ASIC-based point multiplication design. The proposed design needs only $0.81 \mu s$ for 233-bit ECPM and $0.46 \mu s$ for 163-bit ECPM, either Koblitz or random curve, to complete. The point multiplication over $GF(2^{233})$ takes 7.56 mm^2 (for K-233) with 3635K gate count and 8.42 mm^2 (for B-233) area with 4048K gate count in UMC 65-nm technology. Similarly, the results for 163-bit ECC (both in Koblitz and random curves) are depicted in Table 6.3, which takes 3.43 mm^2 (for K-163) with 1649K gate count and 3.47 mm^2 (for B-163) area with 1668K gate count. The implemented design is also energy-efficient. The energy is computed from the power consumption and point multiplication time. The energy consumption per point multiplication over $GF(2^{163})$ and $GF(2^{233})$ is between 0.22 and $0.98 \mu J$ which is far less than most recent designs. For example, the power consumption of B-163 point multiplication is 487 mW , of which 178 mW is for cell internal power, 307 mW is for net switching power, and the rest is leakage power. Similarly, the power consumption for 233-bit ECC is simulated from the Synopsys design compiler.

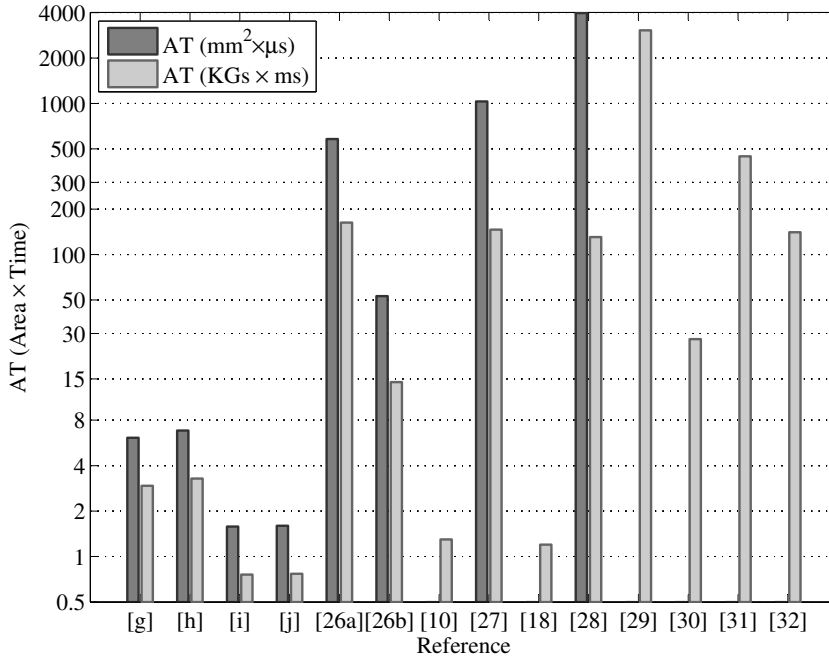


Figure 6.8: *AT comparison of point multiplication ([g], [h], [i], and [j] represent our work) with references.*

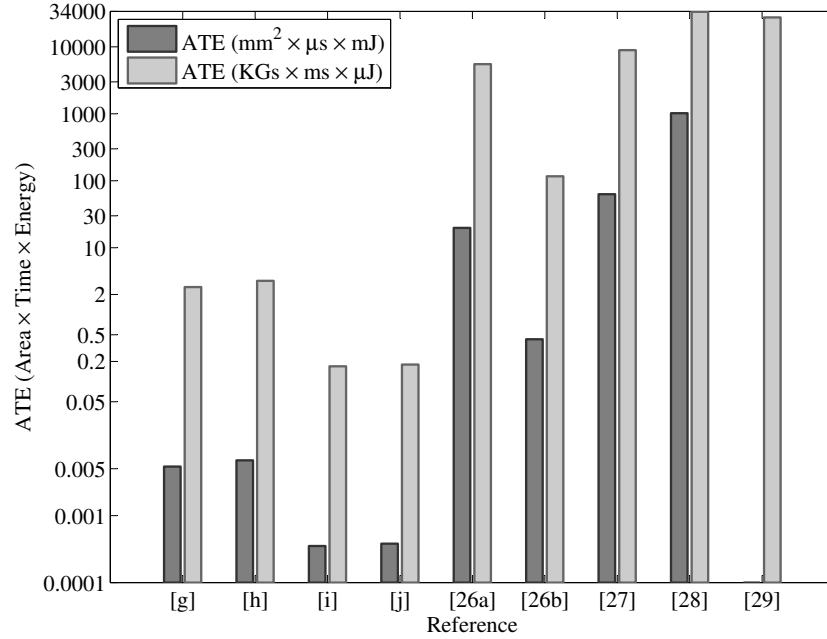


Figure 6.9: *ATE comparison of point multiplication ([g], [h], [i], and [j] represent our work) with related designs.*

Table 6.3 shows our synthesis results and the most recent work using ASIC implementation. As can be seen from Table 6.3, our design is faster as well as more energy-efficient than all other significant designs found in the available literature. However, our design is not area-efficient due to the parallel architecture. This is a kind of design trade-off between area, time, and energy. For a fair comparison, we have calculated area×time (AT) and area×time×energy (ATE) products. Figs 6.8 and 6.9 show the area-delay and area-delay-energy products for our proposed design and related circuits presented in Table 6.3. It is crystal clear that we present more outstanding results than other designs in terms of AT and ATE.

6.8 Conclusion

A novel parallel architecture for point multiplication, the core operation of an ECC processor, has been proposed and implemented over $GF(2^{233})$ and $GF(2^{163})$. It is implemented by the double-and-add method using Jacobian projective coordinates. To provide efficient point multiplication, a novel combined group operation (PDPA) is designed which performs the PD and PA operations in parallel, aimed at reducing the number of levels and logic stages needed with separate PD and PA operations. A parallel field multiplication using a polynomial basis is developed for group operations, hence point multiplication.

Using parallel architecture, the proposed 233-bit ECPM takes only $3.05 \mu s$ (for K-233) and $3.56 \mu s$ (for B-233) in a Xilinx Virtex-7 FPGA. In addition, we have achieved a point multiplication over $GF(2^{163})$ in $0.31 \mu s$ and $0.33 \mu s$ in a Virtex-7 and Virtex-6 FPGA, respectively. Regarding ASIC synthesis results, the proposed design takes a similar delay to FPGA implementation. The core area of the proposed design is a little bit higher than similar designs, namely 7.56 mm^2 (for K-233) and 3.43 mm^2 (for K-163). The energy consumption per point multiplication is only 0.88 and $0.22 \mu J$ for K-233 and K-163, respectively.

We can say that the proposed parallel architecture for point multiplication is energy-efficient. However, in both technologies (FPGA and ASIC), we require more area for implementation. According to our best knowledge, the proposed parallel point multiplication architecture is the fastest hardware implementation result to date. Based on the overall performance and comparisons, a 50% improvement is achieved over recent FPGA implementations and significant improvement is gained over the most recent ASIC-based designs. We conclude that our proposed design provides better performance which can be used for modern high-speed cryptographic applications.

Chapter 7

Efficient Hardware Implementation of Elliptic Curve Cryptography Processor Over NIST Binary Fields¹

7.1 Abstract

This paper presents a high-performance hardware architecture of an elliptic curve cryptography processor (ECP) over NIST binary fields. A novel elliptic curve point multiplication (ECPM) architecture is proposed for an ECP using combined point doubling and point addition (PDPA) hardware in Jacobian coordinates. Therefore, we have presented three versions of ECPs, two for FPGA-based ECPs using bit-serial and digit-serial field multiplication, and one for ASIC-based implementation. Our proposed ECP support all five Koblitz and random curves for the key sizes from 163 to 571-bit recommended

¹In review as: Md Selim Hossain, Shahzad Asif and Yinan Kong, "Efficient Hardware Implementation of Elliptic Curve Cryptography Processor Over NIST Binary Fields," *IET Computers & Digital Techniques*, in review.

by NIST. For the ECP, elliptic curve group operations are optimized for both Koblitz and random curves using a combined architecture. This paper also proposes bit-serial and digit-serial field multiplication architectures. In addition, a field inversion is designed to convert from Jacobian to affine coordinates. Using this PDPA and field arithmetic unit, the point multiplication, hence the ECP, latency is reduced. Finally, the FPGA-based design provides better efficiency for both bit-serial and digit-serial version ECP than other related work. Furthermore, we have achieved an energy-efficient ECP in which the $\text{Area} \times \text{Time} \times \text{Energy}$ (ATE) value is lower in an ASIC platform than all comparable work in the literature.

7.2 Introduction

Elliptic curve cryptography (ECC), is currently the popular and leading public-key cryptographic technique in terms of security, speed, area, and power consumption rather than the commonly used cryptosystem RSA [8]. It was first proposed in the mid-80s by Koblitz and Miller [1, 10]. ECC utilizes smaller binary field arithmetic that consumes minimum hardware resources, which is the most favorable for portable devices to save data with their limited resources. Furthermore, ECC contains a very attractive feature; computationally much more efficient than other cryptosystems, nearly ten times smaller key size provides equivalent security to the RSA cryptosystem, making ECC very popular for resource-constrained applications [35, 73]. Consequently, less memory and hardware resources are needed to develop an ECC processor (ECP). Therefore, to implement an efficient ECP, a high-performance finite-field arithmetic (FFA), for instance, finite field addition, squaring, multiplication, and inversion, are mandatory. Also, efficient FFA operations are enabling potentially higher data rates at a much lower implementation cost.

Hence, we have proposed and implemented an efficient FFA which speeds up the overall performance of the ECP. The NIST [22] and IEEE P1363-2000 [23] standards are utilized for the ECP implementation.

7.2.1 Related Work

Numerous hardware implementations of ECPs over binary fields $\text{GF}(2^m)$ have been presented in the literature. Among them, [118, 126–129, 131, 134, 136, 161–167] proposed FPGA-based ECPs using bit-serial multiplication, [110, 111, 116, 117, 119–121] presented FPGA-based implementations of ECPs using digit-serial multiplication. However, only a few hardware implementation of ECPs over binary fields $\text{GF}(2^m)$ target an ASIC platform [73, 131, 149–153, 168, 169]. In the literature, most of the ECPs were implemented by either FPGA or ASIC as a hardware platform, and some implementations are targeted to both technologies [35, 131, 149]. Despite that, some hardware architectures of ECP in the literature can support a maximum of four NIST binary fields, whereas our proposed ECP supports all five NIST binary fields over $\text{GF}(2^m)$. In addition, the proposed designs were synthesized for both FPGA and ASIC technologies in which we focus on both bit-serial and digit-serial field multiplication. Besides, most of the literature focuses on algorithm and corresponding hardware architecture optimization for FFA units only. Furthermore, point doubling (PD) and point addition (PA) are the most used ways to compute elliptic curve group operations in the available literature. In this paper, both FFA and group operations are optimized to improve the performance of point multiplication, which is the main operation of an ECP.

7.2.2 Our Contribution

In this paper, a novel ECP architecture over $\text{GF}(2^m)$ is proposed using a combined PDPA architecture which improves the performance of point multiplication. The implemented

ECP hardware supports all NIST binary fields from 163 to 571 bits, which can be used for higher-security (cryptographic) applications. The proposed ECP is synthesized using FPGA technology, which allows lower cost and greater system flexibility (like updating algorithms). Consequently, the FPGA-based ECP provides more than 50% better performance than similar work. We also present FPGA energy consumption using a proper power analysis technique. In addition, the proposed ECP is synthesized on an ASIC platform which allows high speed and low energy consumption. We have achieved 3 to 100 times or 2 to 55 times less Area \times Time \times Energy (ATE) values using ASIC 65-nm technology than the most significant work in the literature. For high-performance point multiplication, an efficient combined architecture (PDPA) for group operations is developed which can compute the PD and PA operations concurrently. We optimize the PDPA architecture for both Koblitz and random curves using proper data-flow, pre-computation, and parallelization techniques. The proposed PDPA and ECPM is implemented in Jacobian coordinates where we can avoid the costly field-inversion operation. The latency of group operations, hence point multiplication, is reduced using the novel PDPA architecture. A Jacobian-to-affine coordinate conversion unit is developed for the ECP using a field inversion, four field multiplications, and one field squaring unit. Hence, a high-performance FFA unit is designed which contains finite field multiplication, squaring, and inversion. Moreover, we have proposed a polynomial basis multiplication over $\text{GF}(2^m)$ using both a bit-serial and digit-serial approach. Finally, the main focus of this paper is to achieve a high-speed, low-area, and low-power ECP hardware which can be used for modern high-security applications.

7.2.3 Organisation of the Chapter

Section 7.3 deals with preliminary background of ECC over the binary field \mathbb{F}_2^m . Section 7.4 describes algorithms and corresponding hardware architectures for finite field arith-

metic and Section 7.5 presents PDPA architecture. The description of the proposed ECP hardware architecture is delivered in Section 7.6. Implementation results and comparisons of the most significant work in the literature over binary fields are discussed in Section 7.7. Finally, summaries of the work are given at the end of this chapter.

7.3 Mathematical Background of ECC

An elliptic curve E for the binary field \mathbb{F}_2^m in affine coordinates is the set of solutions to the equation

$$y^2 + xy = x^3 + ax^2 + b \quad (7.1)$$

where $x, y, a, b \in GF(2^m), b \neq 0$. The coefficients $a, b \in \mathbb{F}_2^m$ are defined by the NIST and IEEE P1363-2000 standards. There are two NIST standards in the binary field: one is for Koblitz curves and one is for random curves or binary curves are listed in [2, 22].

An elliptic curve for a finite or Galois field provides a group structure that is used to implement the cryptographic system. The elliptic curve group operations are point addition (PA) and point doubling (PD). A combined group operation has been developed for a faster hardware implementation and called PDPA. Various coordinate systems exist in the literature such as Jacobian, Lopez-Dahab, and Chudnovsky coordinates; a detailed coordinate system is discussed in [2]. However, affine and Jacobian coordinate systems are often used for ECC to represent elliptic curve points. In this paper, Jacobian projective coordinates are utilized for PDPA. Then we convert to affine coordinates for a more practical realization. In Jacobian coordinates, a point P needs three elements $X, Y, Z \in \mathbb{F}_2^m$, i.e. $P(X, Y, Z)$ corresponding to the affine point $P = (x, y)$. Then the Jacobian coordinates $P = (X, Y, Z)$ are given by

$$X = x; Y = y; Z = 1. \quad (7.2)$$

The Jacobian point $P = (X, Y, Z)$, $Z \neq 0$ corresponding to the affine point $P = (x, y)$ is given by

$$x = X/Z^2; \quad y = Y/Z^3. \quad (7.3)$$

Using (7.1), (7.2), and (7.3), the projective form of the Weierstrass equation of the elliptic curve becomes

$$Y^2 + XYZ = X^3 + aX^2Z^2 + bZ^6 \quad (7.4)$$

where the point at infinity is $(1, 1, 0)$. If $P = (X_1, Y_1, Z_1)$ and $Q = (X_2, Y_2, Z_2)$ are two points on the elliptic curve, then the PD and PA formulae in Jacobian projective coordinates are given below, for doubling (7.5) and adding (7.6) [87]:

$$\begin{aligned} R(X_3, Y_3, Z_3) &= 2P(X_1, Y_1, Z_1) \in E(\mathbb{F}_2^m), \\ Z_3 &= X_1 Z_1^2, \\ X_3 &= (X_1^4 + bZ_1^8), \\ Y_3 &= X_1^4 Z_3 + (X_1^2 + Y_1 Z_1 + Z_3) X_3; \end{aligned} \quad (7.5)$$

$$\begin{aligned} R(X_3, Y_3, Z_3) &= P(X_1, Y_1, Z_1) + Q(X_2, Y_2, Z_2) \in E(\mathbb{F}_2^m), \\ Z_3 &= Z_1 Z_2 W, \\ X_3 &= aZ_3^2 + R(R + Z_3) + W^3, \\ Y_3 &= (R + Z_3) X_3 + Z_1^2 W^2 (RX_2 + Y_2 Z_1 W), \end{aligned} \quad (7.6)$$

where $W = (X_1 Z_2^2 + X_2 Z_1^2)$ and $R = Y_1 Z_2^3 + Y_2 Z_1^3$.

7.4 Hardware for Finite Field Arithmetic

Fig. 7.1 depicts the hardware implementation hierarchy of the ECC operations over the binary field $\text{GF}(2^m)$. Each level of this hierarchy depends on the performance of the

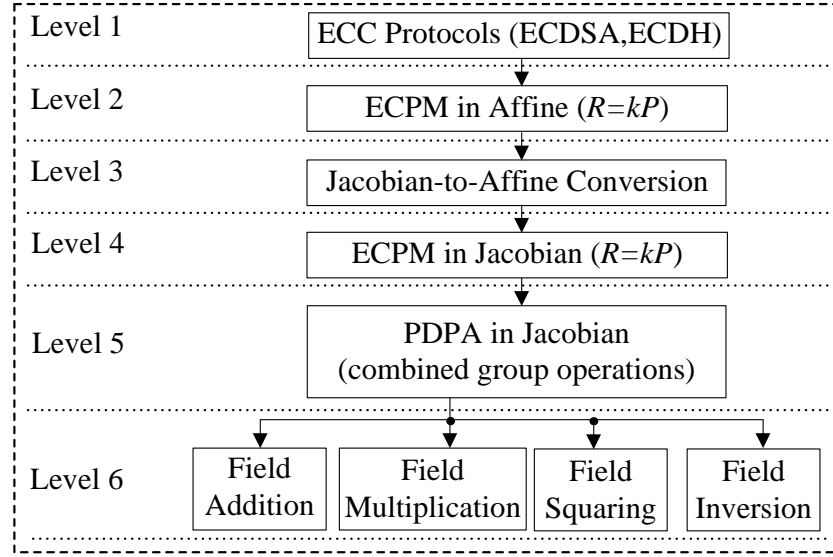


Figure 7.1: *Implementation hierarchy of the ECC operations over \mathbb{F}_{2^m} .*

underlying level of operations. As we can see in Fig. 7.1, elliptic curve digital signature algorithm (ECDSA) and elliptic curve Diffie-Hellman (ECDH), which called ECC protocols, are the top level of the hierarchy. Consequently, in the second level called ECPM in affine coordinates, is the main operation of an ECP, which depends upon a Jacobian-to-affine coordinate conversion in the third level and point multiplication in Jacobian coordinates in the fourth level, respectively. The conversion from Jacobian to affine coordinates needs only one inversion operation. The field inversion operation in the binary field can be avoided for point multiplication in Jacobian coordinates. Therefore, ECPM in Jacobian coordinates (fourth level in the hierarchy) relies on the combined elliptic curve group operations (PDPA) (fifth level in the hierarchy), which depends on the finite field arithmetic (FFA) units field addition, multiplication, and squaring. Lastly, the FFA units make up the sixth level in the hierarchy, and these are the most crucial for high-performance ECP. Note that, field addition is the cheapest operation over a binary field; it is just a bit-wise exclusive-or (xor) operation. In this section, all algorithms and corresponding hardware architectures related to FFA units are discussed.

7.4.1 Finite Field Modular Reduction

FFA for a binary field is performed modulo the corresponding reduction polynomial of the binary field used. A modular reduction needs to be performed on the result of multiplication, squaring, and inversion to ensure that it exists within the binary field chosen. Hence, it uses the reduction polynomial $f(x)$ of the corresponding binary field so that a binary polynomial $Z_v(x)$ is reduced to $Z(x)$, where $Z(x) = Z_v(x) \bmod f(x)$. Modular reduction in arbitrary polynomials can be done one bit at a time for bit-serial polynomial multiplication. Therefore, the reduction or irreducible polynomials recommended by NIST in the FIPS 186-2 standard $f(x) = x^{163} + x^7 + x^6 + x^3 + 1$, $f(x) = x^{233} + x^{74} + 1$, $f(x) = x^{283} + x^{12} + x^7 + x^5 + 1$, $f(x) = x^{409} + x^{87} + 1$, and $f(x) = x^{571} + x^{10} + x^5 + x^2 + 1$, have been utilized for 163, 233, 283, 409 and 571-bit ECC, respectively.

7.4.2 Finite Field Multiplication

Field multiplication in a polynomial basis is the most important and an expensive operation for point multiplication in Jacobian coordinates. The most popular methods to perform field multiplication in a polynomial basis includes an interleaved modular reduction algorithm, the Karatsuba-Ofman Algorithm, bit-serial multiplication, higher-radix multipliers, digit-serial multiplication, and digit-parallel multiplication [2, 43–45]. Field multiplication computes the product of two polynomials, then applies modular reduction with $f(x)$. Let $U(x)$ and $V(x)$ be two inputs and $Z(x)$ be their output, then

$$Z(x) = U(x).V(x) \bmod f(x), \quad (7.7)$$

where $f(x)$ is a constant irreducible polynomial of degree m .

Algorithm 7.1 shows the proposed bit-serial finite field multiplication. In this algorithm, the shift-and-add technique is used for faster hardware implementation because a

Algorithm 7.1: Bit-Serial Field multiplication in a polynomial basis over $\text{GF}(2^m)$

Input: $U(x), V(x) \in \text{GF}(2^m)$, irreducible polynomials of degree m

Output: $Z(x) = U(x) \cdot V(x) \bmod f(x)$

1. $Z_v = 0$; $P = f(x)$;
 2. **for** $j = m - 1$ **to** 0 **do**
 - 2.1 $U_v = '0' \ \& \ U(x)$; $Z_v = Z_v \cdot x$ (left-shift operation) ;
 - 2.2 **for** $i = 0$ **to** $m - 1$ **do** $U_v(i) = U_v(i)$ and $V(j)$; **end for**
 - 2.3 $Z_v = Z_v \oplus U_v$;
 - 2.4 **for** $l = 0$ **to** m **do** $P_v(l) = P(l)$ and $Z_v(m)$; **end for**
 - 2.5 $Z_v = Z_v \oplus P_v$;
 3. **end for**
 4. **Return** $Z(x)$
-

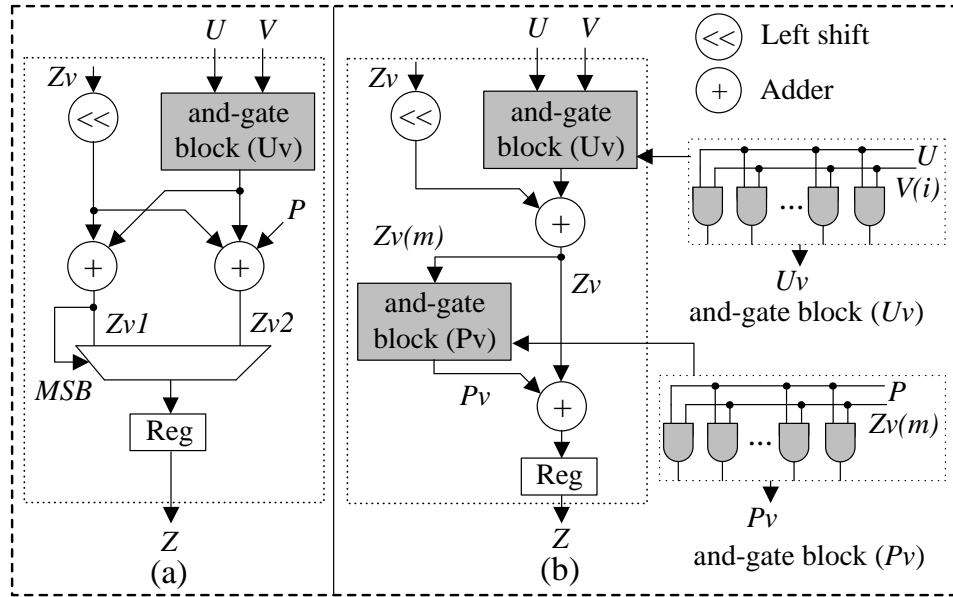


Figure 7.2: Proposed bit-serial field multiplication architecture in $\text{GF}(2^m)$.

vector shift can be performed in one clock cycle (CC). As Step 2 in Algorithm 7.1, if $V_j = 1$, then $U(x)$ is added to the register Z_v and the product is left-shifted ($Z_v \cdot x$). Therefore,

a modular reduction is only needed when the result of $Z_v(m) = 1$. This operation can be easily calculated by the addition of irreducible polynomials, e.g. $Z_v = Z_v \oplus f(x) = Z_v \oplus P_v$. Fig. 7.2 depicts the proposed hardware architecture corresponding to Algorithm 7.1, which performs on left-to-right bits in the bit-serial approach. In Fig. 7.2(a), two field additions are performed in parallel, so that this method requires one multiplexer which is a more expensive than the and-gate (P_v) operation. On the other hand, Fig. 7.2(b) needs only two field additions, one left-shift operation, and two and-gate operations. Besides, the and-gate operation is the most basic and faster operation for hardware implementation. Fig. 7.2(b) is finally implemented, with only a single bit of the result produced at every clock cycle. The area cost of bit-serial field multiplication is only $O(m)$ which is more efficient in terms of area than a parallel multiplier whose area cost is $O(m^2)$.

Though field multiplication using a bit-serial approach is area-efficient, it is not time-efficient. Numerous techniques have been introduced in the literature [2, 43–45, 110, 111, 116, 117, 119–121] to achieve faster hardware implementation of field multiplication. The digit-serial multiplier is one of them, which speeds up the multiplication process significantly. The digit-serial multiplier over a binary field is depicted in Algorithm 7.2. Fig. 7.3 represents the proposed digit-serial multiplier over $\text{GF}(2^m)$ corresponding to Algorithm 7.2. In this method, multiple partial products are generated and added per cycle, thus greatly reducing the number of clock cycles. We have implemented 16-bit, 32-bit, and 64-bit digit-serial multipliers so that their AT products can be compared with others. The latency of a digit-serial multiplier is $O(m/d)$, where m is the ECC field bit-length and d is the digit size. As can be seen from Fig. 7.3, the number of padding bits for register V_v is determined by first calculating $q = m/d$. In this architecture, register U_v is used to store the new multiplier representing U from ‘*shiftModU*’ for the next cycle. Similarly, the product during each multiplication cycle is stored in the register Z_v . This approach is very efficient in hardware cost since it uses left-shift and add (xor) operations. In each

cycle, $V(i)$ is multiplied by ‘ $tempU$ ’ which contains $U.x^{di} \bmod f(x)$ of the current cycle. The computation of $U.x^{d(i+1)} \bmod f(x)$ is performed in parallel during the multiplication of $V(i)$ and ‘ $tempU$ ’. ‘ $tempU$ ’ is left-shifted by d bits followed by reduction with polynomial $f(x)$. Register Z_v contains $V_{q-1}.(A.x^{dq-1}) \bmod f(x)$ at the final iteration, but with degree greater than m . For this reason, reduction of Z_v is also needed to obtain the final result Z . This method requires fewer clock cycles, which means faster computation. For example, this method needs only 4 clock cycles using 64-bit digit serial for 233-bit field multiplication. However, this technique needs more area to implement.

Algorithm 7.2: Digit-Serial field multiplication in $GF(2^m)$

Input: $U(x), V(x) \in GF(2^m)$, irreducible polynomials of degree m

Output: $Z(x) = U(x) \cdot V(x) \bmod f(x)$

1. $Z_v = 0$;
 2. $V = V_0 + V_1x^d + \dots + V_{q-1}x^{d(q-1)}$, where $V_i = \sum_{j=0}^{d-1} b_{id}x^j$;
 - 3.1 **for** $i = 0$ to $q - 1$ **do**
 - 3.2 $Z_v = Z_v \oplus V_i U$; $U_v = U_v.x^d \bmod f(x)$
 - 3.3 **end for**
 4. $Z = Z_v \bmod f(x)$
 5. **Return** $Z(x)$
-

Finite field squaring is the field multiplication of two identical binary polynomials. A squaring operation is often used since it can be more efficient than field multiplication. This consideration will depend on the performance of field multiplication utilized, since if multiplication becomes more efficient than squaring, then the squaring method can be replaced.

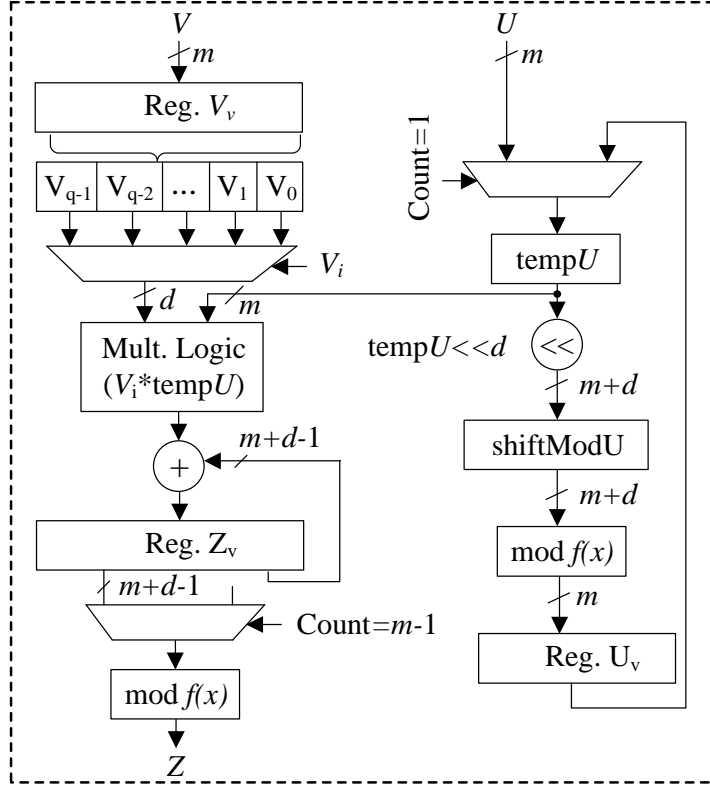


Figure 7.3: Proposed digit-serial field multiplication architecture in $GF(2^m)$.

7.4.3 Finite Field Inversion

Inversion over the binary field is complex and the most expensive operation for FFA. However, it is required only once per point multiplication operation for converting from Jacobian to affine coordinates. Numerous field inversion algorithms exist in the literature, including Fermat's Little theorem, Binary algorithm, the almost inversion algorithm, Extended Euclidean algorithm (EEA), Itoh-Tsujii inversion algorithm, and Montgomery inversion algorithm. In this paper, we adopt the modified Euclidean algorithm which is the most commonly used algorithm for field inversion. Inversion of a non-zero field element $U(x) \in \mathbb{F}_2^m$ is $Z(x) \in \mathbb{F}_2^m$, where $UZ = 1 \bmod f(x)$. Algorithm 7.3 depicts the modified Euclidean algorithm [2, 52]. Using this algorithm, we achieve a better performance than other inversion architectures. Therefore, the hardware architecture for inversion over

$\text{GF}(2^m)$ is described in Fig. 7.4. In this method, all internal signals are $m + 1$ bits long because of the reduction polynomial is also $m + 1$ bits long. In this approach, the inversion is accomplished using a series of additions and shift operations. The computation of division like Z_v/x or multiplication by x is performed by a shift operation. The modular reduction is performed by using the addition operation when the degree of the polynomial is m or higher than m . Using modified EEA, the field inversion $Z(x) = 1/U(x) \bmod f(x)$ is achieved after $2m + 1$ CCs.

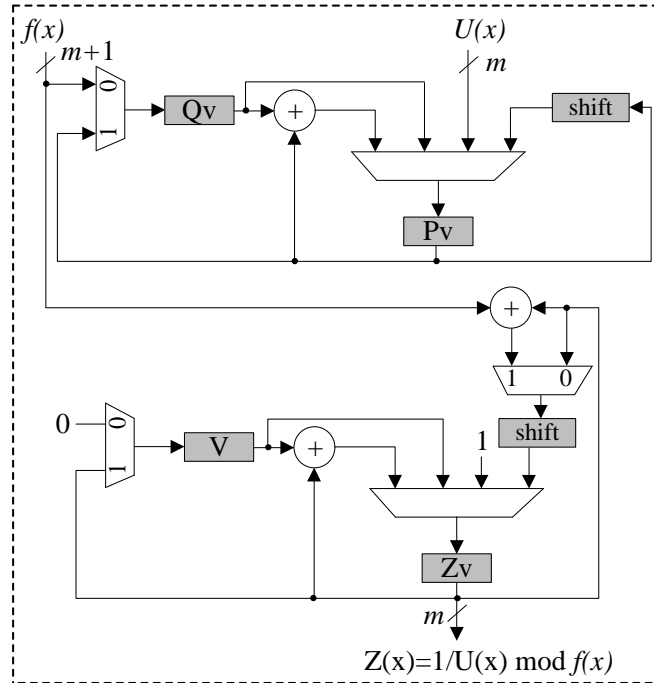


Figure 7.4: Hardware Architecture for field inversion in \mathbb{F}_2^m .

Algorithm 7.3: Modified Euclidean Algorithm for Inversion in $\text{GF}(2^m)$

Input: $U(x) \in \text{GF}(2^m)$, irreducible polynomial of degree m

Output: $Z(x) = 1/U(x) \bmod f(x)$

```

1.   $P_v = '0' \ \& \ U(x)$  ;  $Q_v = f(x)$ ;  $Z_v = 00001$ ;  $V = 0$  ;  $cnt = 0$  ;
2.  for  $i = 1$  to  $2m$  do
2.1.  if  $P_v(m) = '0'$  then
2.1.1.     $P_v = x \cdot P_v$  ;  $Z_v = x \cdot Z_v$  ;
2.1.2.    if  $Z_v(m) = '1'$  then  $Z_v = Z_v + f(x)$  ; end
2.1.3.     $cnt = cnt + 1$  ;
2.1.4.  else
2.1.5.    if  $Q_v(m) = '1'$  then
2.1.6.     $Q_v = Q_v + P_v$  ;  $V = V + Z_v \bmod f(x)$  ; end
2.1.7.     $Q_v = x \cdot Q_v$  ;
2.1.8.    if  $cnt = 0$  then
2.1.9.     $P_v = Q_v$  ;  $Q_v = P_v$  ;  $(P_v \leftrightarrow Q_v)$ 
2.1.10.    $Z_v = V$  ;  $V = Z_v$  ;  $(Z_v \leftrightarrow V, \text{exchange operations})$ 
2.1.11.    $Z_v = x \cdot Z_v \bmod f(x)$ ;  $cnt = cnt + 1$  ;
2.1.12.  else
2.1.13.    $Z_v = Z_v/x \bmod f(x)$  ;  $cnt = cnt - 1$  ;
2.1.14.  end
2.2.  end
3.  end for
4.  Return  $Z(x)$ 

```

7.5 Proposed Group Operations for ECP

This section focuses on the elliptic curve group operations (PD and PA) required to perform point multiplication. We have proposed a combined PDPA architecture for group

operations in Jacobian projective coordinates. In this coordinate system, the costly field inversion operation can be avoided. Note that field inversion in $GF(2^m)$ takes more time and area than field multiplication. Hence the PDPA module consists of field addition, multiplication, and squaring operations. Group operations can be implemented with either NIST Koblitz curves or NIST random curves. Both curves have been implemented in this paper. The data-flow graph of the proposed PDPA architecture for Koblitz curves is illustrated in Fig. 7.5. The bit-serial field multiplication is only used for Koblitz curves (K-163–K-571). Fig. 7.5 (a) is the optimized architecture for the Koblitz curve K-163 because the coefficients a and b are equal to 1 as defined by the NIST standard. The latency (clock cycles) of the PDPA for K-163 is $7m + 16$. Similarly, Fig. 7.5 (b) is optimized for Koblitz curves between K-233 and K-571, because of the coefficients $a = 0$ and $b = 1$. Therefore, the latency for Koblitz curves between K-233 and K-571 is $7m + 15$ which includes one extra clock cycle to store the results into their respective registers. As one can see at level 1 and 2 of the PDPA architecture in Fig. 7.5, one field multiplication and two field squaring operations can be performed concurrently. Consequently, the maximum four field multiplication operations can be performed concurrently at level 10. Fig. 7.6 depicts the proposed combined PDPA architecture for random curves, where Fig. 7.6(a) is the PDPA architecture using bit-serial multiplication and Fig. 7.6(b) is the PDPA architecture using our proposed digit-serial multiplication. Fig. 7.6 demonstrates that 15 levels are required for computing PD and PA operations using this combined architecture. However, 20 levels (9 for PD and 11 for PA) are generally required for group operations using separate architectures [53, 148]. The latency of the PDPA architecture using bit-serial and digit-serial multiplication is $7m + 17$ and $7d + 8$, respectively. Finally, the overall latency of the group operations is decreased by using this combined architecture. The outputs of PDPA are X3_PD, Y3_PD, Z3_PD which stand for the outputs of PD and X3_PA, Y3_PA, and Z3_PA which stand for the outputs of PA. In this architecture, PD and PA can be

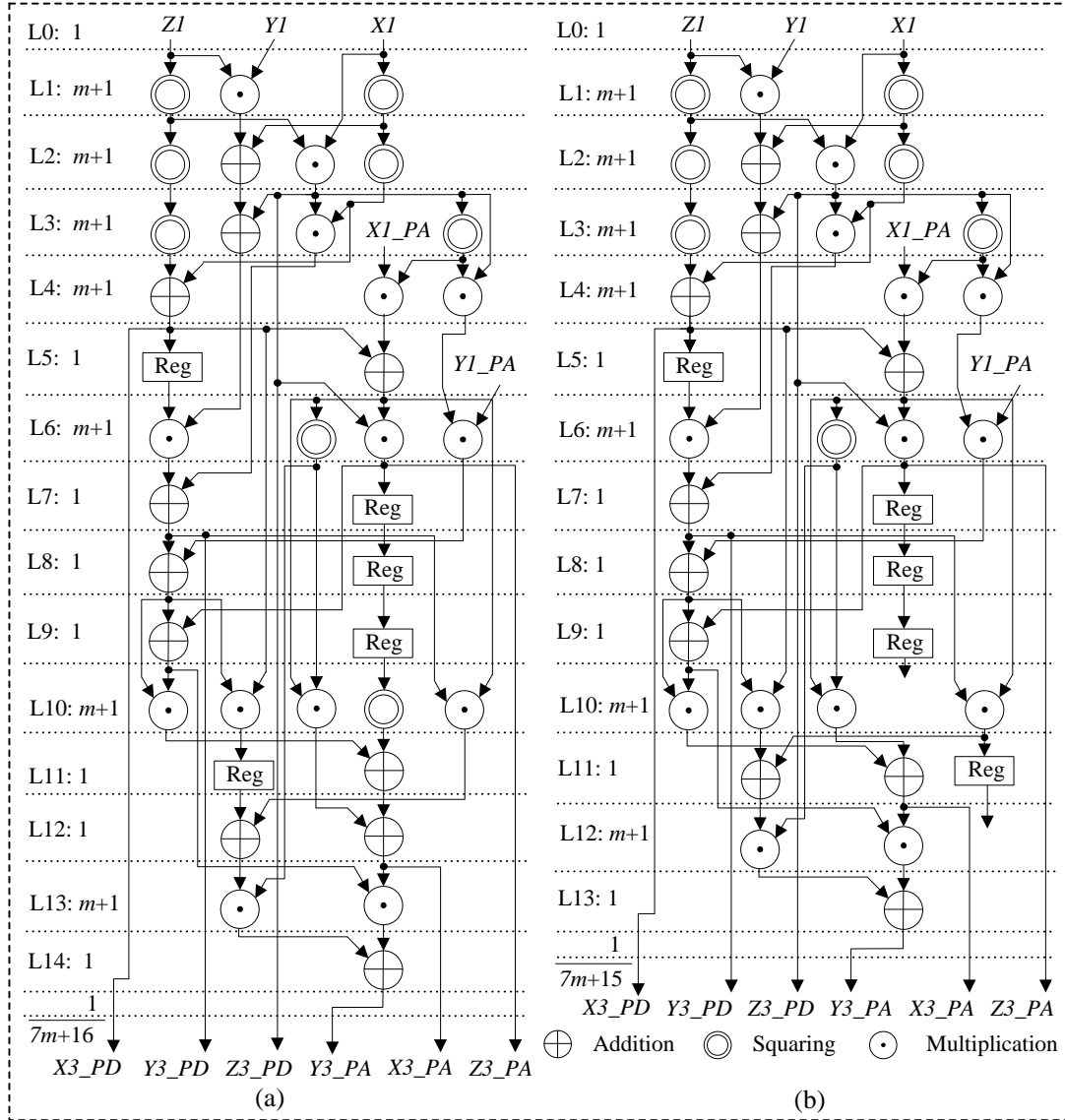


Figure 7.5: Proposed PDPA architecture for Koblitz Curves (a) K-163 and (b) K-233–K-571.

computed concurrently. For example, when $1P(X1, Y1, Z1)$ is an input, this architecture generates the $2P(2PX, 2PY, 2PZ)$ and $3P(3PX, 3PY, 3PZ)$ values concurrently.

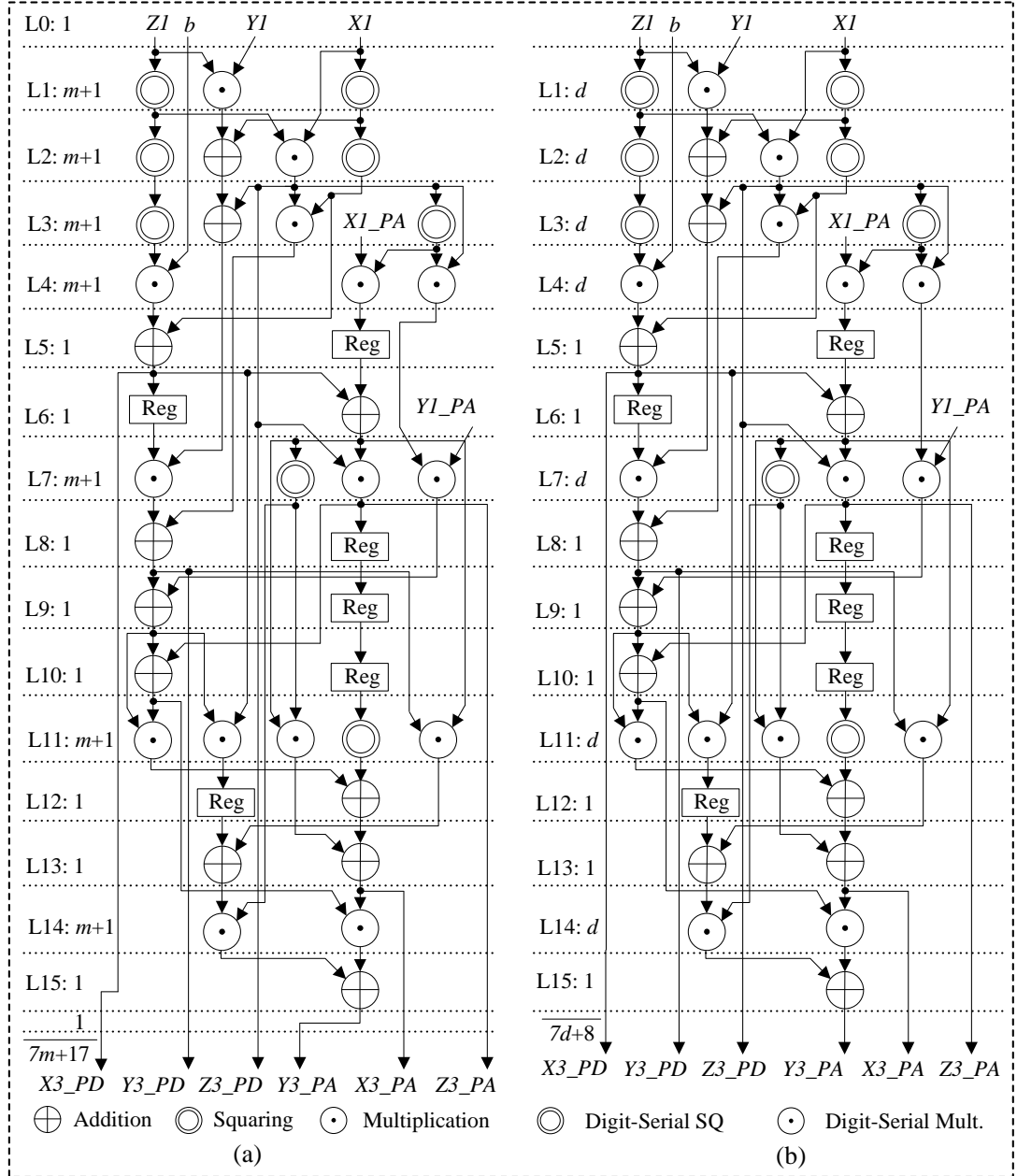


Figure 7.6: Proposed PDPA architecture (a) using bit-serial multiplication and (b) using digit-serial multiplication, for random or binary curves.

7.6 Proposed ECC Processor

The point multiplication operation is $Q = kP$, where k is an integer (which is called the private/secret key), and P and Q are points on the elliptic curve over the binary field. It

is the main operation of an ECC processor (ECP). In this method, a point $P(X, Y, Z)$ is added $k - 1$ times itself to get the final point $Q(X, Y, Z)$ in Jacobian projective coordinates. Different methods/algorithms exist for computing ECPM [2], using execution sequences of group (PD and PA) operations. The execution schedules are directly related to the bit pattern of the scalar multiplier, $k = \text{'key'}$. In this paper, the double-and-add method is used to implement point multiplication, which is depicted in Algorithm 7.4. Generally, a PD operation performs on every iteration, and a PA operation only performs when the particular bit of k is one. However, we have implemented a combined PDPA operation which generates PD and PA results simultaneously on each cycle. Then $m - 1$ iterations are required to compute the point multiplication, where each iteration requires $7m + 17$ or $7d + 8$ clock cycles (CCs for PDPA). In this approach, we can save m clock cycles per point multiplication.

Algorithm 7.4: Double-and-add (Left to right) point multiplication

Input: $k = (k_{m-1}, \dots, k_1, k_0)_2$, $P(X, Y, Z) \in E(\mathbb{F}_2^m)$

Output: $Q(X, Y, Z) = k.P(X, Y, Z)$, where $Q(X, Y, Z) \in E(\mathbb{F}_2^m)$

1. $Q = 0$;
 2. **for** $i = m - 1$ **to** 0 **do** $Q = 2Q$;
 - 2.1 **if** $k(i) = '1'$ **then** $Q = Q + P$; **end**
 - 2.2 **end for**
 3. **Return** $(Q(X, Y, Z))$
-

7.6.1 Proposed Point Multiplication Architecture

Fig. 7.7 illustrates the proposed point multiplication architecture over $\text{GF}(2^m)$ in Jacobian coordinates. The proposed point multiplication architecture consists of combined PDPA, select logic, multiplexer, counter, and a few register sets. The counter unit generates

the respective control signals that act as a control unit, which communicates with other units to perform specific tasks. In this architecture, the PDPA hardware is the main unit, and generates PD and PA results concurrently. Furthermore, the counter module sends a start signal to run the PDPA module. It generates a ‘Count_PDPA’ signal when the PDPA is done. A two-bit control signal named ‘sel2s’ is generated by the select logic module. As can be seen from Fig. 7.7, the control signal ‘sel2s’ is needed as an input for the MUX1 module. When ‘sel2s = 01’ then $1P(PX, PY, PZ)$ results, when ‘sel2s = 10’ then $2P(2PX, 2PY, 2PZ)$ results, which is pre-computed, and when ‘sel2s = 00’ then PA results are generated from the MUX1 module. Hence, the inputs of the MUX2 module come from register 1 and the MUX1 module, which are basically PD and PA results. The outputs of MUX2 depend on the bit pattern of ‘key’. Consequently, after completion of the current cycle, it stores all intermediate results in the second register set then loops back to the next cycle. The output of the point multiplication is in Jacobian form, so conversion is required to acquire affine form. The number of point multiplication clock cycles for a random curve in Jacobian coordinates using bit-serial and digit-serial multiplication are defined by (7.8) and (7.9), respectively .

$$\begin{aligned}
 \text{\#clock cycles (CCs)} &= (m - 1) \times \text{PDPA CCs in Jacobian} \\
 &= (m - 1) \times (7m + 17) \\
 &= (7m^2 + 10m - 17)
 \end{aligned} \tag{7.8}$$

$$\begin{aligned}
 \text{\#clock cycles (CCs)} &= (m - 1) \times \text{PDPA CCs in Jacobian} \\
 &= (m - 1) \times (7d + 8) \\
 &= (7md + 8m - 7d - 8)
 \end{aligned} \tag{7.9}$$

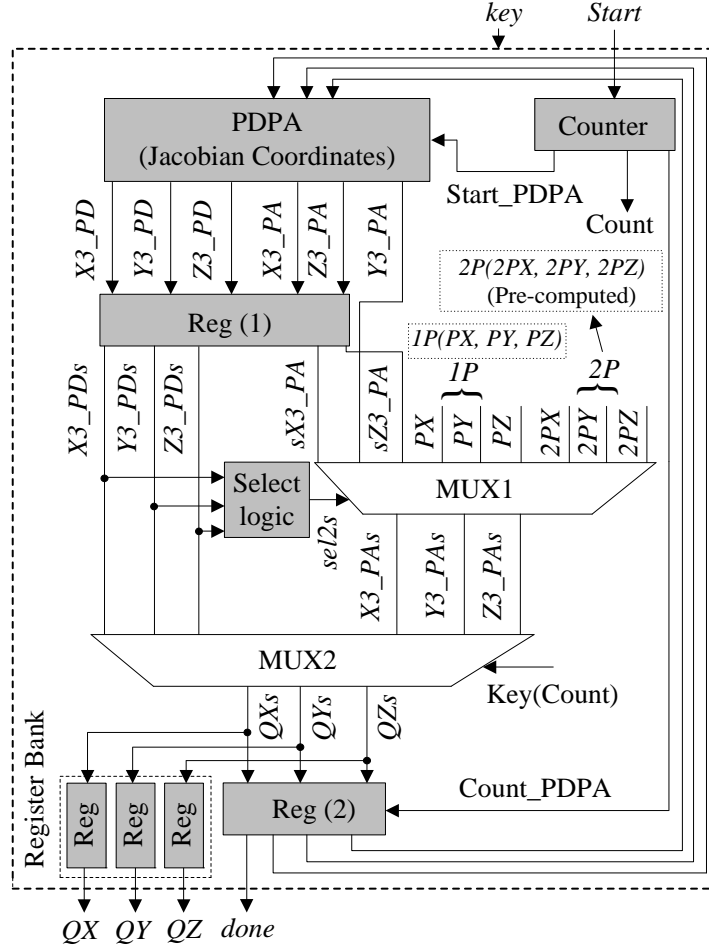


Figure 7.7: Detailed hardware architecture of proposed ECPM in Jacobian coordinates.

7.6.2 Jacobian-to-Affine Coordinate Conversion

A separate module for the conversion between Jacobian and affine coordinates is desired to improve the functionality of the proposed implementation. The Jacobian-to-affine conversion is defined in (7.10)

$$qx = QX/QZ^2; \quad qy = QY/QZ^3. \quad (7.10)$$

In practice, to convert affine coordinates from projective, at least one field inversion is needed for the elliptic curve point multiplication (ECPM) described earlier. The proposed hardware architecture for this conversion is depicted in Fig. 7.8. It is observed that four

field multiplications, one field squaring, and one field inversion are required for Fig. 7.8(a), and three field multiplications, one field squaring, and two field inversions are required for Fig. 7.8(b). The latency for Fig. 7.8(a) and (b) are $2m + 4d + 1$ and $4m + 2d + 2$, respectively. Thus Fig. 7.8(a) gives better performance than Fig. 7.8(b) in terms of area and latency. Our designed modular inversion is also very efficient, which speeds up the computation of ECP.

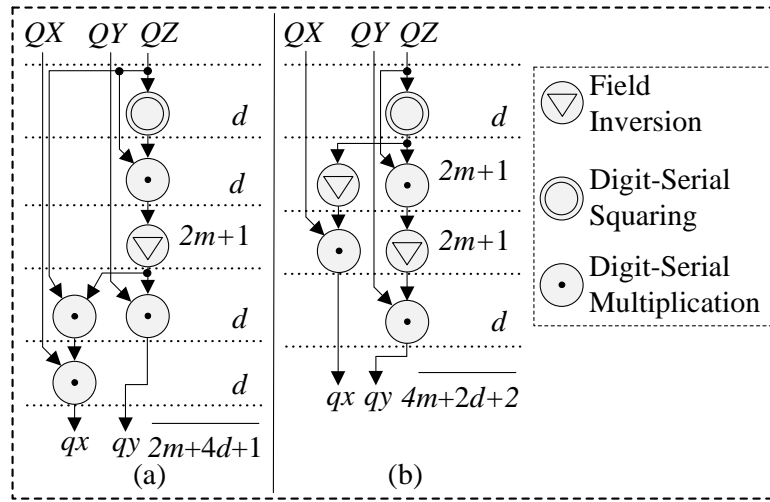


Figure 7.8: Proposed hardware architecture of Jacobian to affine conversion (a) using one inversion and (b) using two inversions.

7.6.3 Main Controller of Point Multiplication

The control unit is a finite state machine (FSM) which is mandatory to maintain the execution sequence of the PDPA unit with other units. Fig. 7.9 displays the control unit for point multiplication in Jacobian projective coordinates over $\text{GF}(2^m)$. The controller unit is initiated with the three basic signals ‘Start_PDPA’, ‘Count’, and ‘Count_PDPA’. Firstly, the possible values of the ‘Start_PDPA’ signal are either zero or one, initialized to zero. The other two signals ‘Count’ and ‘Count_PDPA’ are initialized with $m - 2$ and zero, respectively. The ‘Count’ signal determines the bit position of the input (e.g. ‘key’ or

' k '). When 'Count' is $m - 2$ then the 'Count_PDPA' signal is checked; if this is zero then the start signal is checked. When 'start' is one, which happens once per full cumulation

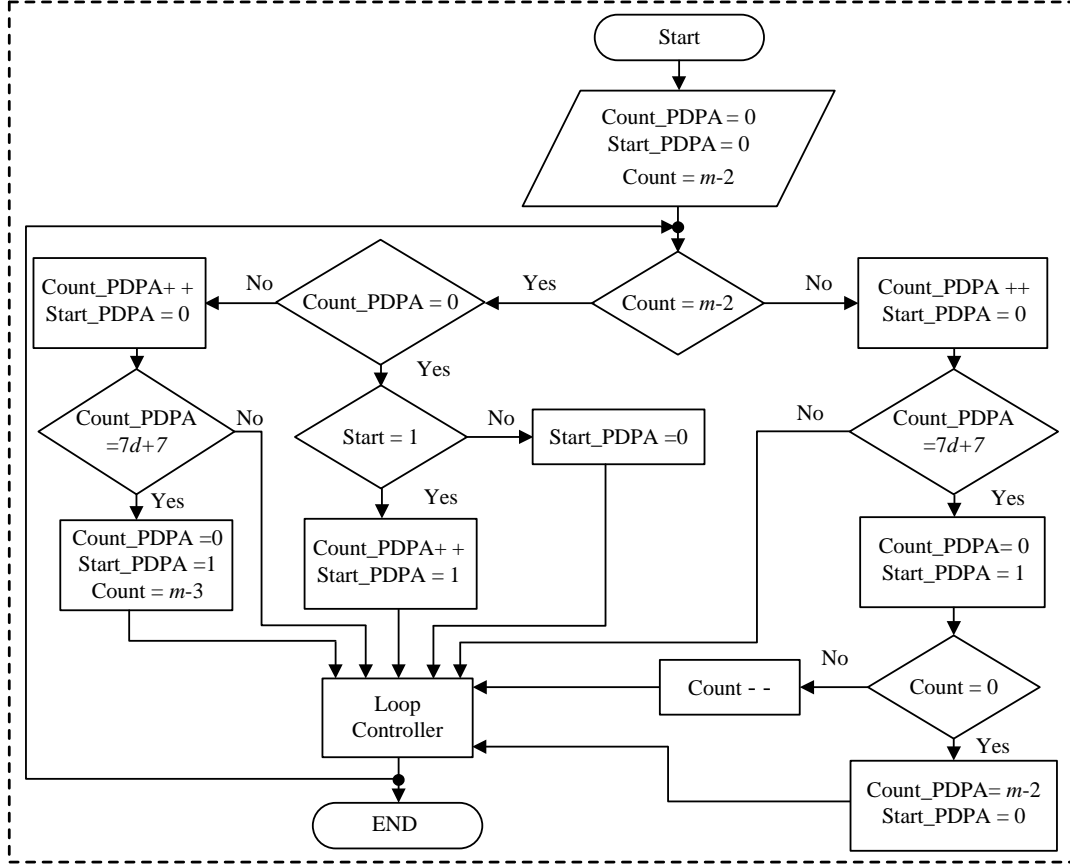


Figure 7.9: Proposed main controller of ECPM in Jacobian coordinates.

of ECPM, then 'Count_PDPA' gives the increment value and the 'Start_PDPA' signal is high in this situation. On the other hand, if 'Count_PDPA' is not zero, then this signal counts till the maximum limit $7d + 7$, as shown in Fig. 7.9. At this condition, the 'Count' signal goes to $m - 3$, and it follows the next iteration of the operation. Consequently, after this operation, the 'Count' signal is less than $m - 2$ and the 'Count_PDPA' signal is incremented till $7d + 7$. The condition for the 'Count_PDPA' signal is checked; it will be triggered to zero when this signal reaches to $7d + 7$ and 'Start_PDPA' enables to high. The computation of point multiplication is achieved after $m - 1$ iterations, where

each iteration needs $7d + 7$ clock cycles. The whole operation is completed only when the ‘Count’ signal goes to zero.

7.6.4 Overall Architecture of ECC Processor

The block diagram for the ECP top module is presented in Fig.7.10. The top module basically consists of two units: ECPM in Jacobian coordinates and Jacobian-to-affine conversion. The point multiplication in Jacobian coordinates is the main unit and is the building block of group operations (PDPA), plus the control unit, pre-computation, select logic, multiplexer, and few register sets. The PDPA architecture in Jacobian coordinates

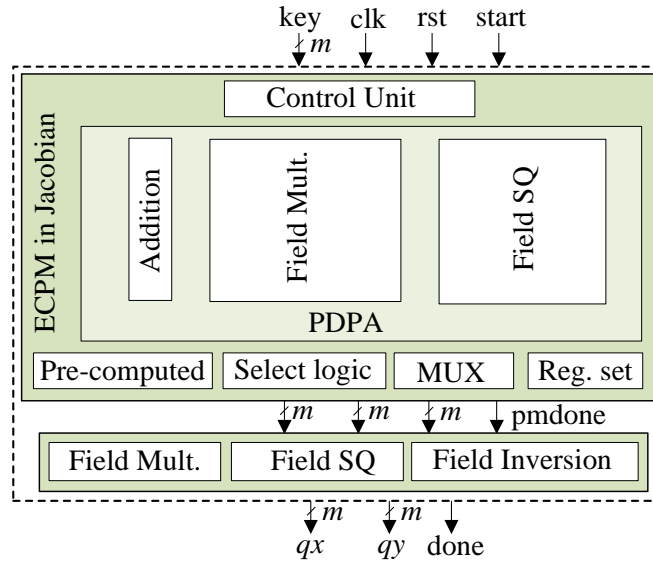


Figure 7.10: Overall block diagram of ECP top module.

needs most of the area, consisting of field addition, multiplication, and squaring. In addition, the PDPA hardware uses a digit-serial multiplier which as described earlier takes $7d + 8$ clock cycles including one cycle for a register. Consequently, the bottom level of Fig.7.10 is called a conversion unit and consists of field arithmetic units such as field inversion, squaring, and multiplication. The Jacobian-to-affine coordinate conversion unit is necessary because, for practical applications, affine coordinates are used. $2m + 4d + 1$

clock cycles are required for conversion. The total number of clock cycles for an ECP top module using digit-serial field multiplication is defined by (7.11). Thus the number of clock cycles for the ECP is computed by

$$\begin{aligned}
&= \text{PM in Jacobian} + \text{Jacobian} - \text{to} - \text{Affine conversion} \\
&= (m - 1) \times \text{PDPA CCs} + (2m + 4d + 1) \\
&= (7md + 8m - 7d - 8) + (2m + 4d + 1) \\
&= (7md + 10m - 3d - 7)
\end{aligned} \tag{7.11}$$

7.6.5 Security Analysis

We have designed a combined PDPA architecture which performs PD and PA operations concurrently, as depicted in Figs 7.5 and 7.6. For this reason, the power consumption pattern for the PDPA hardware will be symmetric in nature. As can be seen from Fig. 7.7, a point multiplication hardware is developed using this combined PDPA architecture. A uniform power consumption profile may be measured throughout the point multiplication computation. From the analysis, we can say that any key value information is difficult to observe from this hardware. Besides, the double-and-add algorithm is secure against timing and simple power analysis (SPA) attacks [90].

7.7 Results and Performance Comparison

This section presents the hardware implementation results of ECPs for both NIST Koblitz and random curves over $\text{GF}(2^m)$. The proposed ECP is implemented on both FPGA and ASIC platforms and the overall performance ECPs is compared to most significant implementations in the literature.

7.7.1 Hardware Implementation Results

The presented ECP is coded in synthesizable VHDL and implemented using Xilinx ISE 14.7 synthesis technologies with an optimized goal of ‘Speed’. The target FPGAs selected are the Xilinx Virtex-6 and Virtex-7 that contain sufficient resources. In addition, the proposed design is synthesized on ASIC 65-nm CMOS technology using Synopsys Design Compiler. The standard logic-cell library for typical process corner (1.2V, 25°C) analysis is utilized for this implementation. The FPGA implementations were simulated using both ModelSim PE and Xilinx ISim. On the other hand, ASIC implementations were simulated using ModelSim/Quarta Sim. A sufficient number of samples were simulated to check the correctness of our implementations.

Table 7.1 depicts the FPGA implementation of ECP for both Koblitz and random curves using bit-serial multiplication. The results of area, clock cycles, time, maximum frequency, area \times time (AT), performance (1/AT), energy, and throughput are reported in Table 7.1. The point multiplication time on a Xilinx Virtex-7 FPGA is between 0.66 and 9.36 ms for all five NIST Koblitz curves. The reported area is between 3065 and 13530 slices only, which shows that the proposed design is area-efficient. The energy consumption was calculated for all of our FPGA-based implementations. It is the product of power and time per ECP. For example, the power consumption per ECP over GF(2^{163}) is 0.784 watts and the timing is 0.66 ms. Thus the energy consumption was computed as 0.52 mJ for the 163-bit ECP. To measure power, first we have mapped our design using the Xilinx ISE, then generated an SAIF file and simulated using ISim. Note that accurate power consumption from an FPGA can be simulated using both VCD or SAIF file. We have used an SAIF file instead of a VCD file because a VCD file is heavier than the SAIF file format. As from the simulator, the switching activity profile was depicted over 95% and the confidence level was shown to be high, which is good enough to get the actual energy consumption for our design. As one can see in Table 7.1, the energy consumption on a

Table 7.1: Performance comparison of FPGA-based ECP using bit-serial field multiplication over NIST Binary Fields $GF(2^m)$

Work	Platform	Field (Length)	Reported Area (slices)	KCycles	Time (ms/ECP) @f (MHz)	AT ¹	Performance ⁴ (1/AT×1000)	Energy ² (mJ/ECP)	TR ³ (kbps)
Virtex-7		163	3065	187.6	0.66@284	2.02	500.0	0.52	247
		233	4754	382.3	1.35@283	6.42	155.7	1.24	173
		283	5567	563.4	2.22@253	12.36	80.9	1.62	127
		409	9126	1175	5.23@225	47.73	20.9	5.10	78
		571	13530	2288	9.36@244	126.64	7.9	11.83	61
This work									
(Koblitz Curves)		163	3116	187.6	0.69@272	2.15	465.0	3.50	236
		233	4025	382.3	1.50@255	6.04	165.5	7.81	155
	Virtex-6	283	5658	563.4	2.49@226	14.09	71.0	13.47	114
		409	8853	1175	5.77@204	51.08	19.6	33.45	71
		571	11673	2288	10.60@215	123.73	8.1	66.75	52

1. AT = Area×Time (Slices×s), 2. Energy = power×time, where power measured from Xilinx Xpower Analyzer, 3. TR = Throughput rate = 1/(time(sec)×Field Length, 4.
Performance = Efficiency = $\frac{1}{Area \times Time} = \frac{1}{AT} = \text{operation/sec/slice}$

Table 7.1: Performance comparison of FPGA-based ECP using bit-serial field multiplication over NIST Binary Fields $GF(2^m)$

Work	Platform	Field (Length)	Reported Area (slices)	KCycles	Time (ms/ECP) @f (MHz)	AT ¹	Performance ⁴ (1/AT×1000)	Energy ² (mJ/ECP)	TR ³ (kpbs)
This work	Virtex-7	163	3454	187.9	0.65@291	2.25	444.0	0.43	251
		233	4790	382.8	1.35@283	6.67	150.0	0.99	173
		283	6534	564.0	2.24@252	14.64	68.0	1.85	126
		409	10046	1176	5.05@233	50.73	19.7	5.54	81
		571	13770	2289	9.27@247	127.65	7.8	12.38	62
(Random Curves)	Virtex-6	163	3159	187.3	0.69@274	2.18	458.7	3.52	236
		233	4653	382.3	1.50@255	6.98	143.2	7.98	155
		283	6186	563.4	2.49@226	15.40	64.9	13.62	114
		409	9100	1175	5.74@205	52.23	19.1	33.66	71
		571	12540	2288	10.35@221	129.79	7.7	65.86	55

1. AT = Area×Time (Slices×s), 2. Energy = power×time, where power measured from Xilinx Xpower Analyzer, 3. TR = Throughput rate = 1/(time(sec)×Field Length, 4.
Performance = Efficiency = $\frac{1}{Area \times Time} = \frac{1}{AT}$ = operation/sec/slice)

Table 7.1: Performance comparison of FPGA-based ECP using bit-serial field multiplication over NIST Binary Fields $GF(2^m)$

Work	Platform	Field (Length)	Reported Area (slices)	KCycles	Time (ms/ECP) @f (MHz)	AT ¹	Performance ⁴ (1/AT×1000)	Energy ² (mJ/ECP)	TR ³ (kpbs)
[128]	Virtex-2	163	3863	54.1	2.26@24	8.73	114.5	0.74	72
[161]	Spartan-3	163	1077	300.7	1.94@155	2.09	478.5	–	84
[162]	Quartus-II	233	8799	447.5	1.62@276	14.25	70.2	–	144
[126]	FLEX10KE	163	6913	640.7	14.9@43	103.00	9.7	–	11
[127]	Spartan-3	131	1180	121.3	1.72@71	2.03	492.6	–	76
		163	1180	190.4	2.70@71	3.19	313.5	–	60
		283	1180	585.2	8.30@71	9.79	102.2	–	34
		571	1180	3622.3	51.38@71	60.63	16.5	–	11
[118]	Virtex-2	163	12861	347.4	2.07@168	26.62	37.6	–	79
[163]	Spartan-6	163	1844	2439	195.1@12.5	359.76	2.8	–	0.84

1. AT = Area×Time (Slices×s), 2. Energy = power×time, where power measured from Xilinx Xpower Analyzer, 3. TR = Throughput rate = 1/(time(sec)×Field Length,

4. Performance = Efficiency = $\frac{1}{Area \times Time} = \frac{1}{AT}$ = operation/sec/slice)

Table 7.1: Performance comparison of FPGA-based ECP using bit-serial field multiplication over NIST Binary Fields $GF(2^m)$

Work	Platform	Field (Length)	Reported Area (slices)	KCycles	Time (ms/ECP) @f (MHz)	AT ¹	Performance ⁴ (1/AT×1000)	Energy ² (mJ/ECP)	TR ³ (kpbs)
[164]	Virtex-4	163	1095	201.8	1.35@150	1.48	675.7	–	121
[129]	Virtex-E	163	2490	–	2.55@–	6.35	157.5	–	64
[165]	Spartan-3	163	2396	156.3	1.25@125	3.0	333.3	–	130
[166]	Virtex-2	161	11395	145.8	1.10@133	12.53	79.8	–	146
[134]	Spartan-3	233	–	183.0	2.28@80	–	–	–	102
[167]	Virtex-2	163	8954	83901	0.84@100	7.52	133.0	–	194
		257	10847	210672	2.11@100	22.89	43.7	–	122
[131]	Virtex-E	163	3600	134.1	3.05@44	10.98	91.1	–	53
[136]	Virtex-2	233	–	335.0	3.35@100	–	–	–	70

1. AT = Area×Time (Slices×s), 2. Energy = power×time, where power measured from Xilinx Xpower Analyzer, 3. TR = Throughput rate = 1/(time(sec)×Field Length,

4. Performance = Efficiency = $\frac{1}{Area \times Time} = \frac{1}{AT}$ = operation/sec/slice)

Xilinx Virtex-7 FPGA is between 0.52 and 11.83 mJ for Koblitz curves between K-163 and K-571. The proposed ECP for random curves takes a similar area, clock cycles, and time. But, it takes a little bit more energy than Koblitz curves.

Our proposed digit-serial version ECP is implemented in FPGA, and the results are displayed in Table 7.2. In this paper, 64, 32, and 16-bit digit sizes were selected for the 233-bit ECP. As can be seen from Table 7.2, the required clock cycles decrease with the increasing digit size, on the other hand the area increases. For example, 64-bit digit-serial multiplication needs only 4 clock cycles per 233-bit field multiplication, whereas 16-bit digit-serial multiplication takes 15 clock cycles to implement. Hence, the ECP using the 64-bit digit version has less latency than other digit-serial versions. The performance, which also called the efficiency metric, is the best indicator to say which design is better. As shown in Table 7.2, the performance of different digit versions provides similar results. We can say that ECP using the 64-bit digit version can be used for high-throughput applications and the 16-bit digit version can be used for low area applications. The proposed ECP over $\text{GF}(2^{233})$ using 64-bit, 32-bit, and 16-bit digit-serial field multiplication takes between 37.52 and 111.92 μs on a Xilinx Virtex-7 FPGA and between 39.50 and 119.54 μs on a Xilinx Virtex-6 FPGA, respectively.

Table 7.3 illustrates the proposed ASIC-based implementation of ECP using both bit-serial and digit-serial field multiplication. For the bit-serial version ECP, ASIC results gives better performance than FPGA results in terms of time and energy. On the other hand, a digit-serial version ECP provides similar results to an FPGA, but provides better energy performance. As we can see in Table 7.3, an ASIC-based implementation of ECP using the 16-bit digit-serial version provides better results than all other digit-serial version ECPs. It can compute an ECP over $\text{GF}(2^{233})$ within 41 μs with a gate count 326.9K. Similarly, the time and area for ECP over $\text{GF}(2^{233})$ is 317 μs and 115.5 K gates, respectively. Note that the gate count was computed from the required ECP area divided

Table 7.2: *Performance and comparison of FPGA-Based ECP using digit-serial multiplication over NIST Binary Field $GF(2^{233})$*

Work	Digit size (d)	Platform	Area (slices)	KCycles	Time (μ s/ECP) @f (MHz)	Area \times Time (AT) (slices \times s)	Performance (1/AT)	TR (Kbps)
This work	64	Virtex-7	72937	8.8	37.52@236	2.74	0.36	6210
	64	Virtex-6	84894	8.8	39.50@224	3.35	0.30	5899
	32	Virtex-7	48652	15.3	65.16@236	3.17	0.32	3576
	32	Virtex-6	47062	15.3	68.60@224	3.22	0.31	3396
	16	Virtex-7	25424	26.7	111.92@239	2.84	0.35	2082
	16	Virtex-6	26286	26.7	119.54@224	3.14	0.32	1949

Table 7.2: *Performance and comparison of FPGA-Based ECP using digit-serial multiplication over NIST Binary Field $GF(2^{233})$*

Work	Digit size (d)	Platform	Area (slices)	KCycles	Time (μs /ECP) @f (MHz)	Area \times Time (AT) (slices \times s)	Performance (1/AT)	TR (Kbps)
[110]	32	Virtex-4	2431	93.8	604@155	1.47	0.68	386
[111]	32	Virtex-4	2648	155.9	1093@143	2.89	0.34	213
[116]	32	Virtex-4	4834	211.2	2110@100	10.20	0.098	110
[117]	32		1127	5013.2	73400@68	82.72	0.012	3
	16	Spartan-3	1127	10265.5	150300@68	169.39	0.006	1.55
	8		1127	35516.0	520000@68	586.04	0.0017	0.45
[119]	64	Virtex-2	–	400.5	2670@150	–	–	87
	32		–	900.0	6000@150	–	–	39
[120]	32	Virtex-E	35800	6.0	89@67.9	3.19	0.31	2618
[121]	64	Virtex-E	–	15.0	225@66.5	–	–	1035

Table 7.3: Performance analysis and comparison of ASIC-based ECC processor (ECP) over \mathbb{F}_2^m

Technology	Field (Length)	Reported Area (mm ² /KGs ¹)	KCycles	Time (μ s)	Area \times Time (AT ²)	Energy ³ (μ J)	ATE ⁴	TR ⁵ (kpbs)
This work⁶	163	0.239/115.5	187.6	156	37.2/18.0	19.0	0.7/0.3	1047
	233	0.324/156.1	382.3	317	102.8/49.5	52.2	5.3/2.6	734
	283	0.396/190.6	563.4	468	185.1/89.2	93.4	17.3/8.3	605
	409	0.587/282.1	1175.0	975	572.5/275.2	307.1	175.8/84.5	419
	571	0.825/396.9	2288.0	1899	1567.0/753.8	760.8	1192/573.6	300
This work⁷	163	0.247/119.0	187.9	156	38.5/18.6	19.8	0.8/0.4	1045
	233	0.352/169.5	382.8	318	111.9/53.9	57.7	6.5/3.1	733
	283	0.434/208.7	564.0	468	203.1/97.7	102.3	20.8/10.0	605
	409	0.638/306.5	1176.0	976	622.7/299.1	376.4	234.3/112.5	419
	571	0.905/435.0	2289.1	1900	1719.0/826.5	834.3	1434/689.5	300
This work⁸	233(d64*)	1.83/879.8	8.8	69	126.9/61.0	48.8	6.2/3.0	3377
	233(d32*)	1.16/557.7	15.3	52	60.7/29.2	33.3	2.0/1.0	4480
	233(d16*)	0.68/326.9	26.7	41	27.8/13.4	7.1	0.2/0.1	5683

1. KGs = Kilo-Gates, 2. AT = (mm² $\times\mu$ s)/(KGs \times ms), 3. Energy (μ J/ECC processor) = power \times time, 4. ATE = Area \times Time \times Energy = (mm² $\times\mu$ s \times mJ)/(KGs \times ms \times mJ), 5.

TR = Throughput rate, 6. For Koblitz Curves, 7. For Random Curves, 8. Using digit-serial polynomial multiplication, * d64, d32, and d16 mean digit size 64-bit, 32-bit, and 16-bit, respectively

Table 7.3: Performance analysis and comparison of ASIC-based ECC processor (ECP) over \mathbb{F}_2^m

Technology	Field (Length)	Reported Area (mm ² /KGs ⁻¹)	KCycles	Time (μs)	Area×Time (AT ²)	Energy ³ (μJ)	ATE ⁴	TR ⁵ (kbps)
[73] 90-nm	163	1.12/313.0	62.5	260	291.2/81.4	14.0	4.1/1.1	627
	233	1.12/313.0	124.3	520	382.4/162.8	34.0	19.8/5.5	448
	283	1.12/313.0	181.3	760	851.2/237.9	55.0	46.8/13.1	372
	409	1.12/313.0	372.5	1580	1769.6/494.5	141.0	249.5/69.7	259
[168] 0.13-μm	160	1.35/179.0	154.8	1000	1350.0/179.0	32.8	44.3/5.9	160
[149] 0.13-μm	163	2.34/332	182.6	440	1030/146.1	61.0	62.8/8.9	371
	283	2.34/332	518.8	1250	2925.0/415.0	221.0	646.4/91.7	226
	409	2.34/332	937.9	2260	5288.4/750.3	566.0	2993/424.7	181
	571	2.34/332	2033.5	4900	11466/1626.8	1360.0	15594/2212.5	116

1. KGs = Kilo-Gates, 2. AT = ($\text{mm}^2 \times \mu\text{s}$)/(KGs \times ns), 3. Energy (μJ /ECC processor) = power \times time, 4. ATE = Area \times Time \times Energy = ($\text{mm}^2 \times \mu\text{s} \times \text{mJ}$)/(KGs \times ns \times mJ), 5. TR = Throughput rate, 6. For Koblitz Curves, 7. For Random Curves, 8. Using digit-serial polynomial multiplication, * d64, d32, and d16 mean digit size 64-bit, 32-bit, and 16-bit, respectively

Table 7.3: *Performance analysis and comparison of ASIC-based ECC processor (ECP) over \mathbb{F}_2^m*

	Technology	Field (Length)	Reported Area (mm^2/KGs^1)	KCycles	Time (μs)	Area \times Time (AT^2)	Energy ³ (μJ)	ATE ⁴	TR ⁵ (kbps)
[169]	0.13- μm	160	1.44/169.4	54.3	372	535.7/63.0	30.5	16.3/1.9	430
[150]	0.18- μm	163	2.10/69	228.1	1890	3969/130.4	257.0	1020/33.5	86
[151]	0.13- μm	163	-/12.5	275.8	244000	-/3050	9.0	-/27.5	0.65
[152]	0.13- μm	163	-/393	22.0	70	-/27.5	-	-/-	2329
		283	-/393	59.0	187	-/73.5	-	-/-	1513
		571	-/393	450.3	1394	-/547.8	-	-/-	409
[153]	0.35- μm	163	-/16	376.8	27900	-/446.4	-	-/-	6
[131]	0.35- μm	163	-/46	134.0	3050	-/140.3	-	-/-	53

1. KGs = Kilo-Gates, 2. AT = ($\text{mm}^2 \times \mu\text{s}$)/(KGs \times ms), 3. Energy (μJ /ECC processor) = power \times time, 4. ATE = Area \times Time \times Energy = ($\text{mm}^2 \times \mu\text{s} \times \text{mJ}$)/(KGs \times ms \times mJ), 5. TR = Throughput rate, 6. For Koblitz Curves, 7. For Random Curves, 8. Using digit-serial polynomial multiplication, * d64, d32, and d16 mean digit size 64-bit, 32-bit, and 16-bit, respectively

by the NAND gate area, where the NAND gate area for 65-nm technology is $2.08 \mu\text{m}^2$. For example, the gate count of a 233-bit ECP (K-233) using bit-serial multiplication is $0.324 \text{ mm}^2 / 2.08 \mu\text{m}^2 = 324000 \mu\text{m}^2 / 2.08 \mu\text{m}^2 \text{ gates} = 156.1 \text{ Kgates}$. As one can see in Table 7.3, the bit-serial version ECP needs less area to implement, but would require more time than the digit-serial version ECP. Therefore, the energy consumption of ECP using a bit-serial version is between 19 and $760.8 \mu\text{J}$ for Koblitz curves (K-163 to K-571) and between 19.8 and $834.3 \mu\text{J}$ for random curves (B-163 to B-571). On the other hand, the energy consumption for 64, 32, and 16-bit digit-serial version ECPs is 48.8, 33.3 and $7.1 \mu\text{J}$, respectively. As can be seen from Table 7.3, some are better in terms of time and energy, some are better in terms of area. For this reason, the $\text{area} \times \text{time} \times \text{energy}$ (ATE) product was calculated to easily compare with other similar work. We can see that ECP using digit-serial field multiplication provides better results in terms of ATE value.

7.7.2 Performance Comparison

We compare our proposed ECP implementations with the most significant work in the literature. Both bit-serial and digit-serial versions of ECP are implemented using combined PDPA hardware on both FPGA and ASIC platforms and their performance compared. The performance comparison of the proposed ECP with related bit-serial version ECPs are presented in Table 7.1. In [128], the authors provide the energy consumption of their proposed ECP over $\text{GF}(2^{163})$. The result shows that they achieved a similar energy-efficient design to our similar 163-bit ECP. Most of the implementations in the literature were implemented for 163-bit ECP only [118, 128, 129, 131, 161, 163–165]. On the other hand, we have reported ECP results for all five NIST binary fields. Our target FPGA is selected as the Xilinx Virtex-7 and Virtex-6 FPGA, whereas in the literature most of the authors were using a lower version of FPGAs. For this reason, a fair comparison is difficult to make. However, we have provided all of our implementation results. The AT and perfor-

mance is computed for our design and compared with related work. The ECP proposed in [127] supports four binary fields over $\text{GF}(2^m)$ including three NIST curves. As can be seen from Table 7.1, our design using a combined PDPA architecture provides more than 50% better performance than those in [127]. Moreover, the FPGA energy consumption of the proposed bit-serial version ECPs are presented in Table 7.1. Furthermore, the results of [118, 126, 127, 129, 131, 134, 136, 161–166] depict the FPGA-based implementations of ECPs, but FPGA energy consumption results are not given.

Table 7.2 demonstrates ECP implementation results using digit-serial polynomial multiplication and compares the performance with related work. The proposed ECP is implemented using a combined PDPA architecture, whereas the available designs in the literature [110, 111, 116, 117, 119–121] were implemented using separate PD and PA operations. Of them, the ECP proposed in [110] presented better performance than others. The results of the ECP in [111, 120] provide similar performance to our design. Though our design needs more area to implement, it is faster than all other similar work. The performance metric shows that higher-digit-based ECP gives better timing, but would require more area. Consequently, the performance or efficiency of ECP using 64, 32, and 16-bit digit serial polynomial multiplication delivers similar results.

ASIC-based implementations and the significant work in the literature for ECP are shown in Table 7.3. We have focused on both FPGA and ASIC technologies, also provide results for both bit-serial and digit-serial ECPs. As can be seen from Table 7.3, the ECPs proposed in [73] and [149] support a maximum of four NIST binary curves. On the other hand, our ASIC-based ECP using bit-serial multiplication supports all five NIST Koblitz and random curves. Besides, our serial version ECP is very efficient in area and energy. The proposed ECC processor using digit-serial polynomial multiplication would require more area to implement, but is faster than all other similar work in the literature. The ATE value for ECP is the best indicator for comparison between two designs, and

is calculated for our design and compared with related work. The Area \times Time \times Energy (ATE) $(\text{mm}^2 \times \mu\text{s} \times \text{mJ}) / (\text{KGs} \times \text{ms} \times \text{mJ})$ value is computed for all the designs which are demonstrated in the second last column in Table 7.3. The ATE value is calculated for ECPs in [73, 149–151, 168, 169]. Of them, the design proposed in [73] provides a lower ATE value than others. However, our proposed digit-serial ECP provides 3 to 100 times $(\text{mm}^2 \times \mu\text{s} \times \text{mJ})$ or 2 to 55 times $(\text{KGs} \times \text{ms} \times \text{mJ})$ smaller ATE value than those in [73]. We can say that an energy-efficient ECP is achieved for both bit-serial and digit-serial versions. We have a trade-off in our proposed ECP design between area, time, and energy, which is suitable for high-speed as well as low-energy cryptographic applications.

7.8 Conclusion

In this paper, an ECP hardware architecture is presented that computes the point multiplication operation with low latency. The proposed design supports all five Koblitz and random curves over binary fields $\text{GF}(2^m)$ as recommended by NIST. In addition, a novel combined PDPA architecture is proposed for the ECP using both bit-serial and digit-serial multiplication. Moreover, a Jacobian-to-affine conversion unit is designed for the practical realization of the cryptosystem. For this conversion unit, a field inversion architecture is designed using a polynomial basis and bit-serial approach. We have synthesized our design in both FPGA and ASIC technologies. The area and time is computed for all the designs. Furthermore, the energy consumption is also calculated from the power and time for both FPGA and ASIC-based ECPs. The proposed ECP using a bit-serial approach takes between 0.66 and 9.36 ms for Koblitz curves and 0.69 and 10.60 ms for random curves from 163 to 571 bits in a Virtex-7 FPGA. The digit-serial ECP can compute the full operation in between 37.52 and 111.92 μs on a Xilinx Virtex-7 FPGA using 64, 32, and 16-bit digit serial field multiplication. In addition, we have implemented our

design on a Virtex-6 FPGA. The ECP on a Virtex-6 FPGA takes $39.50 \mu s$ using 64-bit digit-serial multiplication. Similarly, an ECP using a 16-bit digit size takes $119.54 \mu s$ in a Virtex-6 FPGA. Furthermore, we propose an ASIC-based ECP using both bit-serial and digit-serial field multiplication. The bit-serial ECP takes energy between 19 and $760 \mu J$ on 65-nm technology for Koblitz curves. Thus, the ASIC performance analysis shows that the higher-digit version reduces the latency of ECP, but needs more area. The digit-size 16-bit provides better performance than the other two digit-serial versions. The ATE comparison shows that our design provides 3 to 100 times ($mm^2 \times \mu s \times mJ$) or 2 to 55 times ($KGs \times ms \times mJ$) better performance than the most significant work in the literature. Hence, the bit-serial approach is better in terms of area, whereas the digit-serial version is better in terms of latency and energy. Finally, we can conclude that our proposed ECP offers better performance than the other significant designs in the binary field.

Chapter 8

FPGA-Based Efficient Modular Multiplication for Elliptic Curve Cryptography¹

8.1 Abstract

Modular multiplication is the backbone for the whole asymmetric cryptographic process. In this paper, we have focused on a high-speed hardware implementation of modular multiplication for public-key cryptography, specially for a high-performance Elliptic Curve Crypto-processor (ECC). The proposed design has been implemented over a prime finite field of size p using the National Institute of Standards and Technology (NIST) recommended standards.

Field-Programmable Gate-Array (FPGA) technology with the VHDL language

¹Published as: Md Selim Hossain and Yinan Kong, "FPGA-Based Efficient Modular Multiplication for Elliptic Curve Cryptography," *International Telecommunication Networks and Applications Conference (ITNAC)*, UNSW, Sydney, Australia, pp. 191-195, 18-20 November, 2015, DOI: 10.1109/ATNAC.2015.7366811.

has been used for this hardware implementation. The computational time of a 256-bit modular multiplication in a modern Xilinx Virtex-7 FPGA is $1.683\ \mu\text{s}$ at frequency $152.709\ \text{MHz}$; in this technology we have implemented an area-efficient hardware design technique which takes only 605 slices for a 256-bit modular multiplication. The required area and time are also very low compared with all other recent designs. The product of area and time (AT) of our design is also nearly 9-98 times better than the related designs. To our knowledge, our implemented modular multiplication over $GF(p)$ provides a better performance than the recent hardware implementations.

8.2 Introduction

Asymmetric or public-key cryptography plays a vital role to pass secured information between different wireless devices. The algorithm associated with public-key cryptography (PKC) should be designed in such a way that it requires a small area with the assurance of high security and throughput. Modular arithmetic operations such as modular addition, subtraction, squaring, multiplication, and inversion or division are vital in data communication systems, coding, mobile appliances, and cryptography, specially public-key cryptography (PKC). A dedicated high-speed modular multiplier is mandatory to speed up the calculation of an elliptic-curve crypto-processor for practical applications. This can also be used for other popular and secure public-key encryption techniques like the Rivest-Shamir-Adleman (RSA) cryptosystem [8].

Elliptic Curve Cryptography (ECC) [10, 11] is a popular and powerful PKC, and was first proposed in 1985 independently by N. Koblitz and V. Miller. This cryptosystem can provide equivalent security to the traditional RSA cryptosystem with a significantly shorter key length. For example, a 256-bit ECC over a prime field provides the same

level of security as 3072-bit RSA [1, 22]. This smart feature makes ECC very popular for resource-constrained environments. IEEE P1363-2000 [23] has standardized the use of ECC-based key-agreement and the Digital-Signature Algorithm (DSA). Elliptic Curve (EC) domain parameters and standards for $\text{GF}(p)$ are recommended by the U.S. Government organization called the National Institute of Standards and Technology (NIST) [22]. Field-Programmable Gate-Array (FPGA) technology has been used for a hardware implementation of modular multiplication which assures low cost, better performance, shorter design time, greater flexibility of the system. It also give the flexibility to update the algorithms.

To date, several modular multiplication methods have been presented in the available literature such as Montgomery modular multiplication [66], Radix-4 Montgomery modular multiplication [73], Bipartite Modular multiplication [170], Sum of Residues modular multiplication [133, 171], modular multiplication using the core function [137], and so on. Among them, Montgomery modular multiplication is very popular for hardware implementation due to its simplicity. The basic operation of modular multiplication is defined as $Z = (x \times y) \pmod{p}$, where p is a prime. The hardware implementation of modular multiplication is designed for the NIST prime field denoted by $\text{GF}(p)$. A 256-bit modular multiplication over the prime finite field for ECC was developed by Ghosh [75], Duan [172], Fan [173], and Daly [80]. Most researchers use the Montgomery modular multiplication method for hardware implementation. A Radix-4 Montgomery modular multiplication method over the prime field was proposed and implemented by Lee [73]. Plenty of modular multiplication methods for hardware implementation are available, but Montgomery modular multiplication is the easiest and is also area-efficient from a hardware implementation perspective. However an area-efficient, high-speed design of modular multiplication is still needed for faster implementation of Elliptic Curve Cryptosystems (ECC).

Our main target is to implement a high-performance modular multiplier which will

perform ECC operations with a very low area and latency. For this, most of our effort has been concentrated on improving the modular multiplication operations which are suited for an ECC processor. Efficient algorithmic reformulations underlying the prime field and architectural optimization schemes are explored to improve the operational speed [132].

In this work, a high-speed FPGA-based design is presented using an efficient Montgomery modular multiplication algorithm with hardware architecture, and it is the fastest hardware implementation over the NIST prime field. The rest of the paper is organized as follows. The mathematical background related to ECC and modular multiplication over the prime field are described in Section 8.3. The algorithm with hardware architecture is presented in Section 8.4. FPGA-based implementation results and a comparison with related designs are given in Section 8.5. Finally, a summary of our work is given in Section 8.6.

8.3 Preliminaries

In this section, a brief introduction to the prime field elements and abstract algebra relevant to modular multiplication and ECC which have been used in our hardware implementation are presented. The prime field $\text{GF}(p)$ is the finite field whose elements $x, y \in \text{GF}(p)$ are all the integers between 0 and $(p - 1)$ inclusive, where p is the prime.

8.3.1 Elliptic-Curve Cryptography

Elliptic-Curve Cryptography (ECC) is a powerful Public-Key Cryptography algorithm, and nowadays it is very popular due to the smaller field size. ECC can be implemented in either prime fields $\text{GF}(p)$ or binary fields $\text{GF}(2^m)$. But a prime field will be the emphasis of this work due to the use of efficient finite-field modular multiplication. An elliptic curve

E over $\text{GF}(p)$ in affine coordinates is the set of solutions for an equation such as

$$y^2 = x^3 + ax + b \quad (8.1)$$

where $x, y, a, b \in \text{GF}(p)$ with

$$4a^3 + 27b^2 \neq 0.$$

The coefficients $a, b \in \mathbb{F}_p$ specifying an elliptic curve $E(\mathbb{F}_p)$ are defined by (8.1). The number of points on the elliptic curve E is represented by $\#E(\mathbb{F}_p)$. It is defined over \mathbb{F}_p as nh , where n is the prime order of the curve, and the integer h is a co-factor such as $h = \#E(\mathbb{F}_p)/n$ [2, 10, 11].

In 2000, FIPS-2 was recommended with 10 finite fields: 5 prime fields (\mathbb{F}_p), and 5 binary fields (\mathbb{F}_{2^m}). The prime fields are $\mathbb{F}_{192}, \mathbb{F}_{224}, \mathbb{F}_{256}, \mathbb{F}_{384}$ and \mathbb{F}_{521} [2]. According to NIST, smaller field sizes can be used in ECC than in RSA and Diffie-Hellman (DH) systems at equivalent security levels. This makes ECC a promising branch of public-key cryptography [2, 33].

The implementation hierarchy of the ECC operations over a prime finite field is presented in Figure 8.1. ECC protocols are the building blocks of EC scalar or point multiplication (ECSM), EC group operations and finite-field modular arithmetic. The top level of the ECC cryptosystem contains ECC protocols like EC-DH (EC-Diffie-Hellman) key exchange, EC-DSA (EC-Digital Signature Algorithm). The second level contains ECSM, which is the series of EC group operations like elliptic curve point addition (ECPADD) and elliptic curve point doubling (ECPDBL). The third level comprises ECPADD and ECPDBL, which are called elliptic-curve group operations. These are the series of the finite-field modular arithmetic operations such as modular addition, subtraction, multiplication, squaring, and inversion. The finite-field modular arithmetic units are the bottom or fourth level in the hierarchy. For this finite-field modular arithmetic, modular multi-

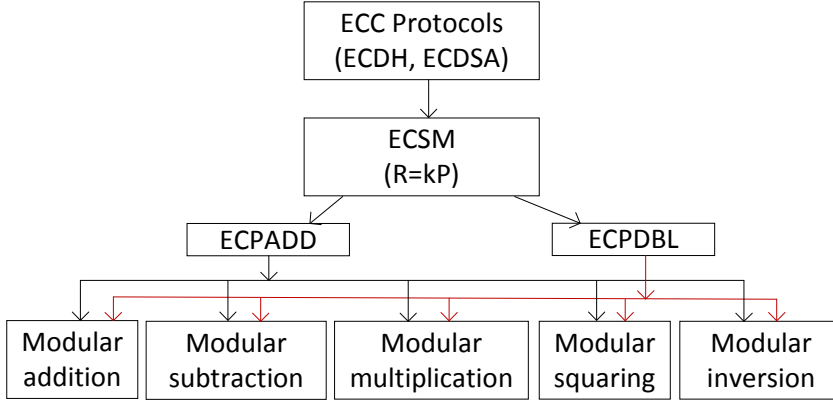


Figure 8.1: *Implementation hierarchy of the ECC operations over $GF(p)$.*

plication is the most crucial for the overall performance of the ECC processor because most of the clock latency depends upon the operation of this modular multiplication.

8.3.2 Modular Multiplication over $GF(p)$

All modular arithmetic operations over the prime field $GF(p)$ are accomplished by integers modulo p consisting of the integers between 0 and $p - 1$, where p is a prime number. Let, $m \simeq \lceil \log_2 p \rceil$ be the bit length of p . The bit length of our design is 256 because our aim is to implement 256-bit modular multiplication for the ECC. Besides, in practice a 256-bit ECC system over a prime field is very useful for modern security purposes. In order to execute the ECSM, modular multiplication is mandatory because this is the fundamental and most crucial operation for ECC over the prime field. We denote this finite field by \mathbb{F}_p and call p the modulus of \mathbb{F}_p . The basic modular multiplication operation can be presented as

$$Z = (x \times y) \pmod{p} \tag{8.2}$$

where p is a prime. A basic example of modular multiplication is as follows, consider $x = 28, y = 20$, and modulus $p = 29$, then

$$\begin{cases} z = (x \times y) \pmod{p} \\ = (28 \times 20) \pmod{29} = (19 \times 29 + 9) \pmod{29} = 9. \end{cases}$$

There are different methods for hardware implementation of modular multiplication presently available. But we have used the Montgomery modular multiplication method, through which we can avoid the costly trial division by the modulus p and, due to the use of this efficient algorithm, modular multiplication can be performed with a high degree of efficiency. Besides using this method we will design an area-efficient hardware implementation that is mandatory for modern applications of public-key cryptosystems.

8.4 Hardware Architecture of Modular Multiplication over $\text{GF}(p)$ for ECC

This section presents the NIST standards for \mathbb{F}_{256} , Montgomery modular multiplication algorithms along with the hardware architecture for our implementation. All the parameters for NIST elliptic curves over \mathbb{F}_{256} are listed in Table 8.1 [2, 22]. According to the NIST standards, the prime number used for 256-bit modular multiplication is

$$\begin{aligned} p &= 2^{256} - 2^{224} + 2^{192} + 2^{96} - 1 \\ &= 115792089210356248762697446949407573530086143415290314195533631308867 \\ &\quad 097853951. \end{aligned}$$

There have been plenty of methods for modular multiplication but most of these algorithms follow the basic concept of the Montgomery method. In 1985, the well-known Montgomery modular multiplication algorithm [66] was shown to be an efficient method

Table 8.1: *NIST-recommended elliptic curves over \mathbb{F}_{256} [2, 22]*

P-256: $p = 2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$, $a = -3$, $h = 1$

p=0x FFFFFFFF 00000001 00000000 00000000 00000000 FFFFFFFF FFFFFFFF FFFFFFFF

S=0x C49D3608 86E70493 6A6678E1 139D26B7 819F7E90

r=0x 7EFBA166 2985BE94 03CB055C 75D4F7E0 CE8D84A9 C5114ABC AF317768 0104FA0D

b=0x 5AC635D8 AA3A93E7 B3EBBD55 769886BC 651D06B0 CC53B0F6 3BCE3C3E 27D2604B

n=0x FFFFFFFF 00000000 FFFFFFFF FFFFFFFF BCE6FAAD A7179E84 F3B9CAC2 FC632551

x=0x 6B17D1F2 E12C4247 F8BCE6E5 63A440F2 77037D81 2DEB33A0 F4A13945 D898C296

y=0x 4FE342E2 FE1A7F9B 8EE7EB4A 7C0F9E16 2BCE3357 6B315ECE CBB64068 37BF51F5

Algorithm 8.1: Radix-4 Montgomery modular multiplication [73]

Input: $X \equiv xr \pmod{p}$, $Y \equiv yr \pmod{p}$, p and m

Output: $Z = \text{MM}(X, Y) \equiv XYr^{-1} \pmod{p} \equiv xyr \pmod{p}$

Let $V = X$, $Z = 0$, $S = Y$;

for $i = 0$ to $(m/2 - 1)$ **do**

If $m \pmod{2} = 1$ and $i = m/2 - 1$ **then**

$Z \equiv (Z + V_0 * S)/2 \pmod{p}$,

$V = V/2$;

else

$Z \equiv (Z + V_0 * S + V_1 * 2S)/2 \pmod{p}$,

$V = V/4$;

Return Z

for computing modular multiplication without using any trial division. Radix-4 Montgomery modular multiplication is shown in Algorithm 8.1 and was presented by Lee [73]. Algorithm 8.1 gives 50% better performance than Algorithm 8.2 in terms of clock cycles, whereas Algorithm 8.1 requires more area (slices).

Algorithm 8.2 represents the simple or radix-2 Montgomery modular multiplication method. This method calculates the Montgomery product using a series of simple additions and right shifts. A hardware architecture of the Montgomery modular multiplier is presented in Figure 8.2. This method avoids the need for costly trial division by modulus p , and keeps the intermediate result bounded within $(m + 2)$ bits over the whole calculation. The Montgomery modular product is given by equation (8.3):

$$\begin{aligned} R &= \text{Montgpro}(x, y, p) \\ &= x \times y \times 2^{-(m+2)} \pmod{p}. \end{aligned} \tag{8.3}$$

Algorithm 8.2: Montgomery modular multiplication in \mathbb{F}_p [66]

Montgpro(x, y, p)

Input: $x, y \in [0, p - 1]$ and p

Output: Z where $Z = x.y.2^{-m} \pmod{p}$ and $Z \in [0, 2p-1]$

$Z_0 = 0$;

for $i = 0$ to $(m - 1)$ **do**

$q_i = (Z_i + y_i.x) \pmod{2}$;

$Z_{i+1} = (Z_i + y_i.x + q_i.p)/2$;

end for

Return Z

The output of a Montgomery modular multiplier is $2^{-(m+2)}$ times the expected result, where m represents the field size in bits. In order to get the exact result, the output must be multiplied by $(2^{(2m+2)})$ to remove the $2^{-(m+2)}$ which is the extra factor of the output. The size of the adders used for this case must be equal to $(m + 2)$ bits to handle the intermediate result at each step of the iteration. There $(m + 2)$ iterations are required to obtain an output in the range between 0 and $2m - 1$ for multiplicands up to twice m .

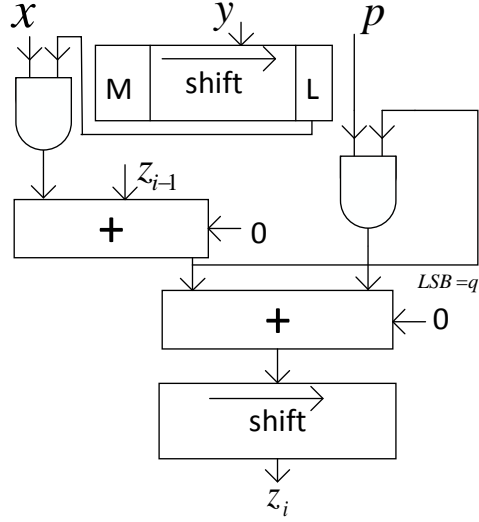


Figure 8.2: A hardware architecture for a modular multiplier [80].

One modular correction is then mandatory to assure that the output is in the range from 0 through $m - 1$, requiring only one extra clock cycle [80].

8.5 Hardware Implementation Results and Performance Analysis

This section presents a hardware implementation of modular multiplication over a prime finite field of size 256. We have implemented our design on a modern 28-nm Xilinx Virtex-7 (XC7V485T-2FFG1761) FPGA device. The implemented design has been simulated using both Modelsim and Isim. The design is synthesized using Xilinx ISE 14.7 synthesis technologies with an optimized goal of 'Speed'. We have also compared the performance of the modular multiplier with related hardware designs. All simulation results are verified using high-level Maple software.

Table 8.2 shows the number of clock cycles required for implementing the proposed modular multiplication over the prime field $GF(256)$. Our 256-bit modular multiplication

Table 8.2: *Performance of Modular Multiplication over \mathbb{F}_{256} on Virtex-7 FPGA.*

Arithmetic Operation	# Clock	Clock Period (ns)	Time (μ s)
Modular Multiplication	$^1m+1$	6.548	1.683

$$^1m \simeq \lceil \log_2 p \rceil$$

Table 8.3: *Device Utilization Summary (Estimated Values) for Modular Multiplication over \mathbb{F}_{256} .*

Logic Utilization	Used	Available	Utilization
Numbers of Slice Registers	773	607200	0.12 %
Number of Slice LUTs	2361	303600	0.78 %
Number of LUT-FF Pairs	772	2362	32 %
Number of BUFG/BUFGCTRLs	1	32	3 %
Number of Slices	605	75900	0.80 %

takes $m + 1$ clock cycles for a m -bit modular multiplication, but our design requires less area than the other designs.

Table 8.3 represents a summary of the estimated values of device utilization. From this table we can see that our implemented modular multiplication over the prime field \mathbb{F}_{256} takes a small amount of resources on the FPGA. The synthesis report shows that our design is area-efficient as it contains only 605 slices.

Table 8.4 represents the 256-bit modular multiplication results and performance comparisons with related designs over the prime field \mathbb{F}_{256} . In this table we have presented information about the frequency (MHz), area (slices), computational time (μ s), product of area and time, and technology. All these parameters have been used for our design and

Table 8.4: *Performance comparison of modular multiplication between our implemented design and other related work over \mathbb{F}_{256} .*

Ref.	Technology	Area (Slices)	Freq. (MHz)	Time (μ s)	Area \times Time (1 AT)
Ours	Virtex-7	605	152.71	1.683	1
Lee [73]	Virtex-II	4843	37	3.46	16.46
Ghosh [75]	Virtex-II	5379	34	7.53	39.78
Duan [172]	Virtex-II	2607	194.56	4.06	10.40
Fan [173]	Virtex-II	3873	93	2.3	8.75
Daly [80]	Virtex-II	5477	14	18.28	98.35

¹The normalization factor for A \times T is $1/0.001018 = 982.3183$, where A is area (slices) and T is time (s). Virtex-II FPGA is

obsolete now

for related designs to make a fair comparison of the performance between them. We have used Virtex-7 FPGA for our hardware platform because Virtex-II FPGA is obsolete now. Radix-4 Montgomery modular multiplication for a dual-field Elliptic Curve Cryptographic processor was given by Lee [73]. All the available results for Lee's method have been presented in the second row of Table 8.4. The available result shows that Lee's method can save 50% on clock cycles because they have used a high-radix modular multiplier, but their design takes more area (slices) than our radix-2 or simple Montgomery modular multiplication. For this case, the area (slices) and computational time for a 256-bit modular multiplication are 4843 and 3.46μ s respectively. For their implementation, they have used a Virtex-II FPGA device. A modular multiplication over the prime field for an elliptic curve scalar multiplier is implemented by Ghosh [75]. The hardware resources of their design contain 5379 slices and the computation time is 7.53μ s to achieve a 256-bit modular multiplication. Their modular multiplication takes k clock cycles for k -bit modular multiplication, which is almost the same as our design, but we have achieved a more

area-efficient design. A 256-bit modular multiplication on a Virtex-II FPGA over the field \mathbb{F}_{256} is presented by [172], [173], and [80], and their implemented modular multiplication requires $4.06 \mu s$, $2.30 \mu s$, and $18.28 \mu s$ respectively. However, our implemented modular multiplier over \mathbb{F}_{256} on a Xilinx Virtex-7 FPGA takes only 605 slices and $1.683 \mu s$ time.

To the best of our knowledge, the best indicator for efficient design is the product of area and time (AT). For this reason, we have calculated the product of area and time (AT) for all the available designs, and make a comparison of their AT with our own design. All these results have been presented in column six of Table 8.4. This column shows that the Daly method gives the worst performance of all the available methods. Of the Ghosh, Duan, Fan, and Daly methods, the method proposed by Fan gives the best performance in terms of AT. However Fan's method has 8.75 times the AT of our design. This result shows that, in terms of the AT parameter, our design gives a far better result than the other available designs presented in Table 8.4.

The required hardware resources of their design are also very high compared to our implementation. We have also achieved a higher clock frequency and less computation time than all the other designs. Our FPGA-based efficient modular multiplication is well suited for a 256-bit ECC processor due to its small area (slices).

8.6 Conclusion

In brief, we have implemented a high-performance modular multiplier over \mathbb{F}_{256} which can be used for efficient operations of elliptic curve scalar multiplication (ECSM), which is the key operation of an ECC processor. The Montgomery modular multiplication method was used for our efficient hardware implementation along with the NIST primes. The implemented hardware is optimized by using different optimization techniques for getting high performance on an FPGA compared to all the related designs. Our design

on a Xilinx Virtex-7 FPGA takes only $1.683 \mu s$ to perform 256-bit modular multiplication, currently the fastest available hardware implementation. Our implemented design is also area-efficient as it contains only 605 slices. The product of area and time (AT) is also nearly 10-100 times as good as the referenced 256-bit modular multiplication methods. The design was simulated using Modelsim PE and all the results verified using high-level Maple software. From the overall performance analysis and comparisons of different methods for modular multiplication over the prime field \mathbb{F}_{256} , we can conclude that our 256-bit modular multiplier provides better performance than all the other designs in terms of the area and the timing.

Chapter 9

High-Performance FPGA

Implementation of Modular Inversion

over \mathbb{F}_{256} for Elliptic Curve

Cryptography¹

9.1 Abstract

Modular Inversion over a prime field is an important operation for public-key cryptographic applications. It is the most crucial operation to speed up the calculation of an elliptic curve crypto-processor (ECC), when affine coordinates are used. In this work, the main goal is to implement a fast, high-performance modular inversion for ECC using field-programmable gate array

¹Published as: Md Selim Hossain and Yinan Kong, “High-Performance FPGA Implementation of Modular Inversion over \mathbb{F}_{256} for Elliptic Curve Cryptography,” *2015 IEEE International Conference on Data Science and Data Intensive Systems (DSDIS)*, UTS, Sydney, Australia, pp. 169-174, 11-13 December, 2015, DOI: 10.1109/DSDIS.2015.47.

(FPGA) technology. A binary inversion algorithm with the VHDL has been used for this efficient implementation. Timing simulation shows that the delay for one modular inversion operation in a modern Xilinx Virtex-7 FPGA takes only 2.329 μ s at the maximum frequency of 146.389 MHz. We have implemented an area-efficient design which takes a small amount of resources on the FPGA and needs only 1480 slices. To the best of the author's knowledge, the proposed modular inversion over \mathbb{F}_{256} provides a better performance than the available hardware implementations in terms of the area and the timing.

9.2 Introduction

Public-key cryptography (PKC) plays a vital role to pass the secured information among the different wireless devices. Modular arithmetic operations such as modular addition, subtraction, squaring, multiplication, inversion, and division are vital in data communication systems, coding, mobile appliances, and cryptography specially public-key cryptography (PKC) such as ECC. A high-performance hardware implementation is mandatory for PKC to fulfil the requirements while conserving a good computing performance. Specially a dedicated high-speed modular inverter is mandatory to speed up the calculation of an elliptic curve crypto-processor when affine coordinate systems are used [73, 75, 132, 137].

The ECC [10, 11] and the Rivest-Shamir-Adleman (RSA) cryptosystem [8] are the two most popular and powerful public-key encryption methods for cryptographic applications. However, ECC can provide similar security to the traditional RSA cryptosystem with a significantly shorter key length. This attractive feature makes ECC very popular for different applications such as smart cards, credit cards, pagers, PDAs (Personal Digital Assistants), cellular phones, and web servers [75]. IEEE P1363-2000 [23] has standardized public-key cryptographic techniques, including cryptographic schemes, the use of ECC-

based key-agreement and digital-signature algorithms (DSA). Elliptic curve (EC) domain parameters for ECC are recommended by the U.S. Government organization called the National Institute of Standards and Technology (NIST) [22]. FPGA technology has been used for this implementation due to the greater flexibility of the system.

Numerous methods for computing modular inversion have been presented in the available literature. Two well-known methods are often used: Fermat's Little Theorem (FLT) and some variant of the Extended Euclidean Algorithm (EEA) such as the Binary Inversion Algorithm, Montgomery Inversion Algorithm, and the Unified Inverse Algorithm. We have extensively studied of modular inversion for hardware implementations, but not many FPGA implementations have been found in the available literature. A modular inversion for ECC over \mathbb{F}_p was developed by Lee [73], Ghosh [75], Vliegen [76], McIvor [79], and Daly [80], but the Binary Inversion Algorithm is the easiest and is also faster and more area-efficient from a hardware implementation perspective. A Xilinx Virtex-7 FPGA is used for this implementation; this is an advanced version of FPGA, and is remarkable for its integration of an embedded hardware multiplier [133, 171]

Our main target is to implement an efficient modular inverter for ECC. For this, we have concentrated on improving the operational speed of modular inversion using algorithmic reformulations and architectural optimization schemes. However an area-efficient and high-speed feature of modular inversion is still needed for faster implementation of elliptic curve cryptosystems (ECC) in affine coordinate systems. In this work, a high-performance FPGA-based design has been proposed using an efficient Binary Inversion Algorithm with hardware architecture, and is the fastest hardware implementation.

The outline of this paper is organized as follows. Section 9.3 describes the mathematical background behind ECC and modular inversion. Section 9.4 presents the architecture of modular inversion and its components. Implementation results and a comparison with the available designs are given in Section 9.5. Finally, Section 9.6 summarizes our work.

9.3 Mathematical Background

In this section, a brief introduction to the prime field elements and all mathematics relevant to modular inversion and ECC which have been used in this hardware implementation is presented.

9.3.1 ECC

ECC is one of the powerful public-key cryptography algorithm and nowadays it is very popular due to the smaller field size. ECC can be implemented in either prime fields $\text{GF}(p)$ or binary fields $\text{GF}(2^m)$. An elliptic curve E over $\text{GF}(p)$ in affine coordinates is the set of solutions for an equation such as

$$y^2 = x^3 + ax + b \quad (9.1)$$

where $x, y, a, b \in \text{GF}(p)$ with

$$4a^3 + 27b^2 \neq 0.$$

The coefficients $a, b \in \mathbb{F}_p$ specifying an elliptic curve $E(\mathbb{F}_p)$ are defined by (9.1). The number of points on the elliptic curve E is represented by $\#E(\mathbb{F}_p)$. It is defined over \mathbb{F}_p as nh , where n is the prime order of the curve, and the integer h is a co-factor such as $h = \#E(\mathbb{F}_p)/n$ [2, 10, 11].

Let, $P = (x_1, y_1)$ and $Q = (x_2, y_2)$ are two points on the EC, then point addition (PADD) and point doubling (PDBL) formulae in affine coordinates are given below.

$$\begin{aligned} R(x_3, y_3) &= P(x_1, y_1) + Q(x_2, y_2) \in E, \\ x_3 &= \lambda^2 - x_1 - x_2, \\ y_3 &= \lambda(x_1 - x_3) - y_1, \end{aligned} \quad (9.2)$$

where $\lambda = (y_2 - y_1)/(x_2 - x_1)$ and $P \neq Q$;

$$\begin{aligned}
R(x_3, y_3) &= 2P(x_1, y_1) \in E, \\
x_3 &= \lambda^2 - 2x_1, \\
y_3 &= \lambda(x_1 - x_3) - y_1,
\end{aligned} \tag{9.3}$$

where $\lambda = (3x_1^2 + a)/2y_1$ and $P = Q$;

where $R = 0$ when $x_1 = x_2$ and $y_2 \neq y_1$, or $x_1 = x_2 = 0$. Hence, when $P \neq Q$ we have the PADD operation in (9.2) and when $P = Q$ we have the PDBL operation in (9.3) [2,10,11].

From (9.2), the PDBL operation in affine coordinates requires one modular inversion, one modular addition, three modular subtractions, five modular multiplications, and two modular squarings. Similarly, from (9.3), the PADD operation in affine coordinates requires one modular inversion, six modular subtractions, two modular multiplications, and one modular squaring. However, modular inversion is the most crucial operation among all these modular arithmetic operations in terms of area, complexity and execution time for point operations on an EC in affine coordinates.

Table 9.1: *comparison of Key length for equivalent security of Symmetric-key and public-key Cryptography [2, 33]*

Symmetric-key	Example-Algorithm	RSA/DH	ECC in GF(p)
80	SKIPJACK	1024	192
112	Triple-DES	2048	224
128	AES Small	3072	256
192	AES Medium	8192	384
256	AES Large	15360	521

In 2000, FIPS-2 was recommended with 10 finite fields as standard: 5 prime fields and 5 binary fields[2]. The comparison between symmetric cipher key length and key lengths for public-key cryptography like RSA, Diffie-Hellman (DH), and ECC over prime field

$\text{GF}(p)$ are given in Table 9.1. It demonstrates that smaller field sizes can be used in ECC than in RSA and DH systems at an equivalent security level. For instance, 256-bit ECC gives equivalent security to 3072-bit RSA with significantly smaller keys and area. ECC is many times more efficient than RSA and DH for either public-key operations (such as signature generation and decryption) or private-key operations (such as signature verification and encryption). This makes ECC a promising branch of PKC [2, 33].

9.3.2 Coordinate Systems for EC Point Representation

There are various coordinate systems to represent elliptic curve points but two well-known coordinate systems are often used for ECC: Affine coordinate systems and projective coordinate systems. A point on the EC $E(\text{GF}(p))$ for affine coordinates can be represented by using two elements $x, y \in \mathbb{F}_p$, i.e. $P(x, y)$. In this coordinate system, the EC group operations such as PDBL and PADD require a modular inversion, a time-consuming operation. The modular inversion over a prime field for each group operation can be reduced by using projective coordinate systems. In projective coordinates, a point P on the EC needs three elements $X, Y, Z \in \mathbb{F}_p$, i.e. $P(X, Y, Z)$. In practice, to convert projective to affine coordinates, one modular inversion is still needed for an elliptic curve point multiplication (ECPM) [79]. However, EC group operation need more modular multiplication in projective coordinates which is also a very costly operation for ECC. But we have proposed a high-performance modular inversion that is well suited for ECC operation. There are plenty of projective coordinates in the available literature; a detailed coordinate system is discussed in [2].

9.3.3 Modular Inversion over $\text{GF}(p)$

Modular inversion over the prime field is an expensive operation in ECC hardware. All modular arithmetic operations over prime field $\text{GF}(p)$ are accomplished using integers

modulo p , where p is a prime number consisting of the integers between 0 and $p - 1$. Let, $m \simeq \lceil \log_2 p \rceil$ be the bit length of p . The bit length of this design is 256 because our main target is to implement a 256-bit modular inversion for applications of ECC. Besides, in practice a 256-bit ECC system over a prime field is very useful for modern security applications. In order to execute ECPM in affine coordinates, modular inversion is mandatory. This modular inversion operation is performed modulo p , in a finite field of order p . For any integer x , $x \bmod p$ shall denote the unique integer remainder q , $0 \leq q \leq p - 1$, obtained upon dividing x by p ; this operation is called reduction modulo p . We denote this finite field by \mathbb{F}_p and call p the modulus of \mathbb{F}_p . The inverse of an integer a modulo p is defined as an integer R such that $a.R \equiv 1 \pmod{p}$. This classical definition of the modular inversion operation can be presented as

$$R = a^{-1} \pmod{p} \tag{9.4}$$

where p is a prime and a is an integer. From equation (4), the inverse of a exists if and only if a is relatively prime with p . Therefore, the Greatest Common Divisor (GCD) of a and p must be 1 or $\gcd(a, p) = 1$. A basic example of a modular inverter is as follows: consider

$$\left\{ \begin{array}{l} a = 12 \text{ and modulus } p = 29, \\ \text{then } R = a^{-1} \pmod{p} \\ \quad = 12^{-1} \pmod{29} = 17 \text{ because } 12 \times 17 \pmod{29} = 1. \end{array} \right.$$

There are different methods for computing modular inversion but two well-known methods are often used. The first one is Fermat's Little Theorem (FLT) and the second is based on the extended Euclidean (GCD) algorithm (EEA).

Fermat's Little Theorem (FLT)

This states that

$$a^{p-1} \pmod{p} = 1$$

Hence,

$$R = a^{p-2} \pmod{p} = a^{-1} \pmod{p}.$$

From this method, we can say that the inverse of any integer a over F_p is a^{p-2} . Using the FLT method, the multiplicative inverse is obtained by modular exponentiation. However, modular exponentiation is a very expensive operation for finding the inverse [174].

Extended Euclidean (GCD) algorithm (EEA)

The second method for the finding inverse is based on the extended Euclidean (GCD) algorithm. This method computes the multiplicative inverse of a integer number $a \in F_p$ by finding two variables R, q that fulfill the following relation:

$$aR + pq = \gcd(a, p) = 1. \tag{9.5}$$

From (9.5), the term pq drops due to the modulus p on both sides of the equation, and the inverse of a or the multiplicative inverse is finally obtained as R . There are many variants of EEA reported in the available literature [2]; one efficient algorithm for inversion is based on the binary method, which is well-known as the Binary Inversion Algorithm [2]. The inversion operation may be calculated using the Montgomery Inverse based on Montgomery multiplication and the Unified Inverse Algorithm [73, 79].

9.4 Hardware Implementation of Modular Inverter over $\text{GF}(p)$

This section presents the modular inversion algorithm along with a hardware architecture for this implementation. All the parameters for NIST elliptic curves over \mathbb{F}_{256} are listed in Table 9.2 [2, 22]. According to the NIST standards, the prime number used for 256-bit modular inversion is

$$\begin{aligned}
 p &= 2^{256} - 2^{224} + 2^{192} + 2^{96} - 1 \\
 &= 115792089210356248762697446949407573530086143415290314195533631308867 \\
 &\quad 097853951.
 \end{aligned}$$

Table 9.2: *NIST-recommended elliptic curves over \mathbb{F}_{256} [2, 22]*

P-256:	$p = 2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$	$a = -3$	$h = 1$
p=0x	FFFFFFFF	00000001	00000000 00000000 00000000 FFFFFFFF FFFFFFFF FFFFFFFF
S=0x	C49D3608	86E70493	6A6678E1 139D26B7 819F7E90
r=0x	7EFBA166	2985BE94	03CB055C 75D4F7E0 CE8D84A9 C5114ABC AF317768 0104FA0D
b=0x	5AC635D8	AA3A93E7	B3EBBD55 769886BC 651D06B0 CC53B0F6 3BCE3C3E 27D2604B
n=0x	FFFFFFFF	00000000	FFFFFFFF FFFFFFFF FFFFFFFF BCE6FAAD A7179E84 F3B9CAC2 FC632551
x=0x	6B17D1F2	E12C4247	F8BCE6E5 63A440F2 77037D81 2DEB33A0 F4A13945 D898C296
y=0x	4FE342E2	FE1A7F9B	8EE7EB4A 7C0F9E16 2BCE3357 6B315ECE CBB64068 37BF51F5

A Xilinx Virtex-7 (XC7V485T-2FFG1761) FPGA with VHDL (VHSIC Hardware Description Language) has been used for this hardware implementation. Finite-field modular arithmetic such as modular addition, subtraction, multiplication, squaring, and inversion are the most crucial operations for the overall performance of the ECC processor. In affine

coordinate systems, the PDBL and PADD require a modular inversion, a time-consuming operation. However, we have implemented a very efficient design that may speed up the overall calculation of an ECC. We have used an efficient algorithm for inversion based on the binary method, which is well-known as the Binary Inversion Algorithm shown as Algorithm 9.1 [2]. The modular multiplicative inverse $a^{-1} \bmod p$ or modular inversion over the prime field is accomplished using a series of additions, subtractions, and shifting operations. This algorithm works iteratively, and at every step either u or v decreases by at least one bit length. The result of modular inversion $R = a^{-1} \bmod p$ is achieved after $2m$ iterations, where m is the maximum bit length of p and a .

From Algorithm 9.1, we can see that four registers u, v, x , and y are essential to implement a hardware architecture of inversion over a prime field. These hardware architectures are shown in Figure 9.1. From this algorithm, the calculation of division like $u/2, v/2, x/2, y/2$ and so on depends upon the parity and magnitude comparisons of the m -bit registers named u, v, x , and y . Two multiplexers are used to select u and v and several multiplexers are used to select x and y as appropriate. The Least Significant Bit (LSB) determines (1 indicates odd, 0 indicates even) the parity of any number. But exact comparisons can be attained through full m -bit subtractions, and this contributes a major delay before decisions regarding the next calculation can be made. We have used m -bit carry-propagation adders to execute the additions or subtractions. The implemented designs compute all possible values like $x, x/2, (x+p)/2, (x-y)/2, (x+p-y)/2$ or $y, y/2, (y+p)/2, (y-x)/2, (y+p-x)/2$ simultaneously to save time, and multiplexers are used for selecting the new value of x and y . How to find the new value of u, v, x , and y at each iteration of the main while loop in algorithm 9.1, is shown in Figure 9.1 [174].

Algorithm 9.1: Binary algorithm for inversion in $\text{GF}(p)$ [2]

Input: Prime p and $a \in [1, p - 1]$

Output: $R = a^{-1} \bmod p$

$u = a; v = p; x = 1; y = 0;$

while $u \neq 0$ **do**

while u is even **do** $u = u/2;$

if x is even **then** $x = x/2;$ **else** $x = (x + p)/2;$ **end**

end

while v is even **do** $v = v/2;$

if y is even **then** $y = y/2;$ **else** $y = (y + p)/2;$ **end**

end

if $u \geq v$ **then**

$u = u - v;$ **if** $x > y$ **then** $x = x - y;$

else $x = (x + p - y);$

else

$v = v - u;$ **if** $y > x$ **then** $y = y - x;$

else $y = (y + p - x);$ **end**

end

end

if $u = 1$ **then** $R = x \bmod p;$ **elseif** $v = 1$ **then** $R = y \bmod p;$

end

Return R (At this instance, $R = a^{-1} \bmod p$)

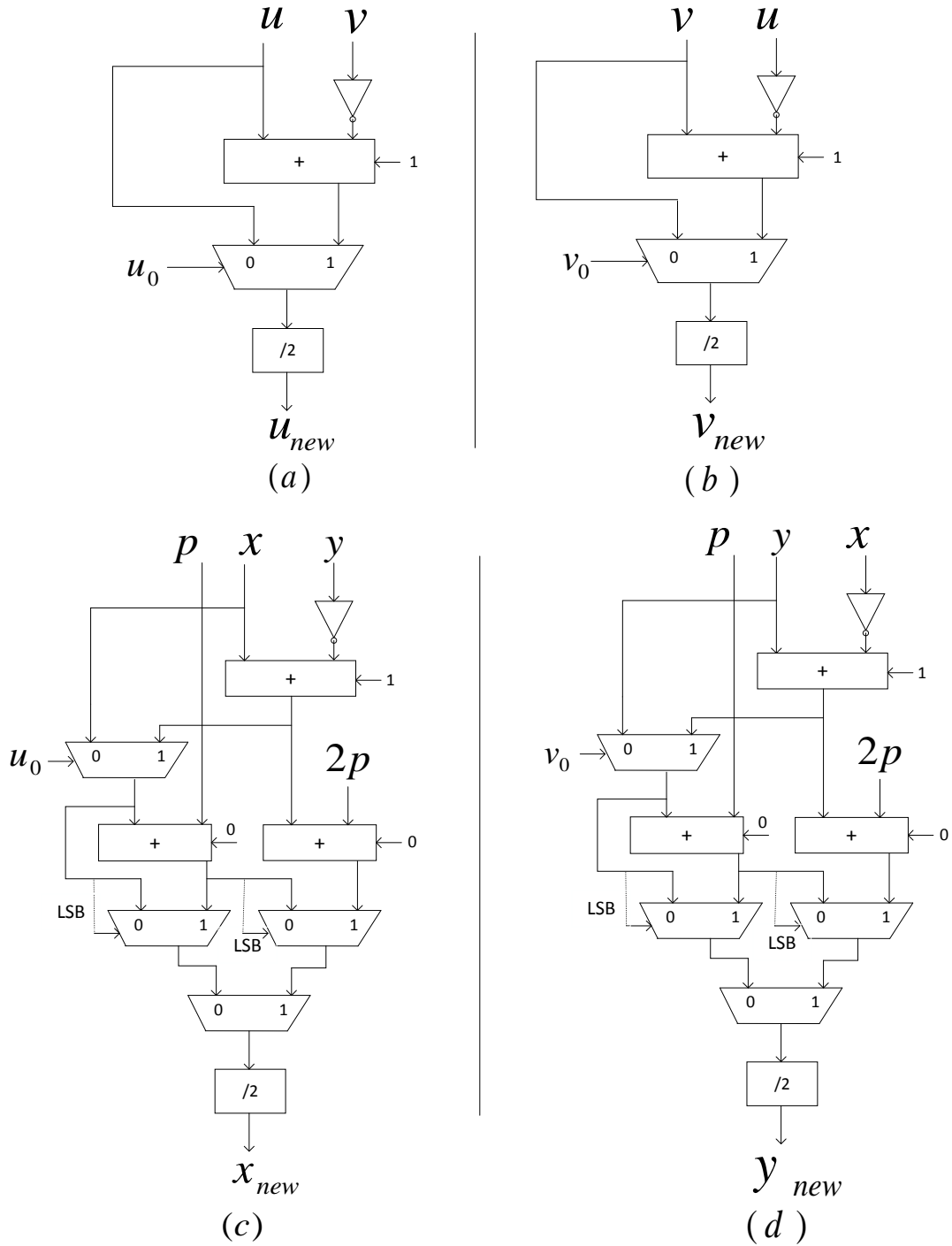


Figure 9.1: A hardware architecture for a modular inverter for finding (a) u_{new} , (b) v_{new} , (c) x_{new} , and (d) y_{new} [174].

9.5 Results and Performance Analysis

The FPGA implementation results of modular inversion over a prime field \mathbb{F}_p of field size 256 is presented in this section. We have implemented this design on a modern 28-nm Xilinx Virtex-7 (XC7V485T-2FFG1761) FPGA device. The proposed design has been extensively simulated using Modelsim PE. The design is synthesized using Xilinx ISE 14.7 synthesis technologies with an optimized goal of 'Speed'. We have also compared the overall performance of the modular inverter with those in the available literature.

Table 9.3: *Device Utilization Summary (Estimated Values) for Modular Inversion over \mathbb{F}_{256} .*

Logic Utilization	Used	Available	Utilization
Numbers of Slice Registers	1551	607200	0 %
Number of Slice LUTs	5868	303600	1 %
Numbers of Fully Used LUT-FF Pairs	1291	6128	21 %
Numbers of BUFG/BUFGCTRLs	1	32	3 %
Numbers of Bonded IOBs	516	700	73 %
Numbers of occupied Slices	1480	75900	1 %

Table 9.3 represents the summary of estimated values of device utilization. From this table we can see that our implemented modular inversion over the prime field \mathbb{F}_{256} takes a small amount of resources on the FPGA. All the simulation results verified using high-level Maple software.

All simulated results and performance comparisons of 256-bit modular inversion over a prime field are given in Table 9.4. It is to be noted that the results provided in the available literature are implemented on different FPGA technologies from our implemented design. In this case, a straightforward comparison is difficult. However, we tried to give all the

Table 9.4: *Performance Analysis of modular Inversion of our design and other related designs over \mathbb{F}_{256} .*

Ref.	Technology	Area (Slices)	Freq. (MHz)	Time (μs)	Area \times Time (AT)
Ours	Virtex-7	1480	146.38	2.329	1
Lee [73]	Virtex-II	9213	37	4.98	13.31
Ghosh [75]	Virtex-II	9146	34	14.60	38.73
Vliegen [76]	Virtex-II	2085	68.17	1160	701.66
McIvor [79]	Virtex-II	14844	40.68	15.22	65.34
Daly [80]	Virtex-II	5477	50	6.4	10.17

[†] The normalization factor for A \times T is $1/0.00344692 = 290.114$, where A is area (slices) and T is time (s). Virtex-II FPGA is obsolete now

information about the frequency (MHz), area (slices), computational time (μs), product of area and time (AT), and technology for fair comparison. A Virtex-7 FPGA has been used for this work as the hardware platform because Virtex-II FPGA is obsolete now. The design proposed by Lee in [73] takes fewer clock cycles than all other available designs, but their design consumes more area (slices) in FPGA than this proposed design. The computational time and area for their design takes $4.98 \mu s$ and 9213 slices respectively on Xilinx Virtex-II FPGA. In [75], the authors proposed a modular inversion for ECC that shows the comparable area (slices) and frequency with Lee's design. However, Lee's design is three times more efficient than Ghosh's design in terms of timing. Ghosh's design takes $2m$ clock-cycles and $14.6 \mu s$ time for m -bit modular inversion. All the available results for Lee's and Ghosh's designs have been presented in the second and third rows respectively of Table 9.4. However, our design provides better performance than their designs in terms of the area and the timing.

A 256-bit modular inversion on a Virtex-II FPGA over the field \mathbb{F}_{256} is presented

by [76], [79], and [80], and their implemented modular inversion requires 1160 μs , 15.22 μs , and 6.4 μs respectively. In [76, 79, 80], their designs consume 2085, 14844, and 5477 slices respectively on a Virtex-II FPGA. Among them, the McIvor design consumes more area than all other available designs and Vliegen's design takes more time than all other available designs. However, this proposed modular inversion over \mathbb{F}_{256} on a Xilinx Virtex-7 FPGA takes only 1480 slices and 2.329 μs time. It may be observed that our design is far better than both designs in terms of the area and the timing.

To the best of our knowledge, the best indicator for efficient design is the product of area and time (AT). For this reason, we have calculated the product of area and time (AT) for all the available designs, and make a comparison of the AT with our own design. All these calculated AT values have been presented in column six of Table 9.4. This column shows that the Vliegen design gives the worst performance among all the available methods. Their design consumes less area but takes more time than all other designs. The AT value of their design is almost 701 times our AT. Of the Lee, Ghosh, and Daly methods, the method proposed by Daly gives the best performance in terms of AT. However, Daly's design takes 10.17 times the AT of our design. This result shows that, in terms of the AT parameter, our design gives a far better result than the other available designs which have been presented in Table 9.4.

The required hardware resources of their designs are also very high compared to our implementation. However, our implemented design efficiently optimizes the area \times time (AT) per bit value for modular inversion. Our design takes an average 1.33 m clock-cycles for m -bit modular inversion. We have also achieved a higher clock frequency and less computation time than all the other available designs. Thus, our designed modular inversion is well suited for a 256-bit ECC processor due to its small area (slices).

9.6 Conclusion

This paper presents a fast, high-performance modular inversion over \mathbb{F}_{256} which can be used for targeted elliptic curve cryptographic applications. The Binary Inversion Algorithm along with the NIST prime has been used for this efficient hardware implementation. The implemented hardware is optimized by using different optimization techniques such as algorithmic reformulations and architectural optimization for getting high performance on an FPGA compared to all the related designs. Our design on a Xilinx Virtex-7 FPGA can achieve a maximum clock frequency of 146.38 MHz and it takes only 2.329 μ s to perform 256-bit modular inversion. So far this is the fastest available hardware implementation. The product of the area and time (AT) of our design is also nearly 10-700 times better than the related designs. The hardware architecture delivers a high-performance inversion operation with low resource usage and contains only 1480 slices. The design was simulated using Modelsim PE and all the results verified using high-level Maple software. From the overall performance analysis and comparisons of different modular inverters over the prime field \mathbb{F}_{256} , it can be concluded that this proposed 256-bit modular inverter provides better performance than all the other designs in terms of the area and the timing.

Chapter 10

High-Performance Elliptic Curve Cryptography Processor Over NIST Prime Fields¹

10.1 Abstract

This paper presents a description of an efficient hardware implementation of an elliptic curve cryptography processor (ECP) for modern security applications. A high-performance elliptic curve scalar multiplication (ECSM), which is the key operation of an ECP, is developed both in affine and Jacobian coordinates over a prime field of size p using the NIST standard. A novel combined point doubling and point addition (PDPA) architecture is proposed using efficient modular arithmetic to achieve high speed and low hardware utilization of the ECP in Jacobian coordinates. This new architecture has been

¹Published as: Md Selim Hossain, Yinan Kong, Ehsan Saeedi and Niras C. Vayalil, “High-Performance Elliptic Curve Cryptography Processor Over NIST Prime Fields,” *IET Computers & Digital Techniques*, Vol. 11, Iss. 1, pp. 33-42, 2017, DOI: 10.1049/iet-cdt.2016.0033.

synthesized both in ASIC and FPGA. A 65nm CMOS ASIC implementation of the proposed ECP in Jacobian coordinates takes between 0.56 ms and 0.73 ms for 224-bit and 256-bit ECC respectively. The ECSM is also implemented in an FPGA and provides a better delay performance than previous designs. The implemented design is area-efficient and this means that it requires not many resources, without any digital signal processing (DSP) slices, on an FPGA. Moreover, the area-delay product of this design is very low compared to similar designs. To the best of the authors' knowledge, the ECP proposed in this paper over \mathbb{F}_p performs better than available hardware in terms of area and timing.

10.2 Introduction

The demand for secure transactions over the network and associated appliances has increased rapidly in recent times. Advanced communication systems require secure information transmission in different areas such as health care, confidential systems, storage, and financial services. For these applications, asymmetric cryptography, or public-key cryptography (PKC), plays a vital role in passing secured information among different devices. PKC offers an important type of technology for key agreement, encryption/decryption, and digital signatures. The algorithm associated with PKC (e.g. elliptic curve cryptography (ECC)) should be designed so that it requires minimal resources with the assurance of high security and throughput. A high-performance hardware implementation is vital for ECC, especially to speed up the calculations in an ECP [75, 175].

10.2.1 Related Work

ECC was first proposed in the mid 1980s by N. Koblitz [11] and V. Miller [10]. The ECC and Rivest-Shamir-Adleman (RSA) [8] cryptosystems are the two most popular and powerful public-key encryption methods for cryptographic applications. However, ECC can provide the same level of security as the traditional RSA cryptosystem with a significantly shorter key. For example, a 256-bit ECC over a prime field provides the same level of security as a 3072-bit RSA. In addition, less memory and hardware resources are required to implement elliptic curve cryptosystems [22, 54, 79, 176]. This smart and attractive feature makes ECC very popular for resource-constrained devices such as smart cards, credit cards, pagers, personal digital assistants (PDAs), and cellular phones. The IEEE [23] and National Institute of Standards and Technology (NIST) [22] have standardized elliptic curve (EC) parameters over $\text{GF}(p)$ and $\text{GF}(2^m)$ for PKC. Moreover, Certicom has provided NIST-recommended EC domain parameters, which are standard for efficient cryptography in SEC2 (Standards for Efficient Cryptography) [54]. FPGA technology has been used for hardware implementation of an ECP; it assures low cost, better performance, shorter design time, and greater system flexibility, for instance updating algorithms.

To date, several FPGA-based hardware implementations of ECPs over a prime field have been proposed [44, 75, 76, 79, 90, 169, 175–181]. The core operation, of elliptic curve scalar/point multiplication (ECSM/ECPM), is defined as $R = k.P$, where the multiplication of an EC point P by a scalar k provides the resultant point R [2]. Scalable/flexible FPGA-based ECPs were proposed by Kung et al. [175] and Ananyi et al. [177], respectively. Both these ECPs support all five prime-field elliptic curves recommended by NIST. ECPs over $\text{GF}(p)$ on an FPGA were proposed in [75, 90], and a parallel architecture unit is used for ECC. In [90], they also synthesized their ECSM architecture on a 130nm CMOS ASIC. They used the double-and-add always algorithm for implementing ECSM.

In [169], Lai et al. proposed a dual-field ECP, implemented on a TSMC 130nm CMOS ASIC. Marzouqi et al. [176] and Vliegen et al. [76] proposed an FPGA-based ECP over an NIST prime field \mathbb{F}_{256} on a Xilinx Virtex-5 and Virtex-II FPGA, respectively. A programmable PKC coprocessor was proposed by Mentens et al. [180] and a reconfigurable modular arithmetic logic unit for PKC was developed by Sakiyama [181]. These ECPs were implemented over prime field \mathbb{F}_{256} on a Xilinx Spartan-3 FPGA. Ahmadi et al. [178] and Fan et al. [179] proposed ECPs over a prime field \mathbb{F}_{192} on 0.13 μm CMOS and Xilinx Virtex-II FPGA, respectively. Although a few high-speed ECPs have been reported over \mathbb{F}_p , most are only area-efficient or are superior only in terms of speed. To have a trade-off between speed and area complexities, a well-designed and efficient ECP is a better option for modern cryptographic applications.

10.2.2 Our Contribution

In this paper, we present a new ECSM with a focus on a system-level description of an efficient FFMA for implementing area-efficient and faster ECP hardware over prime field \mathbb{F}_p . The major contributions of the paper are as follows:

1. We have proposed a novel elliptic curve scalar multiplication (ECSM) architecture using PDPA hardware in Jacobian coordinates; it is the fastest hardware implementation both in ASIC and FPGA.
2. We have developed an architecture for Jacobian-to-affine coordinate conversion, serial-in parallel-out (SIPO), and parallel-in serial-out (PISO) at the top level in order to interface the I/O ports of the ECC processor (ECP) due to the limited number of pins on the FPGA.
3. We have also proposed an efficient ECP in affine coordinates in which ECSM operations are achieved in a very low area (around 9K slices without using any DSP

slices) and latency (20% less than recent implementations).

4. We have proposed a new hardware for a combined EC group operation named point doubling and point addition (PDPA) to develop a high-performance ECP in Jacobian coordinates.
5. We have designed an optimized data-flow architecture for EC group operations such as point doubling (PDBL) and point addition (PADD) for the ECP in affine coordinates.
6. We have proposed and developed high-performance finite-field modular arithmetic (FFMA), for example modular addition, subtraction, multiplication, and inversion algorithms, with hardware architectures over $\text{GF}(p)$ (\mathbb{F}_p). To improve these FFMA and hence the EC group operations, efficient algorithmic reformulations underlying the NIST prime field and architectural optimization schemes are proposed.

10.3 Mathematical Background

In this section, a brief introduction to abstract algebra, field and group theories relevant to the ECP used in this hardware implementation are presented.

10.3.1 Elliptic Curve Cryptography

ECC can be implemented in either prime fields (\mathbb{F}_p) or binary fields ($\text{GF}(2^m)$); both provide almost the same level of security. To design an efficient FFMA, use of an EC over a prime field has been intensively investigated. An EC defined over \mathbb{F}_p provides a group structure that is used to implement cryptographic systems. The group operations are PDBL and PADD. We have implemented all EC operations in both affine and Jacobian coordinates. An elliptic curve E over $\text{GF}(p)$ in affine coordinates is the set of solutions

for an equation such as

$$y^2 = x^3 + ax + b \quad (10.1)$$

where $x, y, a, b \in \text{GF}(p)$ with

$$4a^3 + 27b^2 \neq 0.$$

The coefficients $a, b \in \mathbb{F}_p$ specifying an elliptic curve $E(\mathbb{F}_p)$ are defined by (10.1). The number of points on elliptic curve E is represented by $\#E(\mathbb{F}_p)$. It is defined over \mathbb{F}_p as nh , where n is the prime order of the curve and the integer h is a co-factor such as $h = \#E(\mathbb{F}_p)/n$. A detailed elliptic curve group operation in affine coordinates is found in [2, 10, 11], and described in Section 10.4.3.

Let $P = (x, y)$ be a point in an affine coordinate system; the projective coordinates $P = (X, Y, Z)$ are given by the following:

$$X = x; Y = y; Z = 1. \quad (10.2)$$

The projective point $P = (X, Y, Z)$, $Z \neq 0$ corresponding to the affine point $(P = (x, y))$ is given by

$$x = X/Z^2; y = Y/Z^3. \quad (10.3)$$

Using (10.1), (10.2), and (10.3), the projective form of the Weierstrass equation of the elliptic curve becomes

$$Y^2 = X^3 + aXZ^4 + bZ^6. \quad (10.4)$$

EC group operations formulae in Jacobian coordinates are given in [2, 89]; the ECSM $R = kP$ which is the most important operation in ECC has been implemented in Jacobian coordinates.

10.4 Hardware Architecture over $\text{GF}(p)$ for ECC

This section presents all algorithms and hardware architectures related to FFMA and ECC which are important for the ECP. All parameters and standards for NIST elliptic curves over \mathbb{F}_{224} and \mathbb{F}_{256} are listed in [2].

10.4.1 Modular Multiplier/Squarer over \mathbb{F}_p

In order to implement the ECSM, modular multiplication is mandatory because this is the most crucial operation for the ECP over the prime field, presented as

$$Z = (x \times y) \pmod{p}. \quad (10.5)$$

The Montgomery modular multiplication (MMM) method has been used to implement the ECP in affine coordinates also, because modular multiplication can be performed very efficiently. In 1985 the well-known Montgomery multiplication algorithm [66] was shown to be an efficient method for performing modular multiplication. This method calculates the Montgomery product using a series of simple additions and right shifts. A hardware architecture of the MMM/squarer is presented in Fig. 10.1. This method avoids the need for costly trial division by modulus p , and keeps the intermediate result bounded to $(m+2)$ bits throughout the calculation. The Montgomery modular product is given by

$$\begin{aligned} R &= \text{Montgpro}(x, y, p) \\ &= x \times y \times 2^{-(m+2)} \pmod{p}. \end{aligned} \quad (10.6)$$

The output of a MMM is a factor $2^{-(m+2)}$ times the expected result, where m is the field size in bits. In order to obtain the exact result, the output must be multiplied by $2^{(m+2)}$ to remove the $2^{-(m+2)}$ which is the extra factor of the output. The size of the adders used for this must be $(m+2)$ bits to handle the intermediate result at each iteration; $(m+2)$ iterations are required to obtain an output in the range between 0 and $2m-1$ for

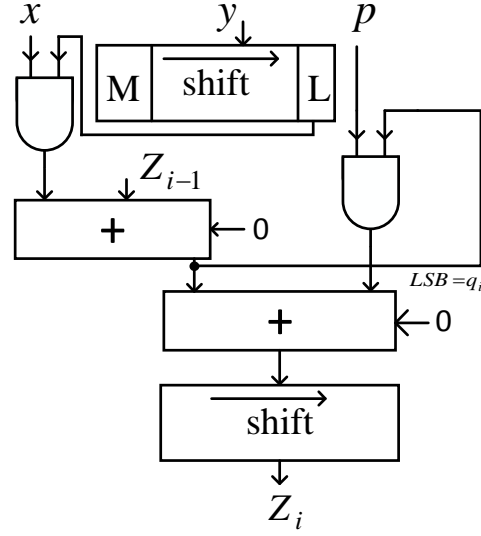


Figure 10.1: Hardware architecture for a modular multiplier/squarer using Montgomery method [80, 182]

multiplicands up to twice m . One modular correction is then required to ensure that the output is in the range between 0 and $m - 1$ inclusive, taking only one extra clock cycle (CC) [69, 80, 182].

An efficient algorithm also proposed for modular multiplication is shown in Algorithm 10.1, and based on interleaved modular multiplication [60]. Fig. 10.2 shows the proposed architecture for modular multiplication over prime field \mathbb{F}_p . In this method, the multiply-by-two operation is performed by a simple left-shift operation. The and-gate operation is: one bit $A[i]$ of the first operand A is multiplied by the whole second operand B bitwise, and then added to the intermediate result. The intermediate result $C3$ is then reduced with respect to the modulus p by two subtractions operating in parallel until the values $C4$ and $C5$ are smaller than the modulus. For doing this, two subtractions and two comparisons are required per iteration. These operations (for $C4$ and $C5$) are running in parallel where $p2$ is pre-computed. Then we need a two-bit multiplexer to select which result is correct. In this architecture, parallelization in operations and pre-computations (to get

Algorithm 10.1: Proposed algorithm for modular multiplication in $\text{GF}(p)$

Input: Prime p and $A, B \in [1, p - 1]$

Output: $C = (A * B) \bmod p$

```

1:   $C = 0$ ;  $p2 = 2 * p$  (pre-computed) ;
2:  for  $i = m - 1$  downto 0 do
3:     $C1 = C$ ;  $C2 = 2 * C1$  (left-shift operation);
4:     $I1 = A[i] * B$  (and-gate operation);
5:     $C3 = C2 + I1s$ ;  $C4 = C3 - p$ ;  $C5 = C3 - p2$  ( $p2 = 2p$ );
6:    if  $C3 \geq p$  then  $C6 = C4$ ;
7:    elseif  $C3 \geq p2$  then  $C6 = C5$ ; else  $C6 = C3$ ; end if  $C = C6$ ;
8:  end for
9:  Return  $C$ 

```

$p2$) are used to calculate all intermediate results. The latency of this method is mostly and-gate, addition, and subtraction. This method requires $m + 1$ cycles to compute the final result of modular multiplication, where m is the maximum bit length of the operands A, B or p and $m \simeq \lceil \log_2 p \rceil$. Therefore, we have designed a high-performance modular multiplier for an ECP in Jacobian coordinates. A modular squarer is similar to a modular multiplier except that only one input is required for a modular squarer rather than the two inputs for a modular multiplier; otherwise all other operations are the same as for modular multiplication.

10.4.2 Modular Inversion over \mathbb{F}_p

Modular inversion over a prime field is the most expensive operation in ECC hardware, and it is mandatory for ECP in affine coordinates. An efficient algorithm has been used for inversion based on the binary method, which is well known as the binary inversion

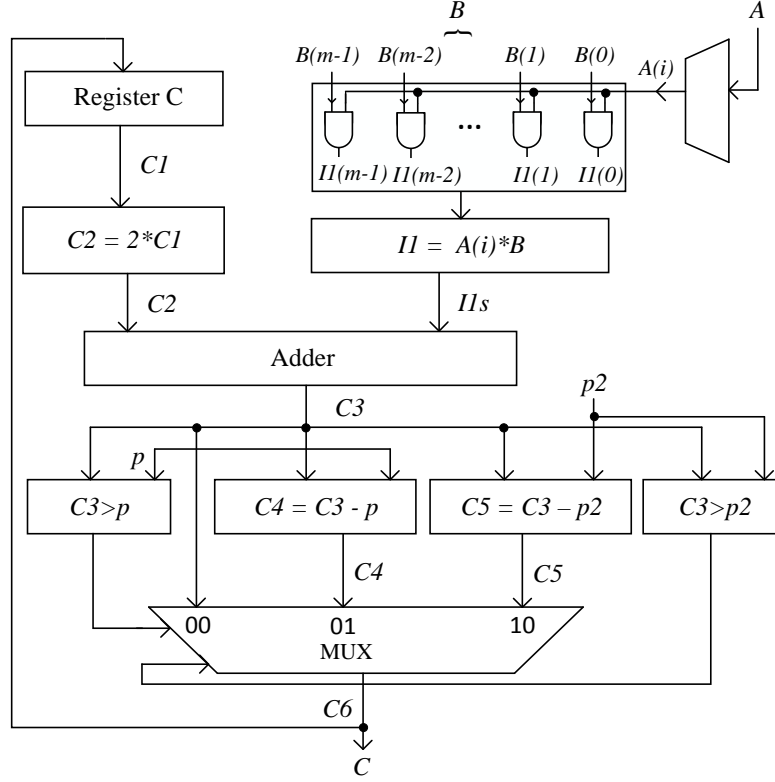


Figure 10.2: Proposed hardware architecture for a modular multiplier/squarer.

algorithm shown in Algorithm 10.2 [2].

The modular inversion over the prime field is accomplished using a series of additions, subtractions, and shift operations [84]. This algorithm works iteratively, and at every step either u or v decreases by at least one in bit length. The result of modular inversion is achieved after $2m$ iterations, where m is the maximum bit length of p and a . The inverse of an integer a modulo p is defined as an integer R such that $a.R \equiv 1 \pmod{p}$. This classical definition of the modular inversion operation can be presented as

$$R = a^{-1} \pmod{p} \quad (10.7)$$

where a is an integer. From (11.8), the inverse of a exists if and only if a is relatively prime with p . Therefore, the Greatest Common Divisor (GCD) of a and p must be 1, or $\gcd(a, p) = 1$. From Algorithm 10.2, four registers - u, v, x , and y - are essential to

Algorithm 10.2: Binary algorithm for inversion in $\text{GF}(p)$ [2]

Input: Prime p and $a \in [1, p - 1]$ **Output:** $R = a^{-1} \bmod p$

```

1:   $u = a; v = p; x = 1; y = 0$  ;
2:  while  $u \neq 1$  and  $v \neq 1$  do
3:    while  $u$  is even do  $u = u/2$ ;
4:      if  $x$  is even then  $x = x/2$ ; else  $x = (x + p)/2$ ; end
5:    end
6:    while  $v$  is even do  $v = v/2$ ;
7:      if  $y$  is even then  $y = y/2$ ; else  $y = (y + p)/2$ ; end
8:    end
9:    if  $u \geq v$  then
10:       $u = u - v$  ; if  $x > y$  then  $x = x - y$ ;
11:      else  $x = (x + p - y)$ ;
12:    else
13:       $v = v - u$  ; if  $y > x$  then  $y = y - x$ ;
14:      else  $y = (y + p - x)$ ; end
15:    end
16:  end
17:  if  $u = 1$  then  $R = x \bmod p$ ; elseif  $v = 1$  then  $R = y \bmod p$ ;
18:  end
19:  Return  $R$  (At this instant,  $R = a^{-1} \bmod p$ )

```

implement a hardware architecture for inversion over a prime field. The calculation of divisions such as $u/2$, $v/2$, $x/2$, and $y/2$ depends upon parity and magnitude comparisons of the m -bit registers named u , v , x , and y . Two multiplexers are used to select u or v and several multiplexers are used to select x or y as appropriate. The Least Significant Bit

(LSB) determines (1 indicates odd, 0 indicates even) the parity of any number. But exact comparisons can be attained only through full m -bit subtraction, and this contributes a major delay before decisions regarding the next calculation can be made. We have used m -bit carry-propagation adders to execute the additions or subtractions. The implemented designs compute all possible values like $x, x/2, (x + p)/2, (x - y)/2, (x + p - y)/2$ or $y, y/2, (y + p)/2, (y - x)/2, (y + p - x)/2$ simultaneously to save time, and multiplexers are used for selecting the new values of x and y . Using this algorithm, one modular inversion over prime field \mathbb{F}_p takes an average $1.33m$ CCs for an m -bit modular inversion.

10.4.3 Proposed EC Group Operations

The EC group operations (PDBL and PADD) are the building blocks of FFMA. There are different techniques for a data-flow architecture such as balancing the architecture, which minimizes the power consumption and reduces the longest path for better performance. Parallelization in operations and pre-computations can be used for further improvement, and we have used all these techniques to increase the throughput rate of EC group operations for a high-performance ECP. As one can see in level 2 in Fig. 10.3a, one inversion, one squaring, and one addition module are operating in parallel. The $2P$ ($2Px, 2Py$) value of PDBL is pre-computed, and used for the PADD module in Fig. 10.3b if $x_1 = y_1$ and $x_2 = y_2$. Using this parallel operation and the pre-computation technique, the data path is reduced for the PDBL and PADD operations. In Fig. 10.3a and b, the latencies of PDBL and PADD are $5m + 12$ and $5m + 10$ respectively, in affine coordinates. Hence, we have designed a high-performance ECSM in affine coordinates for an FPGA. In addition, we propose a novel PDPA technique for computing EC group operations together, shown in Fig. 10.3c. As can be seen from Fig. 10.3a and b, 11 steps are needed to perform the PDBL operation and 9 steps are needed to perform the PADD operation whereas a combined PDPA module needs only 12 steps/levels. Parallelization in operations is used

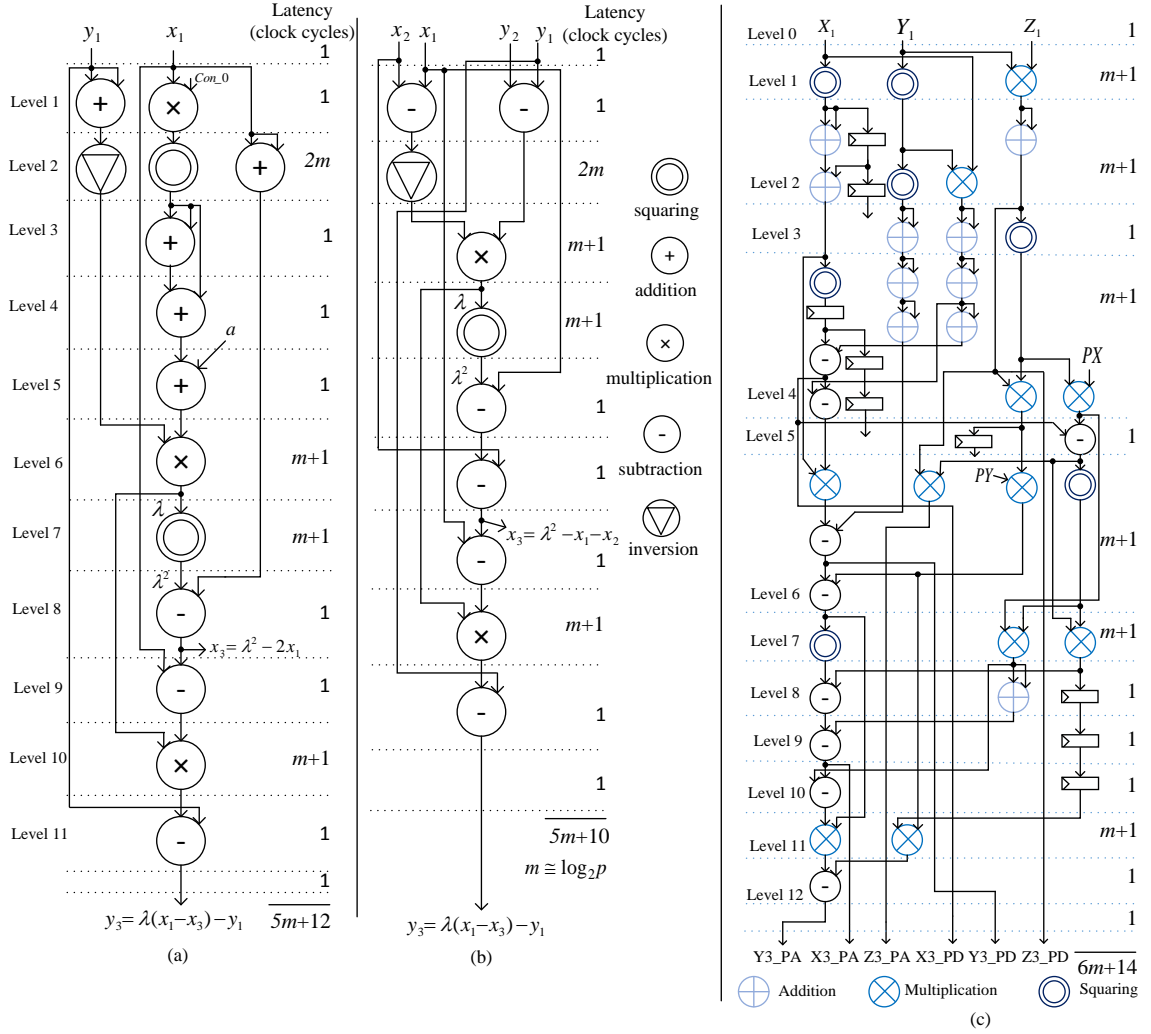


Figure 10.3: Proposed hardware architecture for EC (a) PDBL, (b) PADD, and (c) PDPA.

to increase the latency and throughput of EC group operations. As we can see in level 1 in Fig. 10.3c, two squarings and one multiplication module are operating in parallel. Similarly in level 6, three multiplications and one squaring module are running in parallel. Therefore the number of levels in the data path is reduced, and the overall latency of the PDPA module is reduced to $6m + 14$ in Jacobian coordinates. Using this efficient PDPA hardware, we have designed a high-performance ECP in Jacobian coordinates. Fig. 10.3a, b, and c depicts the proposed architecture of the PDBL, PADD, and PDPA operations

respectively. The costs of PDBL, PADD and PDPA over \mathbb{F}_p are $3\text{MUL} + 2\text{SQ} + 1\text{INV} + 5\text{ADD} + 3\text{SUB}$, $2\text{MUL} + 1\text{SQ} + 1\text{INV} + 6\text{SUB}$, and $11\text{MUL} + 7\text{SQ} + 10\text{ADD} + 9\text{SUB}$ respectively, where MUL, SQ, INV, ADD, and SUB are the costs of modular multiplication, squaring, inversion, addition, and subtraction respectively.

10.4.4 Proposed ECSM

ECSM over \mathbb{F}_p is the key operation of an ECP; it is computationally the most expensive. However we have designed a high-performance ECSM using efficient group operations and FFMA units. The basic operation of ECSM is defined as kP , where k is a positive integer and P is a point on the elliptic curve E defined over a prime field \mathbb{F}_p . Various methods exist for implementing ECSM: the binary method, the Non-adjacent form (NAF) method, and the Montgomery method. The easiest way to implement ECC is the binary method (left to right) [2], shown in Algorithm 10.3. Using this algorithm, on average m PDBL and $m/2$ PADD operations are required for an ECSM in affine coordinates. An ECSM architecture in affine coordinates over \mathbb{F}_p using separate PDBL and PADD is presented in Fig. 10.4. In this ECSM architecture, the PDBL module computes $2Q$ ($Q2x, Q2y$) and the PADD module computes $P + 2Q$ ($Q2px, Q2py$). The comparator module is used for comparison between the output of the PDBL module and the input of the PADD module. When the two inputs of PADD are equal (e.g. $Px = Q2x$ and $Py = Q2y$) then the output of MUX2 is the same as $2P(2Px, 2Py)$ if $\text{key} = 1$. This result occurs when $\text{key} = n + 1$; in this case if the input of the PDBL module is $(n + 1)/2$ then the output of this module is $(n + 1)P = 1P(Px, Py)$ which is the same as the other input of the PADD module. The pre-computed value of $2P$ ($2Px, 2Py$) from the PDBL module is used in MUX1 for this comparison. The output of MUX2 always depends upon the bit pattern of the input key. For a bit pattern of $\text{key} = 0$, the output of PDBL goes directly to the input of MUX2, this appears at the output of MUX2. When $\text{key} = 1$ the output of PDBL goes to the input of

PADD and the output of MUX2 is the same as the output of PADD. In this architecture, $5m + 12$ and $5m + 10$ CCs are required for computing PDBL and PADD respectively in affine coordinates. The total number of CC for computing ECSM in affine coordinates is defined by (10.8). Total average CCs for ECP in affine coordinates

$$\begin{aligned}
 &= m \times (\text{PDBL CC}) + (m/2) \times (\text{PADD CC}) \\
 &= (m(5m + 12) + (m/2)(5m + 10)) \\
 &= (7.5m^2 + 17m)
 \end{aligned} \tag{10.8}$$

Algorithm 10.3: Binary method (Left to right) for point multiplication

Input: $k = (k_{m-1}, \dots, k_1, k_0)_2$, $P(x, y) \in E(\mathbb{F}_p)$

Output: $Q(x, y) = k.P(x, y)$, where $Q(x, y), P(x, y) \in E(\mathbb{F}_p)$

```

1:  $Q = 0$  ;
2: for  $i = m - 1$  to  $0$  do
3:    $Q = 2Q$ ;
4:   if  $k(i) = '1'$  then
5:      $Q = P + Q$  ;
6:   end
7: end for
8: Return  $(Q(x, y))$ 

```

To have a high-performance ECP in Jacobian coordinates, we have proposed a novel ECSM architecture using our proposed PDPA module. Most ECC hardware implementations in the literature have used separate PDBL and PADD modules, and require more CCs than our design. Also, most of the ECC hardware has been implemented in FPGA form. We have found a few hardware implementations targeting an ASIC. The proposed hardware is implemented in both FPGA and ASIC. Fig. 10.5 shows our proposed hardware

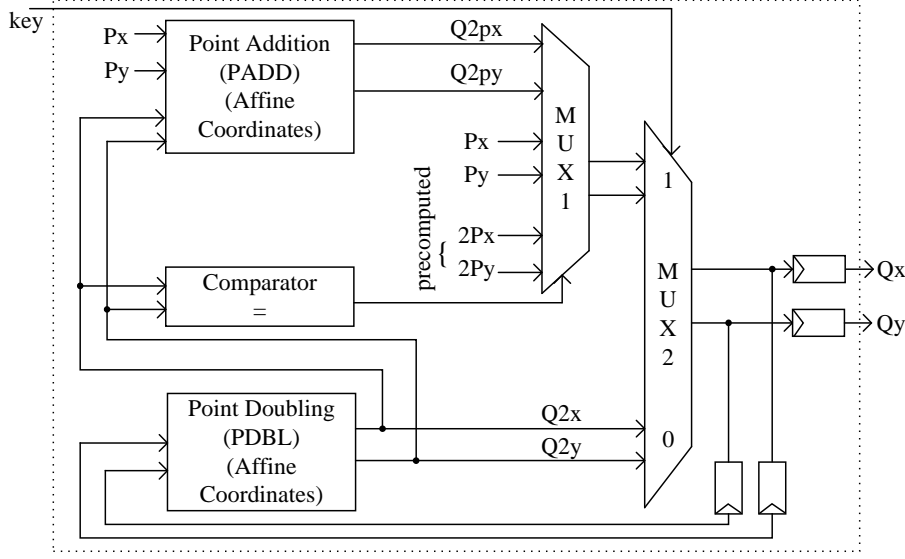


Figure 10.4: Overall hardware architecture of proposed ECSM in affine coordinates for prime field.

architecture of ECSM in Jacobian coordinates over the prime field \mathbb{F}_p . In this architecture, $6m + 14$ CCs are needed for the PDPA module including 2 cycles are needed for the two registers to store all intermediate results. Equation (10.9) represents the total number of CCs for ECP in Jacobian coordinates. As we can see in Fig. 10.5, the PDPA module performs the EC group operations (PDBL and PADD) together, and these results are stored in the register module Register_PDPA. The select logic module generates a 'sel2s' signal for the MUX_1_NEW module. In the MUX_1_NEW module of Fig. 10.5, when 'sel2s = 00' then PADD results from the Register_PDPA module go to the output, which is the same as the input of the MUX_2_NEW module. Similarly, when 'sel2s = 01' then 1P (P_x, P_y, P_z) results, and when 'sel2s = 10' then 2P ($2P_x, 2P_y, 2P_z$) results, which are pre-computed, go to the input of MUX_2_NEW. Hence, the inputs of the MUX_2_NEW module are the PDBL and PADD results and the output of this module is either PDBL or PADD, depending upon the bit pattern of input key. When the bit pattern of key is high then the output is the PADD result otherwise the PDBL result,

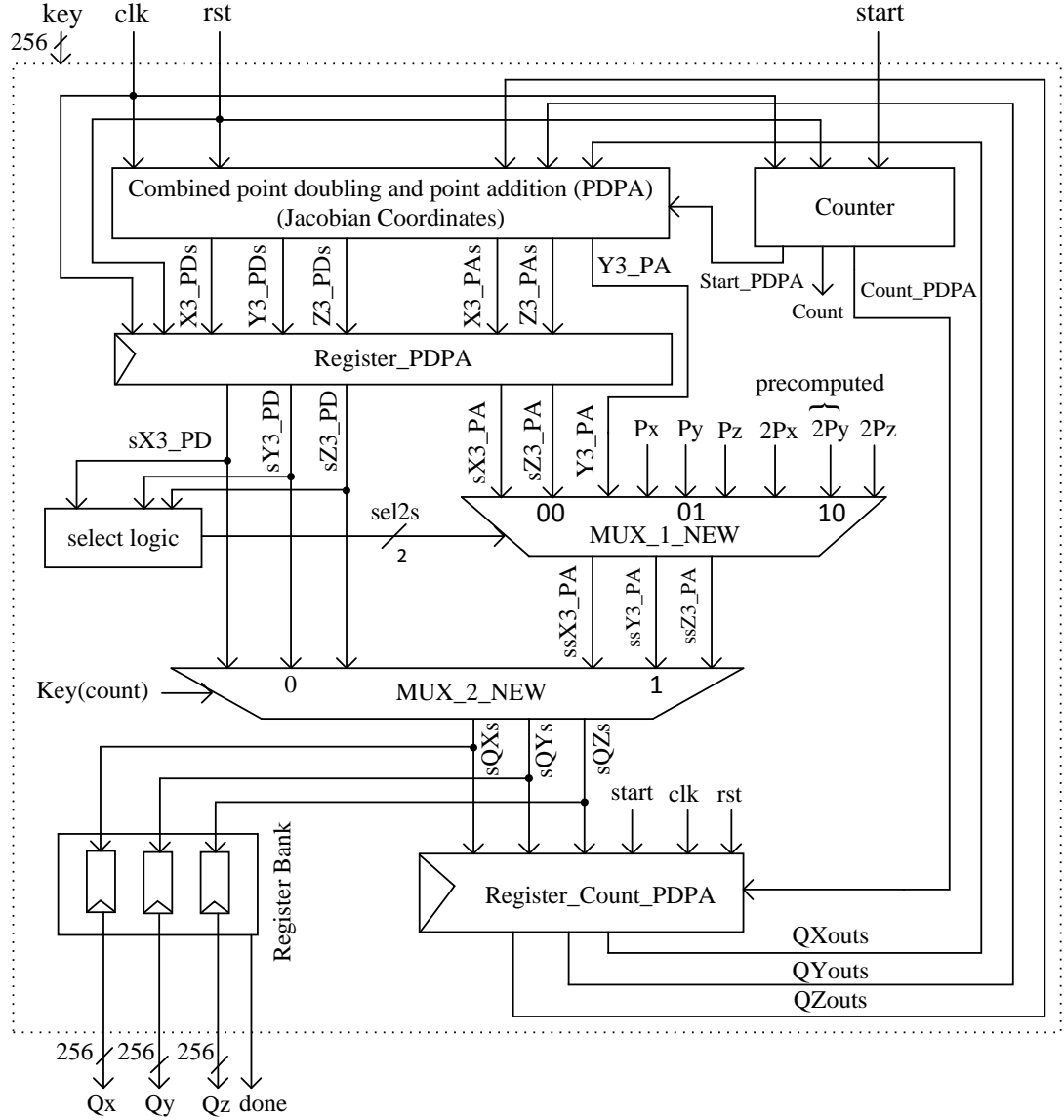


Figure 10.5: Overall hardware architecture of proposed ECSM in Jacobian coordinates.

which is stored in register Register_Count_PDPA. The counter module of this architecture act as a control unit, and decides when the results of this register will be passed to the next input of the PDPA module. There $m - 1$ cycles are required to compute the final result of this ECSM module, where each cycle needs $6m + 14$ CCs (CCs for PDPA). The final results of this ECSM module are in Jacobian coordinates, and need conversion to verify the result in affine coordinates. Fig. 10.6a shows the hardware architecture for

conversion from Jacobian to affine coordinates. As can be seen from this figure, $6m + 6$ cycles are required to get the result in affine coordinates. The complexity of this module is $4\text{MUL} + 1\text{SQ} + 1\text{INV}$, where INV is the cost of modular inversion which is the most expensive operation in the prime field. However, to get the result in affine coordinates, only one modular inversion is needed. We have also designed serial-in parallel-out (SIPO)

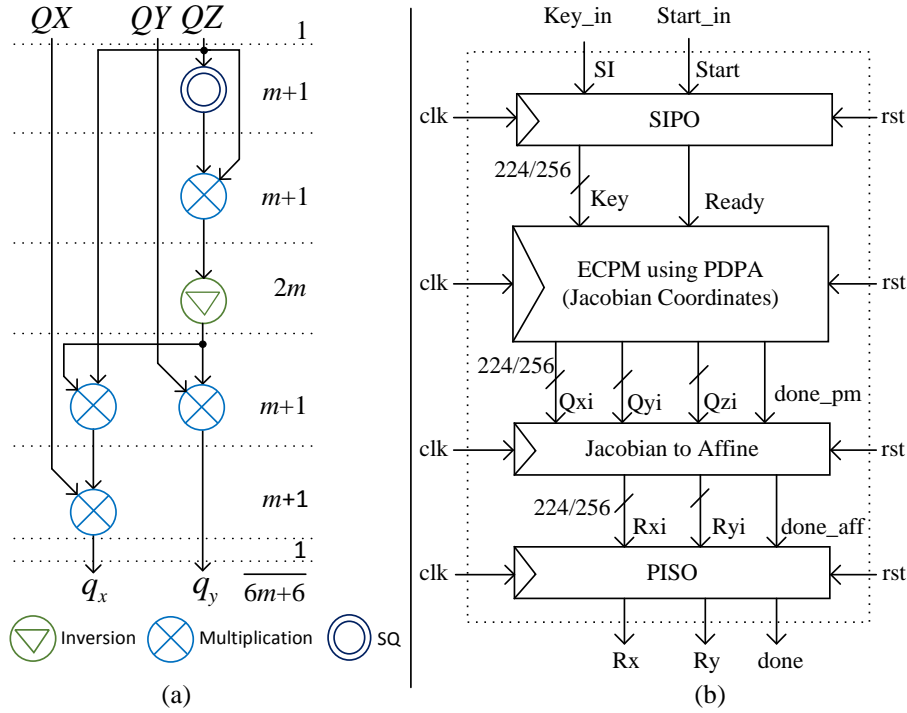


Figure 10.6: Proposed hardware architecture of (a) Jacobian to affine conversion and (b) top module of ECP.

and parallel-in serial-out (PISO) modules to reduce the pin numbers of the top module. The top module of the final ECP is shown in Fig. 10.6b and are the building blocks of four modules, namely SIPO, ECSM, Jacobian to affine, and PISO. The upper module is SIPO which is required to send data serially and receive data in parallel. The second module contains ECSM, which is the main operation of the ECP. The Jacobian to affine conversion module is the third level, and is needed to get the result in affine coordinates. The bottom level is PISO, which sends the final results serially. The top module of ECP

needs only five pins, two for input and three for output. Total CCs for ECP in Jacobian coordinates

$$\begin{aligned}
&= (m - 1) \times \text{PDPA} + \text{Jacobian to Affine} + \text{SIPO} + \text{PISO} \\
&= (m - 1) \times (6m + 14) + (6m + 6) + (m + 2) + (m + 1) \\
&= (6m^2 + 16m - 5)
\end{aligned} \tag{10.9}$$

10.5 Implementation Results and Performance Analysis

This section presents the implementation results for our proposed design. The design has been extensively simulated using both ModelSim PE and ISim, and synthesized using Xilinx ISE 14.7 synthesis technologies with an optimized goal of ‘Speed’. All simulation results are verified using high-level Maple software. The proposed ECP hardware in Jacobian coordinates is also synthesized using Synopsys Design Compiler with United Microelectronics (UMC) standard logic-cell library (65nm, 1.2V, 25°C) for normal case analysis. In our experiment, the Synopsys Design Compiler gives better results using our new proposed hardware. We have compared the overall performance of the ECSM with those in the available literature. The detailed results are given in Table 10.5.

Modular addition and subtraction are implemented very efficiently; both operations take only one clock cycle. The computation time of modular addition and subtraction is only 1.13 ns in Kintex-7 (XC7K325T-2FFG900) FPGA and 3.51 ns in Virtex-5 (XC5VLX330-2FF1760) FPGA for a prime field of either \mathbb{F}_{224} or \mathbb{F}_{256} . Both designs were also synthesized using 65nm CMOS technology with a target clock of 1 ns. Thus, the area of modular addition, subtraction, and combined addition and subtraction consumes only 7.96 Kgates (0.01655 mm²), 16.17 Kgates (0.0336 mm²), and 18.38 Kgates (0.038 mm²) respectively.

Table 10.1 presents modular multiplication and inversion results and performance over the prime field \mathbb{F}_p . A Kintex-7 FPGA has been used as the hardware platform for the ECP in affine coordinates because the Virtex-II FPGA is now obsolete. A new modular multiplication algorithm and architecture has been designed, and implemented in both Kintex-7 and Virtex-5 FPGA, and the results are shown in Table 10.2. We have implemented this new architecture on a Virtex-5 FPGA also, and its performance is slightly worse than with a Kintex-7 FPGA. As can be seen from Table 10.2, this new design on a Kintex-7 FPGA shows better performance than our Montgomery modular multiplication method. Using this new modular multiplication, we have designed a high-performance ECP in Jacobian coordinates. A modular multiplication over the prime field for an ECSM is implemented by Ghosh et al. [75]. They have also used the interleaved modular multiplication method. The hardware resources of their design are 4657 and 5379 slices, and the computation times are $6.00 \mu s$ and $7.30 \mu s$ respectively to achieve 224-bit and 256-bit modular multiplication. Their design takes k clock cycles for k -bit modular multiplication, which is almost the same as our design, but we have achieved a more area-efficient design. A 256-bit modular multiplication using the Montgomery method on a Virtex-II FPGA over the field \mathbb{F}_{256} is presented in [172] and [80], and their implemented modular multiplications require 18.28 and $4.06 \mu s$, respectively. However, our designed modular multiplier using the Montgomery method over \mathbb{F}_{256} on a Xilinx Kintex-7 FPGA takes only 605 slices and $1.68 \mu s$. The throughput rate of our design is also higher than the other available designs.

We have also designed a modular inversion for ECP in both affine and Jacobian coordinates. In [75], the authors propose a modular inversion for ECC over the NIST prime fields \mathbb{F}_{192} , \mathbb{F}_{224} , and \mathbb{F}_{256} . Their design takes $2m$ clock cycles for an m -bit modular inversion. A 256-bit modular inversion on a Virtex-II FPGA is proposed in [79], [76], and [80], and their implemented modular inversion requires $15.22 \mu s$, $1160 \mu s$, and 6.4

Table 10.1: *Performance analysis of FFMA for ECP in affine coordinates over \mathbb{F}_p on FPGA.*

FFMA Operation	Platform (Technology)	Field, Bit Length	Area (Slices)	Frequency (MHz)	Time (μ s)	AT	Throughput Rate (Mbps)
Modular Multiplication	Kintex-7	\mathbb{F}_p 224	506	163.76	1.37	0.68	163.50
		\mathbb{F}_p 256	605	152.71	1.68	1 ^a	152.38
Modular Inversion	Kintex-7	\mathbb{F}_p 224	1263	156.27	1.90	0.70	117.90
		\mathbb{F}_p 256	1480	146.38	2.33	1 ^b	109.87

^aThe normalization factor of $A \times T$ for modular multiplication is $1/0.001018 = 982.3183$, where A is area (slices) and T is time (s)

^bThe normalization factor of $A \times T$ for modular inversion is $1/0.00344692 = 290.114$, where A is area (slices) and T is time (s)

Table 10.2: *FPGA Implementation of modular multiplication for ECP in Jacobian coordinates.*

Circuit	Platform (Technology)	Field, Bit Length	Area (Slices)	Frequency (MHz)	Time (μ s)	AT ^a	Throughput Rate (Mbps)
Modular Multiplication	Kintex-7	\mathbb{F}_p 224	365	130.49	1.71	0.63	130.99
		\mathbb{F}_p 256	397	135.89	1.88	0.76	136.17
	Virtex-5	\mathbb{F}_p 224	362	82.39	2.72	1	82.39
		\mathbb{F}_p 256	420	78.22	3.27	1.40	78.29

^aThe normalization factor for $A \times T$ is $1/0.001018 = 982.3183$, where A is area (slices) and T is time (s)

μs respectively. In [76, 79, 80], their designs consume 14844, 2085, and 5477 slices respectively. Of them, the McIvor design consumes more area than all other available designs and Vliegen's design takes more time than all other designs. We have a trade-off between area and speed, and take only 1480 slices and $2.33 \mu\text{s}$ for a 256-bit modular inversion. Our design takes an average $1.33m$ CCs for an m -bit modular inversion. We have also achieved a higher clock frequency and less delay than all the other designs.

Most of the modular multiplication architecture was implemented on an FPGA. We have also synthesized our new modular multiplication (Algorithm 10.1, Fig. 10.2) using UMC 65nm CMOS technology; results are shown in Table 10.3. This indicates that the proposed design is very fast as well as area-efficient. It takes only 468 ns with an area of 0.0275 mm^2 (13.26 K gates) for 256-bit modular multiplication. The throughput rate of our design is 547 Mbps, which is very high for a prime field of either \mathbb{F}_{224} or \mathbb{F}_{256} .

Hardware implementation results for PDBL and PADD are presented in Table 10.4. PBDL over the prime fields \mathbb{F}_{224} and \mathbb{F}_{256} is performed in Xilinx Kintex-7 in $6.20 \mu\text{s}$ and $7.65 \mu\text{s}$ respectively. Similarly, PADD needs computation times of $6.09 \mu\text{s}$ and $7.57 \mu\text{s}$ with lower hardware resources. Based on our implementation results, we note that the computation times for both operations are much less due to the efficient implementation of FFMA. We have achieved a high throughput rate of almost 34 Mbps for both operations. In addition, a novel PDPA architecture has been designed (shown in Fig. 10.3c) with the latency of $6m + 14$, which is very small compared to other available designs in the literature.

Table 10.5 represents the ECSM results and performance comparisons with similar designs over the NIST prime field \mathbb{F}_p . Xilinx Kintex-7 and Virtex-5 FPGAs have been used for this work as the hardware implementation platforms because the Virtex-II FPGA is now obsolete. In affine coordinates, we achieve a scalar multiplication in 3.05 ms at the frequency of 140.7 MHz and 4.70 ms at the frequency of 119.2 MHz in a Xilinx

Table 10.3: *ASIC implementation of modular multiplication for ECP in Jacobian coordinates over \mathbb{F}_p .*

Circuit	Platform (Technology)	Field, Bit Length	Area	Cycles	Frequency (MHz)	Time (μ s)	Throughput Rate (Mbps)
Modular	65nm CMOS ^a	\mathbb{F}_p 224	0.0266mm ² /12.79K gates	225	549.45	0.409	547.00
Multiplication		\mathbb{F}_p 256	0.0275mm ² /13.26 K gates	257	549.45	0.468	547.00

^aUsed UMC standard logic cell library (65nm, 1.2V, 25°C) for normal case analysis

Table 10.4: *EC Group Operation (PDBL and PADD) Results in affine coordinates for $GF(p)$ on Kintex-7*

Group Opn	$ p , (bits)$	Area (Slices)	Latency (CCs)	Average Time (μs)@f (MHz)	Throughput Rate (Mbps)
PDBL	224	4130	$5m+12$	6.20@156.56	36.13
	256	4597	$5m+12$	7.65@146.40	33.46
PADD	224	3459	$5m+10$	6.09@156.56	36.78
	256	3861	$5m+10$	7.57@146.40	33.82

$m \simeq \lceil \log_2 p \rceil$ and CC=number of clock cycle

Table 10.5: Comparison between our ECC design and similar work over $GF(p)$

Circuit	Platform	Field / Bit Length	Reported Area (slices)	KCycles	Time (ms/ECSM) @f (MHz)	Area×Time (AT)	Relative AT	Throughput rate (kbps)
This work [a] (Jacobian)	65nm CMOS	\mathbb{F}_p 224	0.81mm ² / 393Kgates	304.6	0.56@546.5	-	-	400
		\mathbb{F}_p 256	0.93mm ² / 447Kgates	397.3	0.73@546.5	-	-	350
This work [b] (Jacobian)	Kintex-7	\mathbb{F}_p 224	9.7K	304.6	2.36@128.9	22.89	0.52	94.91
		\mathbb{F}_p 256	11.3K	397.3	3.27@121.5	36.95	0.84	78.28
This work [c] (Affine)	Kintex-7	\mathbb{F}_p 224	8.4K	429.7	3.05@140.7	25.62	0.59	73.44
		\mathbb{F}_p 256	9.3K	560.7	4.70@119.2	43.71	1	54.41
This work [d] (Jacobian)	Virtex-5	\mathbb{F}_p 224	10.7K	304.6	3.64@83.65	38.95	0.89	61.54
		\mathbb{F}_p 256	12.3K	397.3	5.26@75.43	64.70	1.48	48.67
Loi and Ko [175]	Virtex-4	\mathbb{F}_p 192		429.7	2.36@182.0	16.57	0.38	81.32
		\mathbb{F}_p 224	7.02K	666.7	3.66@182.0	25.70	0.59	61.2
		\mathbb{F}_p 256	+	993.2	5.46@182.0	38.30	0.88	46.91
		\mathbb{F}_p 384	8DSPs	2968.4	16.31@182.0	114.50	2.62	23.54
		\mathbb{F}_p 521		7048.9	38.73@182.0	271.88	6.22	13.45

The normalization factor for relative $A \times T$ is 43.71, where A is area (slices) and T is time (s).

Table 10.5: Comparison between our ECC design and similar work over $GF(p)$

Circuit	Platform	Field / Bit Length	Reported Area (slices)	KCycles	Time (ms/ECSM) @f (MHz)	AT ^a	Relative AT	TR ^b (kbps)
Lee et al. [44]	90nm CMOS	\mathbb{F}_p 224	1.12mm ² / 313K gates	127.2	0.59@217.0	-	-	379.66
Marzouqi et al. [176]	Virtex-5	\mathbb{F}_p 256	10.2K	165.1	0.76@217.0	-	-	336.84
Ghosh et al. [75]	Virtex-II Pro	\mathbb{F}_p 192	9.0K	442.2	6.63@66.7	67.63	1.55	38.61
		\mathbb{F}_p 224	10.4K	192.2	4.47@43.0	40.11	0.92	42.95
		\mathbb{F}_p 256	12.0K	260.0	6.50@40.0	67.51	1.54	34.46
Vliegen et al. [76]	Virtex-II	\mathbb{F}_p 192	2.1K+7DSPs	338.0	9.38@36.0	112.12	2.57	27.29
Ananyi et al. [177]	Virtex-II Pro	\mathbb{F}_p 224	20.8K	1074.83	15.76@68.2	32.86	0.75	16.24
		\mathbb{F}_p 256	+	288.0	4.80@60.0	99.84	2.28	40.00
		\mathbb{F}_p 384	32DSPs	348.0	5.80@60.0	120.64	2.76	38.62
		\mathbb{F}_p 521		414.0	6.90@60.0	143.52	3.28	37.10
				1194.0	19.90@60.0	413.92	9.47	19.30
				2736.0	45.60@60.0	948.48	21.70	11.43

^aAT = Area × Time and ^bTR = Throughput rate.

Table 10.5: Comparison between our ECC design and similar work over $GF(p)$

Circuit	Platform	Field / Bit Length	Reported Area (slices)	KCycles	Time (ms/ECSM) @f (MHz)	AT ^a	Relative AT	TR ^b (kpbs)
Ghosh et al. [90]	Virtex-4	\mathbb{F}_p 192	14.9K	227.9	4.30@53.0	63.89	1.46	44.65
		\mathbb{F}_p 224	17.3K	305.5	6.50@47.0	112.65	2.58	34.46
		\mathbb{F}_p 256	20.1K	395.6	9.20@43.0	185.13	4.24	27.83
Ghosh et al. [90]	130nm CMOS	\mathbb{F}_p 192	-/123.1KGates	187.68	1.36@138	-	-	141.17
		\mathbb{F}_p 224	-/143.9K Gates	253.5	1.95@130	-	-	114.87
		\mathbb{F}_p 256	-/167.5K Gates	331.1	3.01@110.0	-	-	85.05
Lai and Huang [169]	130nm CMOS	\mathbb{F}_p 160	-/169.4K Gates	74.02	0.36@208	-	-	444.44
		\mathbb{F}_p 256	-/197.0K Gates	252.1	1.21@208.0	-	-	211.57
Ahmadi et al. [178]	130nm CMOS	\mathbb{F}_p 192	-	9712.5	525@18.5	-	-	0.37
Fan et al. [179]	Virtex-II Pro	\mathbb{F}_p 192	3.2K+16DSPs	920.7	9.90@93.0	31.41	0.72	19.39
Mentens et al. [180]	Spartan-3	\mathbb{F}_p 256	4.8K+66DSPs	1768.8	26.80@66.0	129.33	2.96	9.55
McIvor et al. [79]	Virtex-II Pro	\mathbb{F}_p 256	15.8K+256DSPs	152.3	3.86@39.5	60.81	1.39	66.32
Sakiyama et al. [181]	Spartan-3	\mathbb{F}_p 256	27.6K	708.0	17.7@40.0	488.47	11.18	14.46

^a AT = Area \times Time, ^bTR = Throughput Rate.

Kintex-7 FPGA over the prime fields \mathbb{F}_{224} and \mathbb{F}_{256} , respectively. Our proposed ECSM provides around 20% better delay performance than previous designs, and is also area-efficient: it takes only 8.4K slices and 9.3K slices respectively without using any DSP slices. Most ECC architectures in the available literature have used separate PBDL and PADD modules. We have proposed a new ECSM hardware in Jacobian coordinates using PDPA (combined PDBL and PADD module). There are few hardware implementations on ASIC, most being implemented in FPGA. Our proposed ECP was also synthesized using UMC 65nm CMOS technology. To make a fair comparison with other ASIC implementations, we used post-synthesis results of our new design. Our designed ECP in Jacobian coordinates takes 0.56 ms and 0.73 ms over the prime fields \mathbb{F}_{224} and \mathbb{F}_{256} , respectively, with a clock frequency of 546.5 MHz. We have also achieved an area-efficient design which requires less than 1 mm² area on 65nm CMOS ASIC. The proposed new ECP in Jacobian coordinates takes less time than previous designs on FPGA. It takes 2.36 ms for the prime field \mathbb{F}_{224} and 3.27 ms for the prime field \mathbb{F}_{256} in a Kintex-7 FPGA. This new ECP architecture was also implemented in a Virtex-5 FPGA. All the available results for our new designs are presented in the first, second, and fourth rows of Table 10.5.

The ECP proposed by Kung [175] provides the fastest ECPs, highest frequency, and a better area and time (AT) product. Their design requires fewer slices than our design, however they need 8 DSP slices and more CCs for implementation, whereas we have no need for any DSP slices for our proposed design. A new architecture of ECSM using PDBL and PADD modules was implemented in 90nm CMOS technology by Lee [44]. Their proposed design requires more than 1 mm² of area, whereas our proposed design requires less than 1 mm² of area. Besides, our design is faster and has a higher throughput rate than their proposed ECSM. A 256-bit ECP presented in [176] takes a similar number of CCs but needs more area and time to compute a scalar multiplication than our design, and the AT value of their design is almost twice our AT. An ECSM is presented by Ghosh [75],

and their ECP requires 6.50 ms and 9.38 ms over the fields \mathbb{F}_{224} and \mathbb{F}_{256} respectively to achieve a scalar multiplication. However, our ECP requires only one-third the computation time of their design. Our two implementations show a similar performance to their design in terms of area and CCs. ECPs over the field \mathbb{F}_{256} are presented by Vliegen [76], Mentens [180], and Sakiyama [181], and their proposed crypto-processors require 15.76 ms, 26.80 ms, 3.86 ms, and 17.70 ms, respectively, for a typical scalar/point multiplication. We can see in Table 10.5 that their design has more delay than our proposed design. Besides, the throughput rate of our design is better than the others. The ECP proposed by Ananyi [177] provides results for all five NIST-recommended prime curves, and their design takes between 4.80 ms (192-bit ECC) and 45.60 ms (521-bit ECC) to compute a scalar multiplication. Their proposed ECP requires 20.8K slices and 32 DSP slices on a Virtex-II Pro FPGA, which is more than our designs. A parallel ECP for \mathbb{F}_p is presented by Ghosh [90]; their design takes almost double the area and time of our design. In [90] and [169], they implemented their design on TSMC 130nm CMOS technology using separate PDBL and PADD modules, however we have synthesized our new ECP using a PDPA module on 65nm CMOS. Their design takes more computation time than ours. Ahmadi [178] and Fan [179] also implemented ECPs over the prime field \mathbb{F}_{192} on different platforms, and their cryptographic processors are slower than our processor. Although the McIvor [79] design provides a better result in terms of time, area-time (AT) product, and throughput rate, it requires an additional 256 DSP slices for hardware implementation. On the other hand, we have implemented the ECP with almost the same AT product without any DSP slices on the FPGA.

It can be noted that the ECPs are implemented on different platforms and employ different hardware resources, so it is difficult to state which design is the best. However, the best indicator for efficient design is the product of area and time (AT), and throughput rate. For this reason, we have calculated the AT and throughput rate for all the available

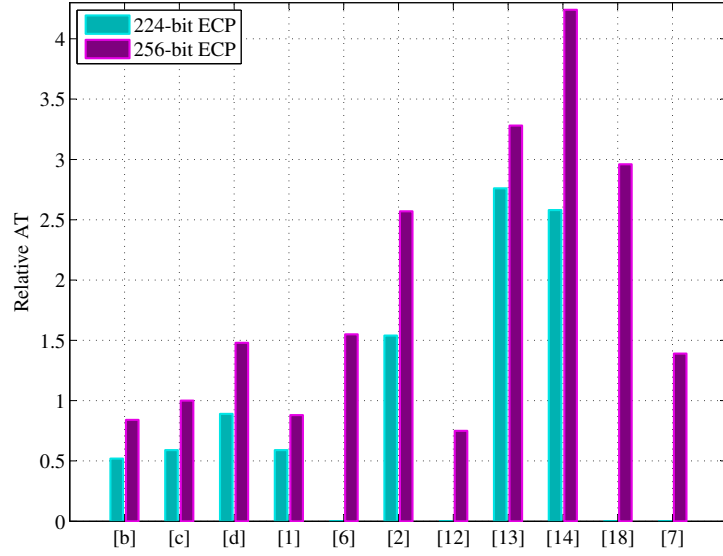


Figure 10.7: Comparison of relative AT values between our ECC design ([b], [c], and [d]) and similar work.

designs, and make a comparison of these AT and throughput rates with our design. All these calculated AT values, relative AT values, and throughput rates have been presented in columns seven to nine of Table 10.5. The relative AT product comparison of similar work is shown in Fig. A.1. In Fig. A.1, [b], [c], and [d] represent our implementation results (both 224-bit and 256-bit ECC) and [76, 176, 180], and [79] illustrate reference implementations for 256-bit ECC only. This figure also demonstrates that our proposed designs require lower AT than most of the similar designs in the available literature. Note that, of all the available designs, in terms of AT value the Kung [175] and Vliegen [76] designs perform the best. However we have achieved a high throughput rate compared to the designs in [175] and [76]. Besides, their designs require additional DSP slices for hardware implementation. The ECP designs in [75, 79, 90, 177] need fewer CCs for a scalar multiplication, but would require more slices than our design. We require less hardware resources for our design compared to other previous designs. Beside, our new ECP in Jacobian coordinates is not only area-efficient but also faster than all other designs, even

better than our previous design. From the performance analysis and comparison of different ECPs over the prime field in Table 10.5, some are only area-efficient or some are better in terms of only speed; it can be concluded that our ECP performs better than other comparable designs.

10.6 Conclusion

A fast, high-performance ECP over prime field $\text{GF}(p)$ is developed using efficient FFMA, EC group operations, and ECSM. Our design supports two NIST prime fields of the five NIST-recommended primes p , with sizes 224 and 256 bits. A novel PDPA technique is proposed to perform EC group operations in parallel, aimed at reducing the number of steps in the PDBL and PADD operations and decreasing the overall latency of the ECP. Thus, we have designed a faster ECP in Jacobian coordinates which takes 0.56 ms for \mathbb{F}_{224} and 0.73 ms for \mathbb{F}_{256} in ASIC 65nm CMOS. We also present the ECSM results of our new proposed design in Xilinx Kintex-7 and Virtex-5 FPGAs. The proposed ECP in Jacobian coordinates takes between 2.36 ms and 3.27 ms on a Xilinx Kintex-7 FPGA and between 3.64 ms and 5.26 ms on a Xilinx Virtex-5 FPGA. In addition, our proposed ECP in affine coordinates on a Xilinx Kintex-7 FPGA takes between 3.05 and 4.70 ms to execute a typical scalar/point multiplication, which represents the fastest hardware implementation result in an affine coordinate system. The hardware architecture delivers a high-performance operation with fewer hardware resources. The implemented design is optimized by using different techniques such as balancing the PDBL and PADD architecture, parallelization in operations, and pre-computations, for obtaining higher performance. Based on the overall performance analysis and comparisons of different ECPs over the prime field \mathbb{F}_p , it can be concluded that this design provides better performance than others in terms of the area, delay, AT, and throughput rate.

Chapter 11

Energy-Efficient, High-Speed, ASIC-Based Elliptic Curve Cryptography Processor¹

11.1 Abstract

In this paper, we propose a high-performance ASIC-based implementation of an elliptic curve cryptography processor (ECP) over NIST prime fields \mathbb{F}_p . Our design supports three prime fields of the five NIST-recommended primes p , with sizes 192, 224, and 256 bits. An energy-efficient elliptic curve point multiplication (ECPM), which is the core operation of an ECP, is developed in Jacobian coordinates, then converted to affine coordinates for the practical realization of the cryptosystem. The ECPM is synthesized in ASIC 65-nm technology by designing a novel combined point doubling and point addition

¹Submitted as: Md Selim Hossain, Shahzad Asif, Oskar Andersson, Joachim Neves Rodrigues and Yinan Kong, “Energy-Efficient, High-Speed, ASIC-Based Elliptic Curve Cryptography Processor,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, major revision submitted.

(PDPA) architecture. This PDPA is designed using efficient modular arithmetic to achieve high speed and low hardware utilization for ECP. The proposed ECPM has been implemented by developing radix-4 modular multiplication, which saves 50% in clock cycles. The delay per ECPM is between 0.21 and 0.37 ms, which is the fastest hardware implementation result reported in the literature to date. Implementation results show that the proposed design is energy-efficient, and the energy dissipation is only 0.3% of that of other similar designs. Based on the overall performance, the proposed ECP over \mathbb{F}_p provides better performance which can be used for modern IoT security applications.

11.2 Introduction

With the fast growth of secure transactions over the network and associated appliances, the demand for data security has increased rapidly in recent days. For these applications, public-key cryptography (PKC) such as elliptic curve cryptography (ECC) [10, 11] and Rivest-Shamir-Adleman (RSA) [8] cryptography play a vital role to transmit the secured information among different wireless devices. PKC is widely used for key-agreement protocols, encryption/decryption, and digital signatures. The Internet-of-Things (IoT) is an interconnected system over a network in which different objects are connected with unique identifiers and the ability to transfer data without requiring human-to-human or human-to-computer interaction. The crypto-algorithms (e.g. ECC) designs should consume minimum hardware resources as well as less energy with high throughput to provide efficient security in IoT applications [183]. Therefore a high-performance hardware implementation is mandatory for ECC, especially to speed up the calculations in an ECC processor (ECP).

The RSA and ECC cryptosystems are the most popular, powerful, and widely used PKC for cryptographic applications. ECC was first proposed by N. Koblitz and V. Miller in the mid-80s. It is gradually becoming a more attractive alternative in the past few years to RSA cryptosystems because ECC can provide the same level of security as the conventional RSA cryptosystem with a significantly shorter key. Besides, less memory and hardware resources are required to implement ECC [5, 22, 54]. This attractive feature makes ECC very popular for IoT hardware security. The National Institute of Standards and Technology (NIST) recommends elliptic curves (ECs) over prime fields for the digital signature standard (DSS) [22]. IEEE P1363-2000 [23] also has standardized public-key cryptographic techniques, including cryptographic schemes, use of ECC-based key agreement and digital signature algorithm (DSA).

Over the decades, numerous implementations of ECPs over a prime field \mathbb{F}_p have been proposed in the literature. We find that most of the implementations are targeted to FPGA [79, 90, 175–177], whereas only a few hardware implementations target an ASIC [19, 21, 44, 90, 168, 169, 178, 184–186]. A flexible dual-field ECP using the hardware-software approach was proposed in [21], but needs more computation time to implement on 55-nm CMOS technology. A heterogeneous dual-processing element architecture for a dual-field ECP was proposed and implemented using ASIC 90-nm CMOS technology [44]. In [168], an energy-efficient 160-bit ECP over a dual field was introduced, capable of parallel and serial operation modes of the cryptosystem. In [90], ECP over $\text{GF}(p)$ was proposed and synthesized in ASIC 0.13- μm technology, and a parallel architecture unit is used for their design. Two efficient ASIC-based high-throughput dual-field ECPs were proposed in [169] and [185]. Efficient hardware implementations of an elliptic curve cryptosystem over the prime field have been proposed in [19], and synthesized using ASIC 0.18- μm CMOS technology. In [178, 184–186], ECPs over the prime field \mathbb{F}_p have been implemented using ASIC 0.13- μm CMOS technology. Only a few high-speed and low-power ECPs have

been reported over \mathbb{F}_p in the literature. Most are only area-efficient or energy-efficient or are superior only regarding speed. To have a trade-off between power, speed, and area complexities, a well-designed and efficient ECP is necessary for the security requirements of modern IoT applications.

Contribution: In this paper, a high-performance ASIC-based ECP is proposed and implemented in which ECPM operations are performed with a very low area and latency. We propose a novel combined algorithm to compute point doubling (PD) and point addition (PA) (EC group operations) together with low latency ($3m + 14$ clock cycles only). Architectures for Jacobian-to-affine coordinate conversion, serial-in parallel-out (SIPO), and parallel-in serial-out (PISO) are developed at the top level to interface the I/O ports of the ECP. A radix-4 modular multiplication is proposed which provides a low latency (saves 50% of clock cycles) and low area complexity. A high-speed, low-power finite-field modular arithmetic (FFMA) unit is implemented that performs modular multiplication, inversion, addition, and subtraction. This FFMA unit is used in the construction of an effective ECP. Thus, our proposed ASIC-based ECP requires less time and energy than all comparable work in the literature, which is appropriate for high-throughput and resource-constrained applications.

The rest of the paper is organized as follows. Section 11.3 gives a brief introduction of the mathematical background of ECC over the prime field \mathbb{F}_p . All finite-field modular arithmetic units over \mathbb{F}_p are described in Section 11.4. Section 11.5 describes a combined EC group operation, namely PDPA. The proposed ECPM, ECP, and their hardware architectures are given in Section 11.6. ASIC implementation results, and comparisons with related designs, are discussed in Section 11.7. Finally, Section 11.8 summarizes our work.

11.3 Preliminaries

11.3.1 ECC

ECC is the most popular and influential public-key encryption technique nowadays, due to the smaller field size. The hardware implementations of ECC can be employed in either the NIST prime fields denoted by $\text{GF}(p)$ (where p is a ‘large’ prime), or in a binary field denoted by $\text{GF}(2^m)$ (where m is a positive integer). But FFMA over $\text{GF}(p)$ or \mathbb{F}_p will be the emphasis of this work. Besides, the most widely used PKC is RSA, which uses modular arithmetic over a prime field \mathbb{F}_p . Furthermore, ECC over the prime field can be utilized for an elliptic curve digital signature algorithm. According to IEEE P1363 [23], an elliptic curve E over $\text{GF}(p)$ is the set of solutions for an equation such as

$$y^2 = x^3 + ax + b \pmod{p} \quad (11.1)$$

where $x, y, a, b \in \text{GF}(p)$ with

$$4a^3 + 27b^2 \not\equiv 0 \pmod{p},$$

together with a special point called the point at infinity. The coefficients $a, b \in \mathbb{F}_p$ specifying an elliptic curve $E(\mathbb{F}_p)$ are defined by (11.1). The number of points on elliptic curve E is represented by $\#E(\mathbb{F}_p)$. It is defined over \mathbb{F}_p as nh , where n is the prime order of the curve and the integer h is a co-factor such as $h = \#E(\mathbb{F}_p)/n$ [2, 10, 11].

Let $P = (x, y)$ be a point in an affine coordinate systems; the projective coordinate systems $P = (X, Y, Z)$ are given by the following equation [88]:

$$X = x; \ Y = y; \ Z = 1. \quad (11.2)$$

The projective point $P = (X, Y, Z)$, $Z \neq 0$ corresponding to the affine point $P = P(x, y)$ is given by

$$x = X/Z^2; \ y = Y/Z^3. \quad (11.3)$$

Using (11.1), (11.2), and (11.3), the projective form of the Weierstrass equation of the elliptic curve becomes

$$Y^2 = X^3 + aXZ^4 + bZ^6. \quad (11.4)$$

Let $P = (X_1, Y_1, Z_1)$ and $Q = (X_2, Y_2, Z_2)$ be two points on the elliptic curve, then the point doubling (PD) and point addition (PA) formulae in Jacobian coordinates are given below.

$$\begin{aligned} R(X_3, Y_3, Z_3) &= 2P(X_1, Y_1, Z_1) \in E(\mathbb{F}_p), \\ X_3 &= (3X_1^2 + aZ_1^4)^2 - 8X_1Y_1^2, \\ \text{or } X_3 &= (3X_1^2)^2 - 8X_1Y_1^2, \\ Y_3 &= (3X_1^2 + aZ_1^4)(4X_1Y_1^2 - X_3) - 8Y_1^4, \\ \text{or } Y_3 &= 3X_1^2(4X_1Y_1^2 - X_3) - 8Y_1^4, \\ Z_3 &= 2Y_1Z_1; (a = 0 \text{ for Koblitz Curve}) \end{aligned} \quad (11.5)$$

$$\begin{aligned} R(X_3, Y_3, Z_3) &= P(X_1, Y_1, Z_1) + Q(X_2, Y_2, Z_2) \in E(\mathbb{F}_p), \\ X_3 &= A^2 - B^3 - 2X_1Z_2^2B^2, \\ Y_3 &= A(X_1Z_2^2B^2 - X_3) - Y_1Z_2^3B^3, \\ Z_3 &= Z_1Z_2B^0, \end{aligned} \quad (11.6)$$

$$\text{where } A = Y_2Z_1^3 - Y_1Z_2^3 \text{ and } B = X_2Z_1^2 - X_1Z_2^2.$$

Hence when $P = Q$ we have the PD operation in (11.5) and when $P \neq Q$ we have the PA operation in (11.6) [89]. We have combined these two EC group equations for compact hardware implementation and called it combined point doubling and point addition (PDPA). Besides, we have used the Koblitz curve where $a = 0$, then we can reduce the area and latency of the data path. Using these group operations, the ECPM $R = kP$, which is the most important operation in ECC, is implemented in Jacobian coordinates.

In 2000, FIPS-2 was recommended with 10 finite fields: 5 prime fields and 5 binary fields. The prime fields are \mathbb{F}_{192} , \mathbb{F}_{224} , \mathbb{F}_{256} , \mathbb{F}_{384} and \mathbb{F}_{521} . Our design supports three of the five NIST-recommended primes p , with sizes 192, 224, and 256 bits. All the parameters for the NIST elliptic curve over the prime fields \mathbb{F}_{192} , \mathbb{F}_{224} , and \mathbb{F}_{256} are listed in [54].

11.3.2 Coordinate Systems for EC Point Representation

An EC defined over a Galois field provides a group structure that is used to implement cryptographic systems. The group operations are PD and PA. We propose PDPA for group operations in Jacobian coordinates. There are various coordinate systems to represent elliptic curve points, but two well-known coordinate systems are often used for ECC: affine coordinate systems and projective coordinate systems. A point on the EC $E(\mathbb{F}_p)$ for affine coordinates can be represented by using two elements $x, y \in \mathbb{F}_p$, i.e. $P(x, y)$. In this coordinate system, the elliptic curve group operations require a modular inversion, the most expensive operation. In projective coordinates, a point P on the EC needs three elements $X, Y, Z \in \mathbb{F}_p$, i.e. $P(X, Y, Z)$. In this paper, we have used Jacobian projective coordinate systems for the EC points; thereby avoiding modular inversion. However, EC group operations need more modular multiplication in projective coordinates, which is also a very costly operation for ECC. To get a high-performance ECP, Radix-4 modular multiplication is proposed that is well suited for faster ECC operation. In practice, to convert projective to affine coordinates, one modular inversion is still needed for an ECPM. There are plenty of projective coordinates in the available literature such as Jacobian, Lopez-Dahab, and Chudnovsky coordinates; a detailed coordinate system is discussed in [2].

11.4 Hardware for FFMA over \mathbb{F}_p

This section presents all algorithms and hardware architectures related to FFMA, which is necessary for the PDPA and ECPM. FFMA and EC group operations are the building blocks of ECPM and ECC protocols. The FFMA units are the bottom level in the hierarchy, and these are the most crucial for the overall performance of the ECP.

11.4.1 Modular Adder and Subtractor

Fig. 11.1 depicts the hardware architecture of a modular adder and subtractor to perform the basic and simplest operations of FFMA over a prime field. The modular adder and subtractor are improved versions of [80, 182]. From Fig. 11.1(a), modular addition adds two inputs, x , y , and subtracts the modulus p from the sum until the sum is less than the modulus p . The intermediate result $(x + y)$ of modular addition must be at least $(m + 1)$ bits long and could be greater than the modulus. To subtract the modulus p from the intermediate result, an adder is used that adds the intermediate result to the bit-wise inverted modulus p with the carry-in set to 1, thus performing a two's-complement subtraction. The carry-out of the second adder checks whether the intermediate result is in the proper range or not. If the sum $(x + y)$ is in the proper range, the result of the first adder is correct; otherwise, the second adder is right. The condition is checked by using a multiplexer to select whether $(x + y)$ is greater than or equal to the modulus p . Modular subtraction over the prime field is performed similarly to modular addition. From Fig. 11.1(b), for modular subtraction, y is bitwise inverted and added to x with a carry-in set to 1. If the result is negative or the carry-out of this adder is low, then the modulus p must be added to produce an output in the correct range between 0 and $p - 1$ inclusive. A modular addition or subtraction operation takes only one clock cycle, and can speed up the overall calculation of PDPA and hence the ECP. A hardware architecture is proposed and

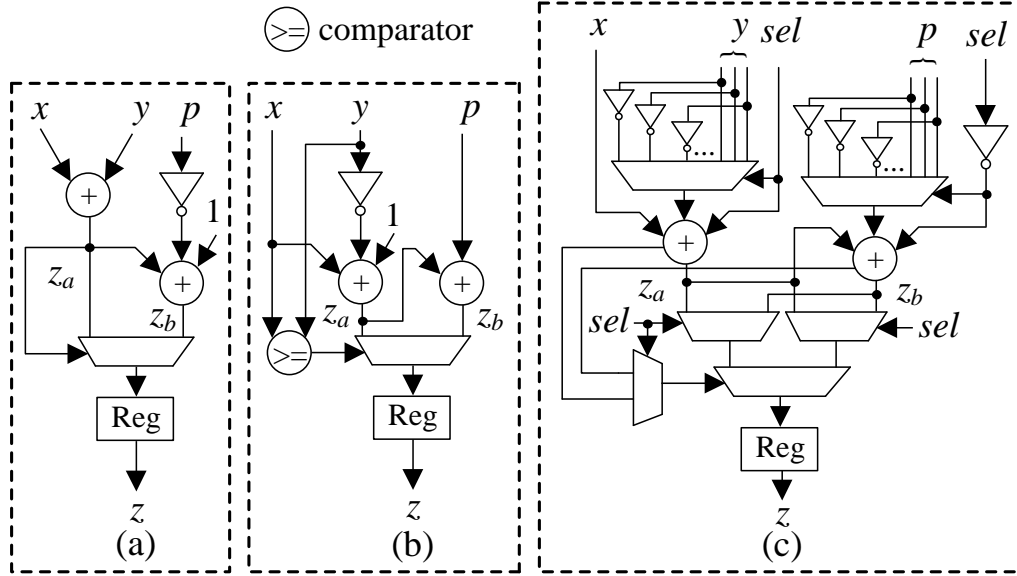


Figure 11.1: Hardware architecture of modular (a) adder (b) subtractor, and (c) combined modular adder and subtractor.

implemented for combined modular addition and subtraction, as shown in Fig. 11.1(c). A $(m + 1)$ -bit adder/subtractor has been used in this architecture. Modular addition or subtraction is performed depending on the ‘sel’ bit using this combined method. When the ‘sel’ bit is one, then modular subtraction is performed, otherwise the circuit performs modular addition. Parallelization in operations is used to increase the throughput rate of EC group operations. However, the combined module takes two clock cycles to get the final output, and requires more area than separate modular addition and subtraction modules.

11.4.2 Modular Multiplier

In order to implement point multiplication, efficient modular multiplication is mandatory because this is the most crucial operation for the ECP over the prime field, presented as

$$Z = (x \times y) \pmod{p}. \quad (11.7)$$

Algorithm 11.1: Standard Interleaved modular multiplication

Input: Prime p and $A, B \in [1, p - 1]$ **Output:** $C = (A * B) \bmod p$

1. $C = 0$;
 2. **for** $i = m - 1$ downto 0 **do**
 - 2.1 $c_1 = C$; $c_2 = A$; $c_3 = B$;
 - 2.2 $c_4 = 2 * c_1$ (Left-shift operation);
 - 2.3 $i_1 = A[i] * c_3$ or $i_1 = A[i] * B$ (and-gate operation);
 - 2.4 $c_5 = c_4 + i_1$; $c_6 = c_5$;
 - 2.5 **if** $c_6 \geq p$ **then** $c_7 = c_6 - p$; **end if** $c_8 = c_7$;
 - 2.6 **if** $c_8 \geq p$ **then** $c_9 = c_8 - p$; **end if** $C = c_9$;
 - 2.7 **end for**
 3. Return C
-

Many methods for performing modular multiplication are available, but most of these algorithms follow the basic concept of the Montgomery method which needs two Montgomery multiplications for one complete modular multiplication [69]. We propose a new modular multiplication based on an interleaved method. The benefit of the interleaved method is that the intermediate result is only one or two bits larger than the operands because the intermediate result is always reduced by taking the modulus. Algorithm 11.1 presents the standard interleaved modular multiplication method, and Fig. 11.2(a) illustrates the architecture of Algorithm 11.1 [60]. The latency/clock period of this approach is very low because the critical path mostly relies on an adder and a subtractor. However, this method needs $3m + 1$ clock cycles due to the three-stage register for the data path. The modified architecture of interleaved modular multiplication is shown in Fig. 11.2(b). As we can see from this hardware, this method needs only a single-stage register; we need

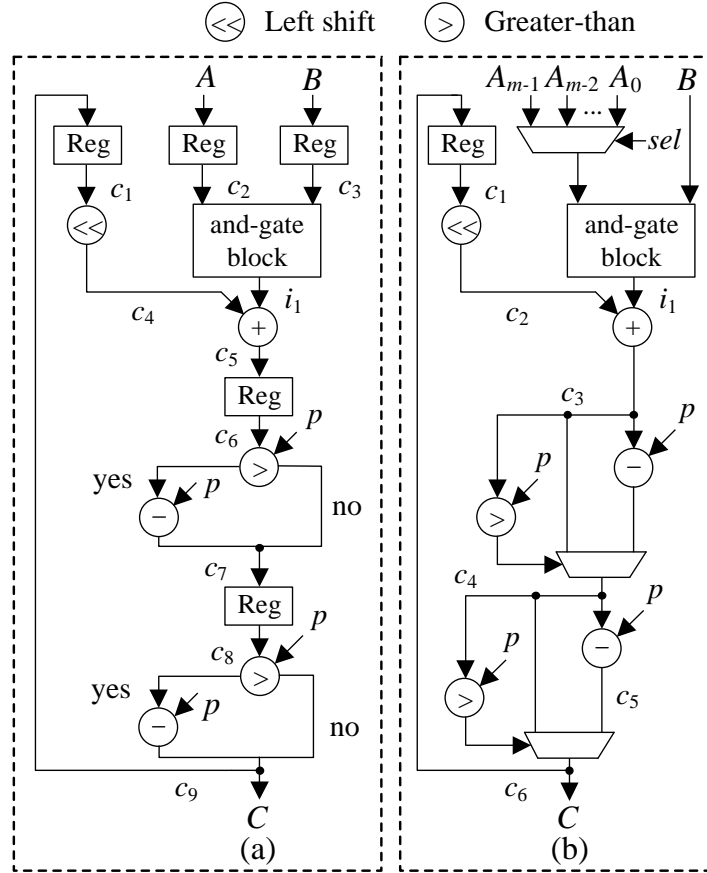


Figure 11.2: Hardware architecture for the interleaved modular multiplier.

only one third the number of clock cycles. Both architectures require two subtractions and two comparisons per iteration. These operations are run sequentially. The size of the adders used for this must be equal to $(m + 2)$ bits to handle the intermediate result at each iteration; $(m + 1)$ iterations are required for the modified method as depicted in Fig. 11.2(b).

An efficient algorithm is also proposed for modular multiplication/squaring, as shown in Algorithm 11.2, based on interleaved multiplication. Fig. 11.3(a) depicts the proposed architecture based on Algorithm 11.2 for modular multiplication over prime field \mathbb{F}_p . The proposed modular multiplication algorithm is very efficient because the critical path consists of only and gates, adder, and subtractors. However, this approach requires $m + 1$

Algorithm 11.2: Proposed algorithm for modular multiplication in $\text{GF}(p)$

Input: Prime p and $A, B \in [1, p - 1]$ **Output:** $C = (A * B) \bmod p$

1. $C = 0$; $p_2 = 2 * p$ (pre-computed) ;
 2. **for** $i = m - 1$ downto 0 **do**
 - 2.1 $c_1 = C$; $c_2 = 2 * c_1$ (left-shift operation);
 - 2.2 $i_1 = A[i] * B$ (and-gate operation);
 - 2.3 $c_3 = c_2 + i_{1s}$; $c_4 = c_3 - p$; $c_5 = c_3 - p_2$ ($p_2 = 2p$);
 - 2.4 **if** $c_3 \geq p$ **then** $c_6 = c_4$;
 - 2.5 **elseif** $c_3 \geq p_2$ **then** $c_6 = c_5$; **else** $c_6 = c_3$; **end if** $C = c_6$;
 - 2.6 **end for**
 3. Return C
-

cycles to compute the final result of modular multiplication where m is the bit length of operands A, B or p . Therefore, a higher-radix modular multiplication is required to reduce the cycle counts.

A radix-4 modular multiplication method, shown in Algorithm 11.3, is proposed to improve the overall performance by halving the number of clock cycles. Fig. 11.3(b) depicts the proposed hardware architecture of radix-4 modular multiplication. This method relies on iterative addition and reduction of partial products and performs reduction simultaneously [55–57]. As can be seen from Algorithm 11.3 or Fig. 11.3(b), this approach needs two left-shift operations, two simple additions, one comparator, six subtractions, and three multiplexer operations. In this method, multiplication by two or four is performed by a simple left-shift operation. The first multiplexer is used to select two bits of the first operand A such as $As = A_{[2i+1]}A_{[2i]}$ which are used as a select signal for the second multiplexer. In the second multiplexer of Fig. 11.3(b), when ‘ $As = 01$ ’ then the

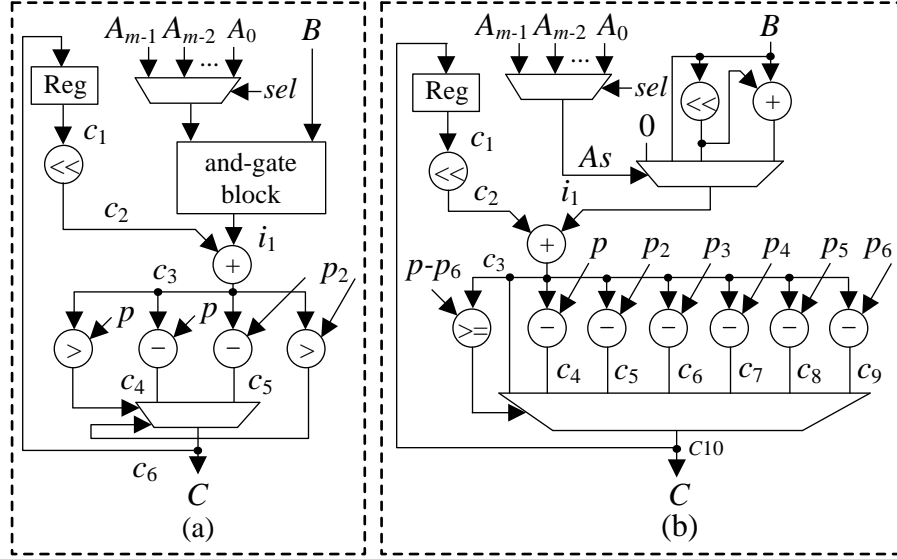


Figure 11.3: Proposed hardware architecture for (a) modular multiplier and (b) Radix-4 modular multiplier.

input value of B goes to the output of second multiplexer, which is the same as the input of the adder module. Similarly, when ' $As = 10$ ' then $2B$ results, and when ' $As = 11$ ' then $3B$ results, otherwise 0 results go to the input of the adder module. The second input c_2 of the adder comes from the left-shift module which is a left-shift by 2- operation. The intermediate result c_3 , which is the output of the adder, is then reduced with the modulus p by subtracting until the values are smaller than the modulus. For doing this, six subtractions and one three-bit comparator are required per iteration. These operations run in parallel, where the p_2, p_3, p_4, p_5 and p_6 values are pre-computed, which reduces the processing time. Then a multiplexer is needed to select which result is correct. In this architecture, parallel operations and pre-computations are used to speed up the computation. This method requires $m/2 + 1$ cycles to compute the final result of modular multiplication, where m is the bit length of operands A, B or p . Therefore, we have designed a high-performance modular multiplier which is essential for ECP in Jacobian coordinates.

Algorithm 11.3: Proposed algorithm for radix-4 modular multiplication

Input: Prime p and $A, B \in [1, p - 1]$ **Output:** $C = (A * B) \bmod p$ 1. $C = 0$; $p_2 = 2 * p$; $p_3 = 3 * p$; $p_4 = 4 * p$; (pre-computed)1.1 $p_5 = 5 * p$; $p_6 = 6 * p$ (pre-computed) ;2. **for** $i = m/2 - 1$ **downto** 0 **do**2.1 $c_1 = C$; $As = A_{2i+1}.A_{2i}$;2.2 $c_2 = 4 * c_1$ (left-shift operation); $i_1 = As * B$;2.3 $c_3 = c_2 + i_1$;2.4 $c_4 = c_3 - p$; $c_5 = c_3 - p_2$; $c_6 = c_3 - p_3$;2.5 $c_7 = c_3 - p_4$; $c_8 = c_3 - p_5$; $c_9 = c_3 - p_6$;2.6 **if** $c_3 \geq p_6$ **then** $c_{10} = c_9$; **elseif** $c_3 \geq p_5$ **then**2.7 $c_{10} = c_8$; **elseif** $c_3 \geq p_4$ **then** $c_{10} = c_7$;2.8 **elseif** $c_3 \geq p_3$ **then** $c_{10} = c_6$; **elseif** $c_3 \geq p_2$ **then**2.9 $c_{10} = c_5$; **elseif** $c_3 \geq p$ **then** $c_{10} = c_4$;2.10 **else** $c_{10} = c_3$; **end if** $c = c_{10}$;2.11 **end for**3. Return C

11.4.3 Modular Inversion

Modular inversion over the prime field is the most expensive operation in ECP, and it is mandatory for converting Jacobian to affine coordinates. There are different methods for performing modular inversion, but two well-known methods are regularly employed. The first is Fermat's Little Theorem (FLT); the multiplicative inverse of this method is obtained by modular exponentiation. However, this modular exponentiation is a very expensive operation for finding the inverse. The second method is based on the Extended

Euclidean GCD Algorithm (EEA). There are many variants of EEA reported in the available literature [2]. An efficient algorithm has been used for inversion which is well-known as the Binary Inversion Algorithm. The detailed steps of this algorithm, named Algorithm 2.22, is given in [2]. The hardware architecture based on this algorithm is explained in [84]. The modular inversion over the prime field is accomplished using a series of additions, subtractions, and shift operations. This algorithm works iteratively, and at every step either u or v decreases by at least one in bit length. The inverse of an integer a modulo p is defined as an integer R such that $a.R \equiv 1 \pmod{p}$. This classical definition of the modular inversion operation can be presented as

$$R = a^{-1} \pmod{p} \quad (11.8)$$

where a is an integer. From (11.8), the inverse of a exists if and only if a is relatively prime with p . Therefore, the GCD of a and p must be 1 or $\gcd(a, p) = 1$. From that Algorithm 2.22 [2], four registers - u, v, x , and y - are essential to implement a hardware architecture for inversion over a prime field. The calculation of divisions such as $u/2$, $v/2$, $x/2$, and $y/2$ depends upon parity and magnitude comparisons of the m -bit registers named u, v, x , and y . Two multiplexers are used to select u or v , and several multiplexers are used to select x or y as appropriate. The LSB determines (1 indicates odd, 0 indicates even) the parity of any number. But exact comparisons can be attained only through full m -bit subtraction, and this contributes a major delay before decisions regarding the next calculation can be made. We have used m -bit carry-propagation adders to execute the additions or subtractions. The implemented designs compute all possible values like $x, x/2, (x + p)/2, (x - y)/2, (x + p - y)/2$ or $y, y/2, (y + p)/2, (y - x)/2, (y + p - x)/2$ simultaneously to save time, and multiplexers are used for selecting the new values of x and y . This method takes $2m$ cycles for an m -bit modular inversion.

11.5 Proposed PDPA

The EC group operations in either a prime field or a binary field are point doubling (PD) and point addition (PA). We have designed a novel combined group operation for an ECP in Jacobian coordinates named PDPA, which performs the two group operations together. The building blocks of PDPA are modular addition, subtraction, multiplication, squaring, and inversion. Therefore, this module should be developed so that it requires minimal resources with the assurance of high throughput.

Fig. 11.4 depicts the proposed architecture of the PDPA operation in Jacobian coordinates, corresponding to (5) and (6) respectively. The PDPA operation requires ten additions, nine subtractions, eleven multiplications, and seven squarings. Note that modular squaring is the modular multiplication of two identical inputs, hence the same modular multiplication algorithm is used for squaring. For implementing PDPA, separate modular addition and subtraction modules are used because the combined module takes more area than a separate module. The proposed radix-4 modular multiplication is therefore utilized for the PDPA; we need almost half the number of clock cycles. In PDPA, parallel operations and pre-computations are used to increase the throughput rate of EC group operations. As can be seen from level 1 in Fig. 11.4, two squaring and one multiplication modules are operating in parallel. Similarly in level 6, a maximum of four modules such as three multiplications and one squaring module are running in parallel. Normally, 11 levels are required to perform the point doubling operation, and nine levels are needed to perform the point addition operation, whereas our proposed combined PDPA module requires only 12 levels. Using this parallel process and combined module, the data path is reduced by reducing the number of levels, and decreasing the latency and throughput of the EC group operations. The overall latency of the proposed PDPA architecture is only $3m + 14$, where m is the latency of modular multiplication in Jacobian coordinates. X_{3PD} , Y_{3PD} , and Z_{3PD} denote the output of PD whereas

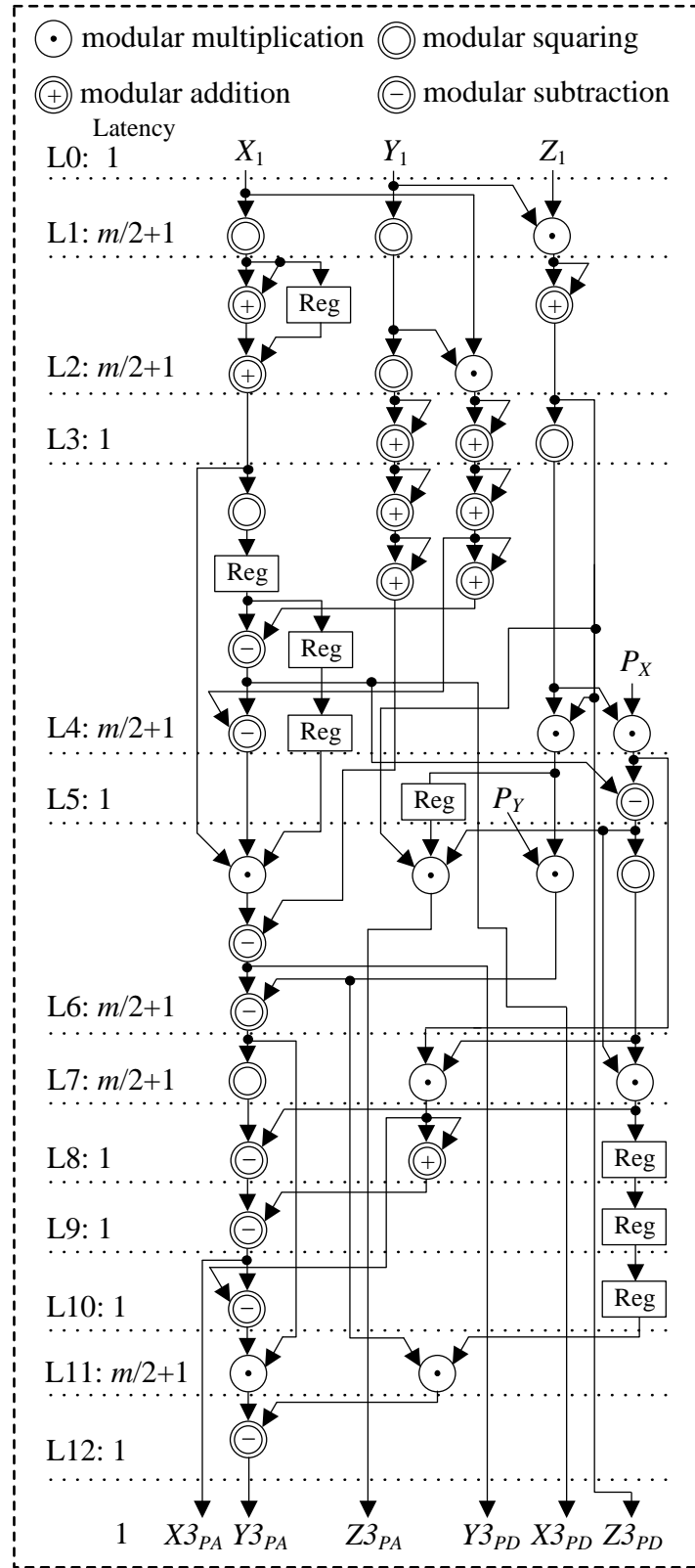


Figure 11.4: Proposed hardware architecture for PDPA.

X_{3P_A} , Y_{3P_A} , and Z_{3P_A} represent the PA output in projective coordinates. This module computes point doubling and point addition simultaneously. For example, if we provide $1P(X_1, Y_1, Z_1)$ in Jacobian coordinates, then this module produce the $2P(2P_X, 2P_Y, 2P_Z)$ and $3P(3P_X, 3P_Y, 3P_Z)$ values at the same time. Using this compact PDPA hardware, we have designed a high-performance ECP in Jacobian coordinates which is discussed in the next section.

11.6 Proposed ECC Processor

The key operation of an ECC processor (ECP) is ECPM, which is computationally the most expensive operation. However, we have designed a high-performance ECPM using our developed PDPA and FFMA units. The basic operation of ECPM is defined as kP , where k is a positive integer and is the private/secret key and P is a point on the elliptic curve E defined over a prime field \mathbb{F}_p . Various methods exist for implementing ECPM: the double-and-add method, the Non-adjacent form (NAF) method, and the Montgomery method. The simplest way to implement ECC is the double-and-add method [2], shown in Algorithm 11.4. As can be seen from Algorithm 11.4, it iterates through each bit of k . Using this algorithm, we have implemented ECPM in Jacobian coordinates then converted to affine coordinates.

11.6.1 Proposed Architecture for ECPM

To implement a high-performance ECP in Jacobian coordinates, we have proposed a novel ECPM architecture, using our proposed PDPA unit, depicted in Fig. 11.5. Note that most ECC hardware implementations in the literature have used separate PD and PA modules, and require more clock cycles for EC group operations (PD and PA). Also, most ECC hardware has been implemented on FPGAs with only a few hardware implementa-

Algorithm 11.4: Double and add method (Left to right) for ECPM

Input: $k = (k_{m-1}, \dots, k_1, k_0)_2$, $P(X, Y, Z) \in E(\mathbb{F}_p)$

Output: $Q(X, Y, Z) = k.P(X, Y, Z)$, where $Q(X, Y, Z) \in E(\mathbb{F}_p)$

1. $Q = 0$;
 2. **for** $i = m - 1$ **to** 0 **do** $Q = 2Q$;
 - 2.2 **if** $k(i) = '1'$ **then** $Q = Q + P$; **end**
 - 2.3 **end for**
 3. **Return** $(Q(X, Y, Z))$
-

tions targeting an ASIC. The proposed hardware is synthesized using ASIC 65-nm CMOS technology. As shown in Fig. 11.5, the PDPA module performs the EC group operations together, and these results are stored in the first register. The Select logic module generates a two-bit ‘*sel2s*’ signal for the MUX1 module. In the MUX1 module, when ‘*sel2s* = 00’ then PA results from the register 1 are forwarded to the output, which is the same as the input of the MUX2 module. Similarly, when ‘*sel2s* = 01’ then $1P(P_X, P_Y, P_Z)$ results, and when ‘*sel2s* = 10’ then $2P(2P_X, 2P_Y, 2P_Z)$ results, which are pre-computed, go to the input of MUX2. Hence the inputs of the MUX2 module are the PD and PA results, and the output of this module is either PD or PA, depending upon the bit pattern of input ‘*key*’. When the particular bit of ‘*key*’ is one, then the PA result, otherwise the PD result, is stored in register 2. The counter module of this architecture acts as a control unit and decides when the results of this register will be passed to the next input of the PDPA module. There $m - 1$ iterations are required to compute the final result of this ECPM module, where each iteration needs $3m + 14$ clock cycles (for PDPA). The total number of clock cycles for computing ECPM in Jacobian coordinates is defined by (11.9). The final results of this ECPM module are in Jacobian coordinates and need conversion to obtain the result in affine coordinates.

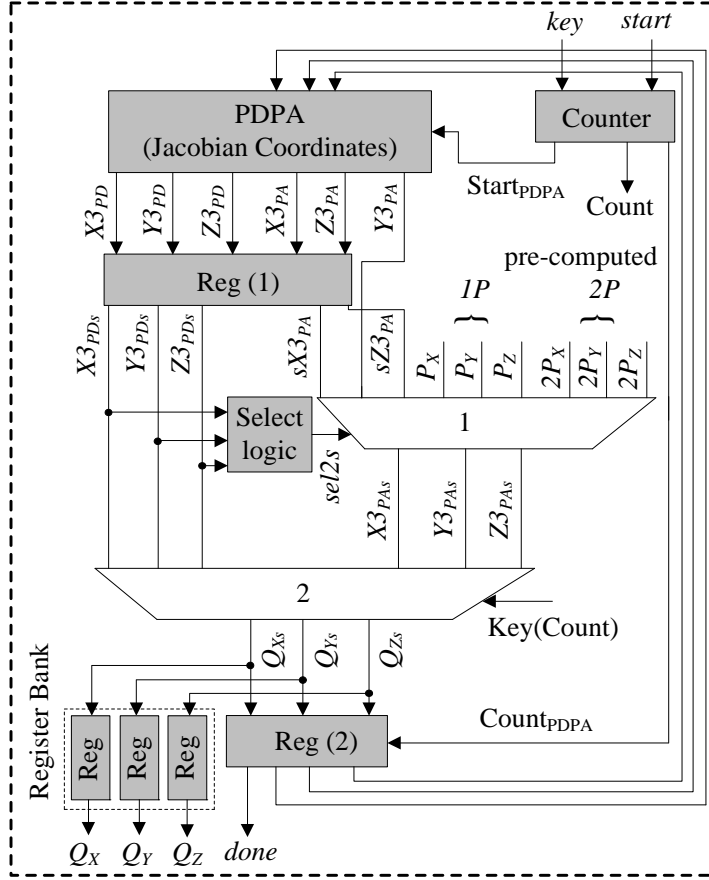


Figure 11.5: Detailed hardware architecture of proposed ECPM in Jacobian coordinates.

$$\begin{aligned}
 \# \text{clock cycles (CCs)} &= (m - 1) \times \text{PDPA CCs} \\
 &= (m - 1) \times (3m + 14) \\
 &= (3m^2 + 11m - 14)
 \end{aligned} \tag{11.9}$$

11.6.2 Jacobian to Affine Coordinates Conversion

For practical cryptography applications, affine coordinates are needed. We have designed a separate module for the Jacobian-to-affine coordinate conversion to increase the functionality of our proposed ECP. Fig. 11.6 shows proposed architectures for conversion from Jacobian to affine coordinates. As can be seen, $4m + 6$ cycles are required for Fig. 11.6(a) to get the result in affine coordinates, whereas Fig. 11.6(b) takes $5m + 4$ cycles due to

the two inversion modules. In this design, we have used our proposed radix-4 modular multiplication which takes only $m/2$ clock cycles while modular inversion is achieved in $2m$ clock cycles. For this reason, Fig. 11.6(a) is used for the proposed ECP. Also, we have implemented a very efficient inversion that may speed up the overall calculation of an ECP. Fig. 11.6(a) is a faster configuration than Fig. 11.6(b) due to the radix-4 modular multiplier; otherwise they have the same complexities. The cost of Fig. 11.6(a) is 4MUL, 1SQ, and 1INV, where MUL, SQ, and INV are the complexities of modular multiplication, squaring, and inversion respectively. Thus, to get the result in affine coordinates, only one modular inversion is needed for an ECPM.

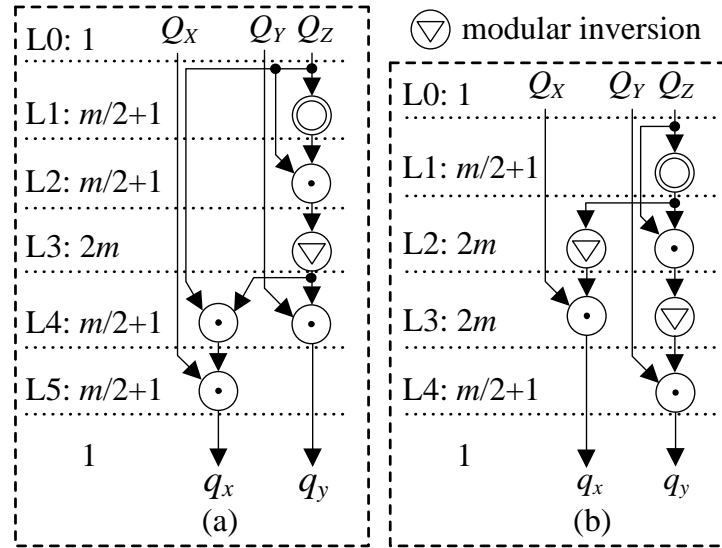


Figure 11.6: Proposed hardware architecture of Jacobian to affine conversion (a) using one inversion and (b) using two inversions.

11.6.3 Control Unit of ECPM

A control unit is the brain of any processor, that handles all control signals and processes all input and output flows. The control unit works by receiving input information, and generates control signals for different processing units. The detailed flow chart of the

control unit of ECPM in Jacobian coordinates is depicted in Fig. 11.7. The proposed controller is very simple and is for one computation of point multiplication. The control unit generates three primary control signals ' $Count_{PDPA}$ ', ' $Start_{PDPA}$ ', and ' $Count$ '. The ' $Count$ ' signal is initialized as $m - 2$ where m is the prime field bit length, e.g., for the prime field \mathbb{F}_{256} , $m = 256$. The ' $Count$ ' signal acts as a bit position of the input (i.e., key) which is a private key. The other two control signals are initialized with a zero value. Firstly, the controller checks whether ' $Count$ ' is $m - 2$, which is the maximum limit of this counter. If the condition is true, then the PDPA counter ' $Count_{PDPA}$ ' is checked. A zero value of ' $Count_{PDPA}$ ' indicates that the processor is in the idle state and waiting for the ' $Start$ ' signal for the next computation. The controller increments ' $Count_{PDPA}$ ' and set a ' $Start_{PDPA}$ ' to '1' when a high pulse is detected at the ' $Start$ ' signal.

' $Count_{PDPA}$ ' is checked in every clock cycle and incremented by one until it reaches the maximum limit of $3m + 13$. When ' $Count_{PDPA}$ ' reaches the maximum limit then it is again set to '0', ' $Start_{PDPA}$ ' to '1', and ' $Count$ ' to $m - 3$. In the next cycle, the main counter signal ' $Count$ ' is not $m - 2$ because this signal is already set to $m - 3$. The ' $Count_{PDPA}$ ' is again incremented in each clock cycle till it reaches the maximum limit of $3m + 13$. When ' $Count_{PDPA}$ ' reaches $3m + 13$ then it is again set to '0', ' $Start_{PDPA}$ ' is set high, and ' $Count$ ' is decremented. The counter ' $Count$ ' keeps decrementing in every iteration until it reaches zero, which indicates the completion of one point multiplication. Hence, the total number of clock cycles of one point multiplication are calculated as $(m - 1) \times (3m + 13) = 3m^2 + 11m - 14$.

After the completion of one point multiplication, the ' $Count_{PDPA}$ ', ' $Start_{PDPA}$ ', and ' $Count$ ' signals are set to '0', '0', and $m - 2$, respectively. The controller then waits for the ' $Start$ ' signal to perform the next computation.

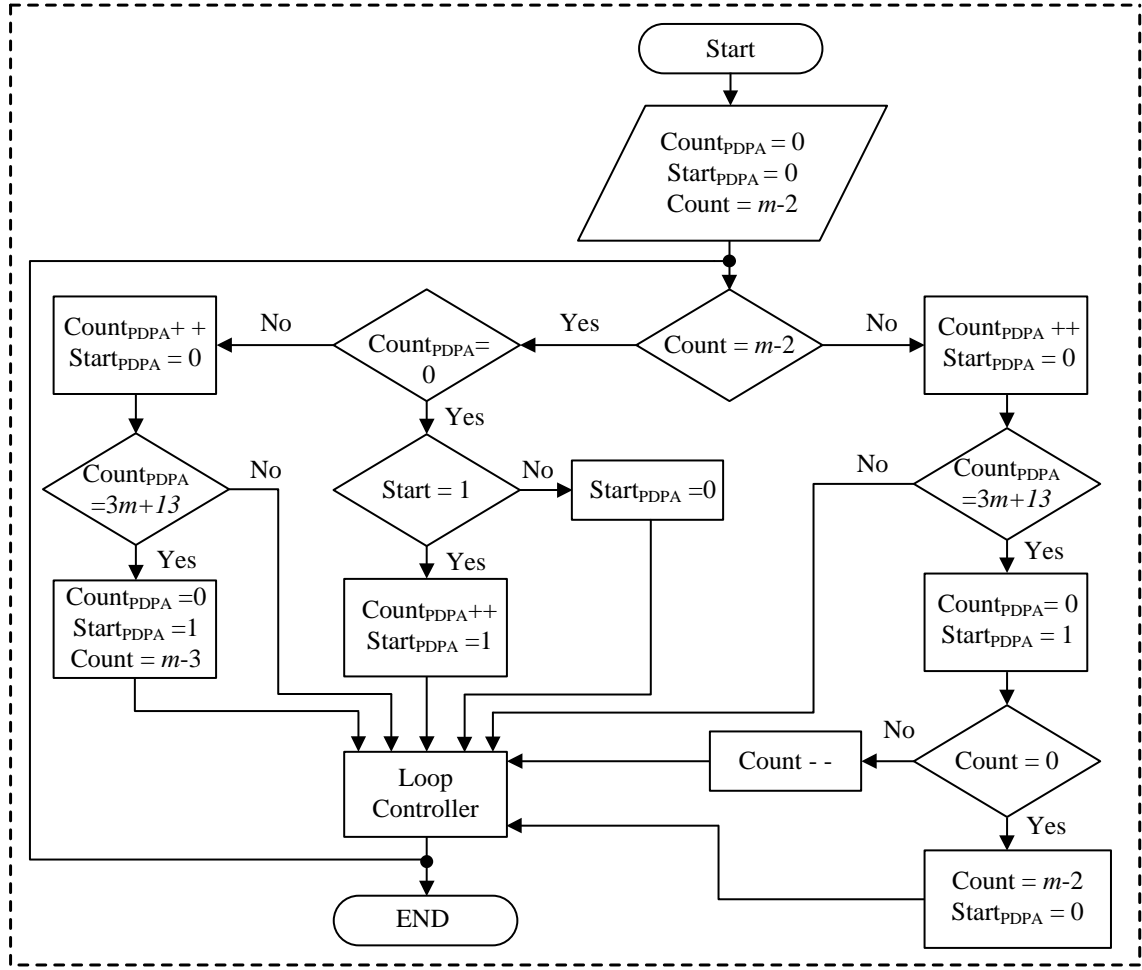


Figure 11.7: Proposed main controller of ECPM in Jacobian coordinates.

11.6.4 Overall architecture

Fig. 11.8 depicts the block diagram of the final ECP which is a building block of four modules, namely serial-in parallel-out (SIPO), ECPM in Jacobian coordinates, Jacobian to affine conversion, and parallel-in serial-out (PISO). The pin count of the top module is reduced by designing with SIPO and PISO, hence increasing the functionality of our proposed ECP. The first module is SIPO, which is required to send data serially and receive data in parallel. The output of the SIPO module is from 192 to 256 bits, depending on the prime field length. There only $m + 2$ clock cycles are required for the SIPO module. The

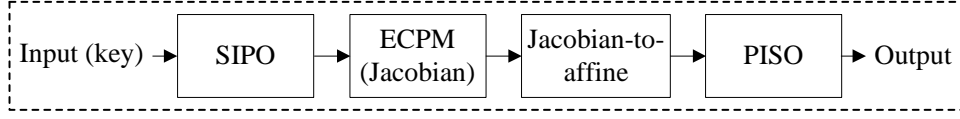


Figure 11.8: Overall block diagram of ECP top module.

second module is ECPM, which is the key operation of the ECP. The ECPM module in Jacobian coordinates needs only $3m^2 + 11m - 14$ cycles. The Jacobian to affine conversion module is the third module and is required to get the result in affine coordinates. To convert affine coordinates from Jacobian, only $4m + 6$ cycles are required. The final module is PISO, which sends the final results serially. This module takes only $m + 1$ clock cycles. Finally, the top module of ECP needs only five pins, two for input and three for output. Total number of cycles for ECP top module

$$\begin{aligned}
 &= \text{SIPO} + \text{ECPM in Jac.} + \text{Jacobian to Affine} + \text{PISO} \\
 &= (m + 1) + (3m^2 + 11m - 14) + (4m + 6) + (m + 2) \\
 &= (3m^2 + 17m - 5)
 \end{aligned} \tag{11.10}$$

11.6.5 ECDSA

The elliptic curve digital signature algorithm (ECDSA) is the most widely standardized elliptic curve based signature scheme. The execution time of ECDSA mostly relies on ECPM. Algorithm 11.5 shows the EC digital signature generation process on a message hash, where the EC domain parameters are given in [2, 22, 23]. In steps 1 and 2 of Algorithm 11.5, P is fixed, and the number k is secret (generated as random) due to ECPM in step2, where $R = kP$ is easy to compute, and the reverse way, which is called the elliptic curve discrete logarithm problem, is responsible for the hardness of ECC. When $R = 0$, called the point at infinity, then another key needs to be generated and return to step 1. In this algorithm, the pair (r, s) is the signature, where r is the x-coordinate of

Algorithm 11.5: Elliptic curve (EC) digital signature generation

Input: EC domain parameters, private key d , message hash z

Output: Signature (r, s)

1. Select $k \in [1, n - 1]$, random number generation
 2. Compute $R = k.P(Px, Py) \bmod p = (Rx, Ry) = \text{ECPM}$,
where $P(Px, Py)$ is the base point on EC
 3. Compute $r = Rx \bmod n$, where Rx is the x-coordinate of R
{if $r = 0$ then choose another key and start from step 1}
 4. Calculate $s = k^{-1}(z + rd) \bmod n$, where d is the private key
{ k^{-1} is the multiplicative inverse of k modulo n }
{if $s = 0$ then return to step 1}
 5. Return (r, s) , the signature is generated at this instant
-

Algorithm 11.6: Elliptic curve digital signature verification

Input: EC domain parameters, public key Q , message hash z

Output: Acceptance or rejection of the signature

1. Compute $w = s^{-1} \bmod n$
 2. Compute $u_1 = zw \bmod n$, and $u_2 = rw \bmod n$
 3. Calculate $X = u_1P + u_2Q \bmod p = (Xx, Xy)$
 4. if $X = \infty$ then return to step 1 ("Reject the signature")
 - 5 Compute $v = Xx \bmod n$, Xx is the x-coordinate of X
 - 6 Signature is valid only if $v = r$ else "Reject the signature"
-

R which is computed from ECPM, and if $r = 0$ then return to step 1. For the signature generation, we need to calculate s , shown in step 4 in Algorithm 11.5. For doing this, we can compute the multiplicative inverse modulo n with our modular inversion module, where n is the order of the base point P of EC. We have also implemented modular

multiplication and modular addition which can be used for signature generation.

The method for elliptic curve signature verification is illustrated in Algorithm 11.6. To verify a signature, we need to compute two point multiplications in step 3 such as $X = u_1P + u_2Q$, and the signature (r, s) , where P and Q are points on EC and P is fixed, but Q is not fixed. For doing this, modular inversion and modular multiplication are essential, as shown in steps 1 and 2 of Algorithm 11.6. In step 3 of this algorithm, ECPM and modular addition are required to compute the valid signature. When the value of X reaches infinity, i.e. $Xx = 0$, then the signature is invalid, and return to step 1. The signature is valid when $v = r$, where v is computed from the x-coordinate of X . We can say that our proposed ECP can be used for ECDSA because our ECP supports FFMA and point multiplication.

11.7 Implementation Results and Performance Comparison

Our proposed ECP is synthesized in ASIC 65-nm CMOS technology with 1.2V, 25°C. We have chosen 65-nm CMOS process technology for the proposed ECP, provided by STMicroelectronics. It was chosen because, after all layout verification, the chip can be manufactured using this process technology by a foundry. Usually, new technologies offer faster computation and lower power consumption with increased design possibilities. We believe that if we use 55-nm CMOS technology for our ECP, then it is possible to achieve faster computation as well as a more energy-efficient design. The design has been extensively simulated using ModelSim/Quarta Sim. All simulation results are verified using Maple software. It can compute the ECPM between 0.207 ms (192-bit ECC) and 0.366 ms (256-bit ECC) with an area between 1.10 mm² and 1.52 mm². We have achieved an energy-efficient design which consumes an energy of between 0.08 and 0.14 μ J at the

frequency of 549.45 MHz. The overall performance of the ECPM is compared with those in the available literature.

Table 11.1: *ASIC implementation of modular multiplication over \mathbb{F}_p .*

Field (Bit Length)	Area (mm ² /Kilo-Gates)	Time (ns) @f (MHz)	Throughput Rate (Mbps)	Energy (pJ/MM)
\mathbb{F}_p 192	0.043/20.67	176@549	1090	5.03
\mathbb{F}_p 224	0.050/24.18	206@549	1087	6.96
\mathbb{F}_p 256	0.059/28.40	216@549	1185	7.65

Used UMC standard logic cell library (65nm, 1.2V, 25°C), Kgs = kilo-gates

Modular addition and subtraction are implemented very efficiently; both operations take only one clock cycle. We have also implemented a combined modular addition and subtraction module. The combined module takes less area at the arithmetic level than separate addition and subtraction modules. However, the proposed PDPA employs dedicated adders and subtractors, therefore the combined modular addition/subtraction is not suitable. For this reason, we have utilized separate modular addition and subtraction modules for our proposed ECP. Both designs are synthesized using ASIC 65-nm CMOS technology with a target clock period of 1 ns. Thus the areas required by modular addition, subtraction, and combined addition and subtraction is only 0.01655 mm² (7.96 KGates), 0.0336 mm² (16.17 KGates), and 0.038 mm² (18.38 KGates) respectively. The gate count of this design is calculated as the total area divided by the NAND gate area of ST 65-nm technology, which is 2.08 μm^2 .

Most of the modular multiplication architectures were implemented on FPGAs. However, we have synthesized our proposed radix-4 modular multiplication (MM) (Algorithm 11.3, Fig. 11.3(b)) using STMicroelectronics 65-nm CMOS technology; results are shown in Table 11.1. It can compute modular multiplication in between 176 and 216 ns with an

area of 0.043 mm^2 to 0.059 mm^2 . The throughput rate of our design is more than 1 Gbps, which is very high for a prime field of \mathbb{F}_{192} or \mathbb{F}_{224} or \mathbb{F}_{256} . In [75], a modular multiplication over the prime field was implemented for an ECPM. They also used the interleaved modular multiplication method. However, their design takes m clock cycles for m -bit modular multiplication, which is almost double that of our design. Table 11.1 indicates that the proposed design is very fast, and area-efficient as well as energy-efficient.

We have also calculated the energy dissipation of all our designs, which is calculated from the power consumption and latency. The power consumption is simulated from Synopsys PrimeTime (PT). The Questa (ModelSim) simulator is used for the post-synthesis netlist with a sufficient amount of input data. From ModelSim, a vcd file is generated that captures all signal activities including net switching activity derived from a netlist and the time of every event on each net. Our generated vcd file gives a 99.7 % switching activity profile which is enough to get the accurate power consumption. Besides, the vcd-based power analysis is assumed extremely accurate because most of the factors considered to measure power consumption are supported in an exact form [187]. Our proposed radix-4 modular multiplication consumes less power than others. For example, the total power consumption for a 192-bit modular multiplication is $0.0286 \text{ mW}@549.45 \text{ MHz}$, of which 0.0156 mW is for nets, 0.0130 mW is for cells, and $0.0346 \mu\text{W}$ is for leakage. The energy dissipation is between 5.03 and 7.65 pJ per modular multiplication, which is very small. A high-performance ECPM is developed using this efficient modular multiplication.

Table 11.2 depicts the implementation results of modular inversion over \mathbb{F}_p . It can compute a modular inversion between 699 and 932 ns with the area between 0.071 mm^2 (33.91 KGates) and 0.096 mm^2 (46.13 KGates). Our designed modular inversion is very fast as well as area- and energy-efficient. The energy dissipation of modular inversion is between 15.34 and 27.20 pJ, almost 3 times as great as for modular multiplication, however only one modular inversion is required to complete ECPM.

Table 11.2: *ASIC implementation of modular inversion over \mathbb{F}_p .*

Field (Bit Length)	Area (mm ² /Kilo-Gates)	Time (ns) @f (MHz)	Throughput Rate (Mbps)	Energy (pJ/MM)
\mathbb{F}_p 192	0.071/33.91	699@549	275	15.34
\mathbb{F}_p 224	0.082/39.49	815@549	275	17.02
\mathbb{F}_p 256	0.096/46.13	932@549	275	27.20

Table 11.3: *ASIC synthesis results for PDPA over \mathbb{F}_p*

$ p $ (bits)	Area (mm ² /Kilo-Gates)	Clock Cycles	Time (μ s)	Throughput Rate (Mbps)	Energy (nJ/PDPA)
192	1.083/520.67	$3m+14$	1.07	179.44	0.33
224	1.310/629.81	$3m+14$	1.25	179.20	0.45
256	1.466/704.81	$3m+14$	1.42	180.28	0.49

$m \simeq \lceil \log_2 p \rceil$ and Frequency = 549 MHz

Table 11.3 presents the ASIC implementation results of PDPA, which is the most area-consuming operation of ECP; it consumes almost 97% of the area of the ECP. As shown in Table 11.3, the number of clock cycles for computing PDPA is only $3m + 14$, which is smaller than other available designs in the literature. For example, the design in [90] needs $5m + 13$ clock cycles for a point-addition operation. Based on our implementation results, we observe that the computation time for PDPA is much less due to the efficient implementation of FFMA. We have achieved a high throughput rate of almost 180 Mbps per PDPA operation. The power consumption is simulated from Synopsys PT, then the energy dissipation of PDPA calculated, is between 0.33 nJ and 0.49 nJ@549.45 MHz.

Table 11.4 shows the ECPM results and performance comparisons with similar designs over the NIST prime field \mathbb{F}_p . There are only a few hardware implementations on

Table 11.4: Performance comparison between our ECC design and similar ASIC-based designs over \mathbb{F}_p

Technology	Field (Length)	Reported Area (mm ² /KGates)	KCycles	Time (ms/ECPM) @f (MHz)	AT ¹	Energy ² (μ J/ECPM)	TR ³ (kpbs)
This work	65-nm	192	1.10/527.9	113.9	0.21@549	227.5/109.4	0.08
		224	1.34/644.2	154.3	0.28@549	376.4/181.0	0.11
		256	1.52/731.7	201.0	0.37@549	556.8/267.7	0.14
Liu [21]	55-nm	256	0.35/189.0	458.2	1.45@316	507.5/274.1	49.7
		192	1.12/313.0	94.2	0.43@220	481.6/134.6	26.0
		224	1.12/313.0	127.2	0.59@217	660.8/184.7	39.0
Lee [44]	90-nm	256	1.12/313.0	165.1	0.76@217	851.2/237.9	54.0
		160	1.35/179.0	54.4	0.38@141	513.0/68.0	31.0
		192	—/123.1	187.7	1.36@138	—/167.4	—
Ghosh [90]	0.13- μ m	224	—/143.9	253.5	1.95@130	—/280.6	—
		256	—/167.5	331.1	3.01@110	—/504.2	—

1. $AT = Area \times Time = (mm^2 \times \mu s) / (KGates \times ms)$, 2. $Energy = power \times time$, where power measured from Synopsys Prime Time (PT), and 3. $TR = Throughput Rate = (1/time(sec)) \times Field Length$

Table 11.4: *Performance comparison between our ECC design and similar ASIC-based designs over \mathbb{F}_p*

	Technology	Field (Length)	Reported Area (mm ² /KGates)	KCycles	Time (ms/ECPM) @f (MHz)	AT ¹	Energy ² (μ J/ECPM)	TR ³ (kbps)
Lai [169]	0.13- μ m	160	1.44/169.4	74.0	0.36@208	511.2/60.1	25.2	450.7
		256	—/197.0	252.1	1.21@208	—/238.4	—	211.6
Karakoyunlu [19]	0.18- μ m	160	—/86.0	264.6	1.80@147	—/154.8	—	88.9
		192	—/92.0	393.3	1.18@333	—/108.6	—	162.7
Ahmadi [178]	0.13- μ m	192	0.17/—	9712.5	525@18.5	89.3/—	20.6	0.4
Lai [185]	0.13- μ m	160	—/150.6	74.0	0.34@217	—/51.2	—	470.6
Chen [184]	0.13- μ m	256	—/122.0	562.0	1.01@556	—/123.2	—	253.5
Sato [186]	0.13- μ m	160	—/117.5	96.3	0.19@510	—/22.3	—	842.1
		192	—/118.0	198.3	1.44@138	—/169.9	—	133.3
		256	—/117.5	340.0	2.68@138	—/314.9	—	95.5

1. $AT = \text{Area} \times \text{Time} = (\text{mm}^2 \times \mu\text{s}) / (\text{KGates} \times \text{ms})$, 2. $\text{Energy} = \text{power} \times \text{time}$, where power measured from Synopsys Prime Time (PT), and 3. $TR = \text{Throughput Rate} = (1/\text{time}(\text{sec})) \times \text{Field Length}$

ASIC, most being implemented in FPGA. However, for a fair comparison, we tried to give all the ASIC implementation results over the prime field. The post-synthesis results are used to compare with other related ASIC-based implementations. Our designed ECP takes 0.207 ms, 0.281 ms, and 0.366 ms over the prime fields \mathbb{F}_{224} , \mathbb{F}_{256} , and \mathbb{F}_{256} , respectively, with a clock frequency of 549.45 MHz. As we can see from Table 11.4, we have achieved a higher clock frequency of 549 MHz because the proposed architectures are designed in such a way that the critical path is shorter, so they can run with a high clock frequency. Our proposed design requires comparable area (in mm^2) with recent ASIC-based implementations. Only a few designs calculated energy dissipation. The energy dissipation of the proposed ECPM for the prime field \mathbb{F}_p is between 0.08 and 0.14 μJ , which is 300 times better than other similar designs in the available literature.

An efficient dual-field ECP using a radix-4 modular arithmetic was implemented in ASIC 55-nm CMOS technology [21]. The gate count of our proposed ECC processor (ECP) is slightly larger than [21]. However, we have achieved high computation speed, hence the area \times time (AT) is comparable. Although our proposed ECP needs more area to implement, which may not be suitable for extremely lightweight applications, it is better for applications that require high throughput. Therefore, we have a tradeoff between area and computation time. Moreover, we have achieved an energy dissipation of only 0.35% that of their design. Therefore, the time-complexity and energy dissipation of our proposed design are much lower than [21]. An ECP using a heterogeneous dual-processing-element architecture was proposed in [44]. They used a double-and-add-always algorithm with separate PD and PA modules. They implemented in ASIC 90-nm CMOS technology. However, we have used PDPA and synthesized our design in ASIC 65-nm CMOS technology. Their design requires less area and logic gates than our design; on the other hand, we have achieved a faster and more energy-efficient design than [44]. The energy dissipation of their design is almost 350 times than our similar design.

[168] and [185] implemented a 160-bit ECPs in ASIC 0.13- μm CMOS technology. Their proposed ECP requires similar resources to our ECP with a more bits, as shown in Table 11.4. A parallel ECP over \mathbb{F}_p is presented in [90]; their design takes more clock cycles hence more computation time than ours. They synthesized their design on TSMC 0.13- μm CMOS technology using separate PD and PA modules. An efficient cipher processor for a dual-field ECP was proposed in [169]. They implemented their design in TSMC 0.13- μm CMOS technology and their design requires three times as much computation time as our design. The energy dissipation of our design is far better than for their design. [19] proposed efficient implementations of ECP over prime fields \mathbb{F}_{160} and \mathbb{F}_{192} . The gate count of their design in ASIC 0.18- μm CMOS technology is far less than for our design, but our design is faster than is given by their synthesized results. Ahmadi and Ali [178] proposed a low-power low-energy ECP over the prime field \mathbb{F}_{192} and synthesized it on 0.13- μm technology, and their cryptographic processor is much slower than our proposed ECP. Also, our design is 184 times better regarding energy dissipation. A high-performance 256-bit ECP for general curves over $\text{GF}(p)$ is proposed by Chen [184]. Their synthesized results show that they require more clock cycles and timing than our proposed design; however, their design requires fewer logic gates. Satoh et al. [186] proposed a scalable dual-field ECP for ASIC-based implementation. Their 256-bit ECP needs more clock cycles than our design; hence their design is slower than our design. Thus the throughput of our design is far better than their design. Besides, our proposed ECP is not only energy-efficient but also faster than all other designs.

11.8 Conclusion

A fast, area- and energy-efficient ECP over the prime field $\text{GF}(p)$ is proposed and synthesized using ASIC 65-nm CMOS technology. The double-and-add scalar multiplication

algorithm using a Jacobian coordinate is utilized for this implementation. The proposed architecture delivers a high-performance operation with less area. The proposed ECP architecture using combined group operation (PDPA) hardware supports three NIST curves with sizes 192, 224, and 256 bits, which are good enough for current security levels, even high-security applications. The same architecture can be used for the remaining two NIST curves whose prime sizes are 384 and 521 bits. However, the higher bit length takes more area, time, and power consumption compared to the lower field length. As can be seen from Table 11.4, 256-bit ECP needs more resources than 224-bit ECP. The required area is between 1.098 mm^2 (192-bit ECC) and 1.522 mm^2 (256-bit ECC) at the frequency of 549.45 MHz. Our proposed ECP takes between 0.207 and 0.366 ms to execute a typical point multiplication, which represents the fastest hardware implementation. We have also calculated the energy dissipation from Synopsys PT using the signal activity file; it is between 0.08 and $0.14 \mu\text{J}$ for the mentioned prime fields. The design is simulated using Modelsim PE and verified using Maple software. From the performance analysis and comparison of different ECPs over the prime field in Table 11.4, some are only area-efficient, or some are better regarding only speed; however, we have a trade-off between area, speed, and energy. It can be concluded that our ECP performs better than other comparable designs.

Chapter 12

Conclusions and Future Work

12.1 Conclusions

This dissertation presented research on hardware implementations of an elliptic curve cryptography processor (ECP) both in binary fields and prime fields. Several high-performance designs were proposed with their performances investigated to achieve high speed, low area, and low power consumption hardware of ECPs. The proposed efficient ECPs can be used for different resource-constrained devices and portable applications, such as smart cards, credit cards, smartphones, tablets, notebooks, and PDAs. In addition, the proposed ECPs were developed with minimum hardware resources as well as low energy dissipation with a high throughput rate, are suitable for IoT hardware security. In this dissertation, numerous hardware implementations of modular arithmetic operations, elliptic curve group operations, and elliptic curve point multiplications (ECPMs) were designed and implemented on different platforms. Based on the different implementation results with different platforms, it can be concluded that it is not easy to say which design is better. Therefore, the performance analyses of all designs were carefully investigated for fair comparisons.

12.1.1 Conclusions for Binary Field Elliptic Curve Cryptography

The first half of this dissertation is based on binary-field $GF(2^m)$ hardware implementations. In Sections 5.3, 6.3, and 7.4, the detailed hardware implementation hierarchy of the ECC operations over the binary field was presented. As can be seen from that, the bottom level in the hierarchy of ECC operations is the finite field arithmetic (FFA) unit, consisting of finite field addition, multiplication, squaring, and inversion, which is the most important operation to speed up the whole computation of ECP. In this dissertation, a high-performance hardware implementation of a finite field arithmetic (FFA) unit was designed to implement efficient ECPs over NIST binary fields $GF(2^m)$. The proposed FFA unit was designed to support all five binary fields $GF(2^{163})$, $GF(2^{233})$, $GF(2^{283})$, $GF(2^{409})$, and $GF(2^{571})$ recommended by NIST. In this FFA unit, three different types of finite field multiplier architectures were designed to achieve low-latency ECPM over the binary field: (1) bit-serial, (2) traditional digit-serial, and (3) modified digit-serial; whose computational complexity in terms of number of clock cycles are m , $\lceil m/d \rceil$, and $2 * \lceil \sqrt{m/d} \rceil$, respectively. In addition, a bit-serial finite field inversion architecture was designed for an ECP in affine coordinates or with conversion from Jacobian projective coordinates to affine coordinates, whose latency (clock cycles) is $2m + 1$. All finite field operations were implemented on both FPGA and ASIC (65-nm CMOS) platforms. Implementation results demonstrated that the bit-serial finite field multiplication needs only 98 to 409 slices and 105 to 453 slices in Virtex-7 FPGA and Virtex-6 FPGA, respectively. On the other hand, it was illustrated that the traditional digit-serial multiplication takes between 1975 and 7046 slices and between 2011 and 7129 slices in Virtex-7 and Virtex-6 FPGAs, respectively. It was noticed that the proposed modified digit-serial multiplications need a similar area to the traditional digit-serial multiplications but take less computation time. Note that the digit sizes 1, 2, 4, 8, 16, 32, and 64 bits were chosen for the finite field multiplication. It can be noted that the best way to compare

the performance of different designs is the area \times time (AT) or the reciprocal of AT, which is called efficiency (or performance). The implementation results revealed that digit-serial multiplications provide better efficiency than bit-serial multiplications, and the efficiency of traditional digit-serial multiplication is similar to modified digit-serial multiplication on an FPGA. However, the ASIC-based implementation results showed that the modified digit-serial multiplications provide far better performance (e.g. area \times time \times energy (ATE)) than the modified digit-serial multiplications. On the other hand, it was demonstrated that inversion takes more computation time, hence lower efficiency, than multiplication. Hence, inversion should be avoided as much as possible. It can be avoided for point multiplication in projective coordinates. All finite field arithmetic operations implemented in this research were compared with related work in the literature. To the best of the author's knowledge, the proposed FFA unit gives better performance in terms of AT and ATE than all other comparable work in the literature.

All the designed ECPs using bit-serial finite field arithmetic operations contained very low area and timing on an FPGA. The proposed ECPs were optimised by using different optimisation techniques, such as parallelisation on operations, pre-computations, balancing the elliptic curve group operations (i.e. PD and PA), and an efficient ECPM algorithm. Implementation results revealed that the ECPs in affine coordinates need only 2253, 3016, and 4625 slices without using any DSP blocks in a Xilinx Kintex-7 FPGA for the binary field $GF(2^{163})$, $GF(2^{233})$, and $GF(2^{283})$, respectively. It was shown that the computation time for 163, 233, and 283 bit binary-field ECPs are 1.06, 2.66, and 5.54 ms at the frequency of 306.48, 255.66, and 251.98 MHz, respectively. It was found that the proposed designs perform nearly 50% better than other than those in comparable work in the literature. On the other hand, it was noticed that the ECPM in affine coordinates with bit-serial arithmetic usually take a huge number of clock cycles, hence more computation time. Besides, it was demonstrated that the performance of ECP in projective

coordinates is better than ECP in affine coordinates.

In this dissertation, ECPM over binary fields $GF(2^{163})$ and $GF(2^{233})$ was implemented using a parallel architecture and in Jacobian projective coordinates. A novel combined group operation, point doubling and point addition (PDPA), was designed using the proposed parallel finite field multiplication in a polynomial basis. The point multiplication was designed and implemented on both FPGA and ASIC platforms using the combined PDPA and parallel multiplier. The FPGA or ASIC implementation results demonstrated that the proposed parallel point multiplication is the fastest hardware implementation result reported in the literature to date. Implementation results reported that the point multiplication computation time is between 0.33 and 3.56 μs in both a Xilinx Virtex-7 FPGA and in 65-nm CMOS technology. However, it was noticed that the proposed design takes more area (slices) than other work. Hence, the area-delay product was compared with related work and it was found that the proposed design provides almost 50% better efficiency than other work in the literature. In addition, it was also found that the ATE value of this design is far better than for recent implementations.

The implementation results of the parallel point-multiplication architecture were investigated and it was found that it was not suitable for an area-efficient design. Therefore, an efficient ECP was designed using both bit-serial and digit-serial finite field multipliers. The presented ECP using the bit-serial multiplier and the combined PDPA hardware supports all five NIST binary curves including both random and Koblitz curves. It was shown that the bit-serial approach ECP takes between 0.66 and 9.36 ms and between 156 and 1899 μs in a Virtex-7 FPGA and in 65-nm CMOS platforms, respectively for Koblitz curves from 163 to 571 bits. Similarly, it was demonstrated that the implemented designs need only 3056 to 13530 slices in a Virtex-7 FPGA and 115.5 to 396.9 kilo-gates in an ASIC 65-nm platform. Based on the performance analysis, the designed ECP in Jacobian projective coordinates with the bit serial multiplier is area efficient but not timing efficient.

Therefore, a high-performance ECP was implemented using the designed combined PDPA hardware and digit-serial multiplication hardware. Implementation results displayed that the FPGA-based design takes 37.52, 65.16, and 111.92 μs with 72937, 48652, and 25424 slices for digit sizes of 64, 32, and 16 bits, respectively over $GF(2^{233})$. Similarly, ASIC-based results showed that it takes only 69, 52, and 41 μs with 879.8, 557.7, and 326.9 kilo gate counts. It was noted that the ASIC-based ECP using a digit-serial multiplier gives better ATE performance than the ECP using a bit-serial multiplier. Finally, all designs were compared with related implementations in the literature, and it was found that the proposed designs deliver far better performance than recent implementations.

12.1.2 Conclusions for Prime Field Elliptic Curve Cryptography

The second half of this dissertation is based on prime-field $GF(p)$ hardware implementations. In this dissertation, a modular multiplier based on the Montgomery method and a modular inverter based on the extended Euclidean algorithm (EEA) (i.e. the binary method) were designed and implemented for an ECP over $GF(p)$. It was explained that modular multiplication is the most important arithmetic operation to implement ECP over prime fields. On the other hand, it was also noted that modular inversion is the most expensive operation over the prime field. For example, implementation results demonstrated that a 256-bit modular inversion needs 1480 slices with the computation time of 2.329 μs in a Virtex-7 FPGA, whereas a 256-bit modular multiplication contains only 605 slices with the delay of 1.683 μs in the same platform. However, it was explained that both designs are mandatory for an ECP over \mathbb{F}_p . Finally, their relative performances were described and compared in terms of area and timing with related work in the literature. It was demonstrated that the implemented designs are faster as well as more area-efficient than all other comparable work in the literature.

A high-performance hardware implementation of ECP over NIST prime fields \mathbb{F}_{192} and

\mathbb{F}_{256} was implemented in both affine coordinates and Jacobian projective coordinates. The point-multiplication architecture in affine coordinates using separate point doubling and point addition operations was explained clearly, and it was implemented in Kintex-7 and Virtex-5 FPGAs. It was shown that the computation time of ECP in a Kintex-7 FPGA is 3.05 and 4.70 ms with the area of 8.4 and 9.3 kilo-slices for prime fields \mathbb{F}_{224} and \mathbb{F}_{256} , respectively. Based on the implementation results, it was observed that the design in affine coordinates needs more clock cycles to implement, hence is a bit slow due to the modular inversion required for group operations. For this reason, the ECP was implemented in Jacobian projective coordinates, where inversion can be removed, but the ECP then needs more modular multiplication. Hence, an efficient modular multiplier architecture was proposed based on the interleaved method to get better performance. Also, a novel combined PDPA hardware was designed using the efficient modular multiplier. The new ECP was implemented in a Kintex-7 FPGA and ASIC 65-nm CMOS technology. It was reported that the ECP in Jacobian coordinates takes 2.36 and 3.27 ms with the area of 9.7 and 11.3 kilo-slices in a Kintex-7 FPGA for 224 and 256 bit ECPs, respectively. In addition, the ASIC-based results showed that the ASIC-based ECP offers far better delay performance than FPGA-based implementations. Also, the performance analyses were investigated carefully for both designs and it was found that the ECP in Jacobian coordinates provides better performance than the ECP in affine coordinates. Finally, the performance comparison was made with the related work in the literature, and it was found that the proposed designs in the prime field take at least 20% less computation time than the most significant work.

The final chapter of this dissertation discussed the fully ASIC-based ECP in Jacobian coordinates. It was presented that the latency of modular multiplication based on the interleaved method is $m + 1$, however, it is possible to save 50% on clock cycles by designing a novel radix-4 modular multiplier. The novel combined PDPA architecture, which

performs PD and PA operations together, was designed in Jacobian projective coordinates using the radix-4 multiplier. It was already explained that the performance of the ECP in Jacobian coordinates is better than the ECP in affine coordinates, hence the ECP in Jacobian coordinates only was considered for implementations. For this, a control unit for ECPM in Jacobian coordinates was designed and the working principle explained clearly. In addition, a separate conversion unit to convert from Jacobian to affine coordinates was designed to improve the functionality of the ECP over the prime field. It was recorded that the proposed ECP in ASIC 65-nm CMOS technology takes only 0.21, 0.28, and 0.37 ms with cycle counts of 113.9, 154.3, and 201.0, respectively for prime fields \mathbb{F}_{192} , \mathbb{F}_{224} , and \mathbb{F}_{256} , respectively. To the author's knowledge, the proposed ECP is the fastest hardware implementation over the prime field reported in the literature to date. It was also demonstrated that the proposed design is energy-efficient, taking between 0.08 and 0.14 μJ only per point multiplication, which is only 0.3% that of other similar designs. Therefore, the proposed ECP can be used in modern cryptographic hardware security applications.

12.2 Future Work

In future the following aspects of ECPs can be considered for further research and development:

- The finite field arithmetics, e.g. finite field addition, multiplication, squaring, and inversions, were implemented based on the polynomial basis. In future, all finite field operations will be designed using a normal basis or/and dual basis, then the performances of all arithmetic operations can easily be compared. In addition, all arithmetic operations can also be further optimised to get the best performance of ECP.

- In this dissertation, the ECP over the binary field was implemented using the traditional digit-serial multiplication. The implementation results obtained were limited to the binary field $GF(2^{233})$ with digit sizes of 16, 32, and 64 bits. This dissertation implemented both the traditional and the modified digit-serial multiplications with digit sizes of 1, 2, 4, 8, 16, 32, and 64 bits for all NIST binary curves $GF(2^{163})$, $GF(2^{233})$, $GF(2^{283})$, $GF(2^{409})$, and $GF(2^{571})$. In future, all digit-serial multiplications will be utilised to implement the ECP for all five Koblitz and random curves recommended by NIST.
- In this dissertation, digit-serial finite field multiplication with various digit sizes were implemented for all NIST binary curves. This dissertation implemented 233-bit ECP with digit sizes 16, 32, and 64 bits. In future, the same digit-serial multiplication will be utilised to implement the ECP for the remaining four NIST curves without degrading performance. However, it is noted that a higher field size requires more area and computation time. For example, a 571-bit ECP implementation needs more hardware resources than a 233-bit ECP implementation.
- The ECPs have been implemented using elliptic curve group operations of Koblitz and random curves only. The ECP using other curves including Edwards curves [188, 189] and the Hessian form of an elliptic curve [190] will be investigated in the future.
- The elliptic curve group operations were implemented in affine and Jacobian projective coordinates. In future, other forms of projective coordinates, including standard projective coordinates, the Lopez-Dahab projective coordinates, and Chudnovsky coordinates will be analysed.
- Inversion is the most complex arithmetic operation in either a binary field or a prime field. In this dissertation, it is implemented in a bit-serial approach. The

future research in this field will be focused on a higher radix inversion implementation. For example, higher-radix inversion based on the Itoh-Tsujii algorithm may be investigated for implementation.

- The ECPs over prime fields support three NIST prime curves of the five NIST-recommended primes p , with sizes 192, 224, and 256 bits. To obtain complete solutions of prime-field ECPs, hardware implementations of ECPM in the remaining NIST prime curves will be conducted.
- This dissertation implemented point multiplication based on the double-and-add algorithm (i.e. binary method). In addition, point multiplication using the Non-adjacent form (NAF) and the Montgomery method may be investigated in the future.
- In future, a complete architecture for elliptic curve digital signature algorithm (ECDSA) will be designed to support the execution of digital signature generation and verification.

Appendix A

Simulation Waveforms and Results

Sample

A.1 Simulation Results for Finite Field Arithmetic

A.1.1 Simulation waveform for Bit-Serial Multiplication

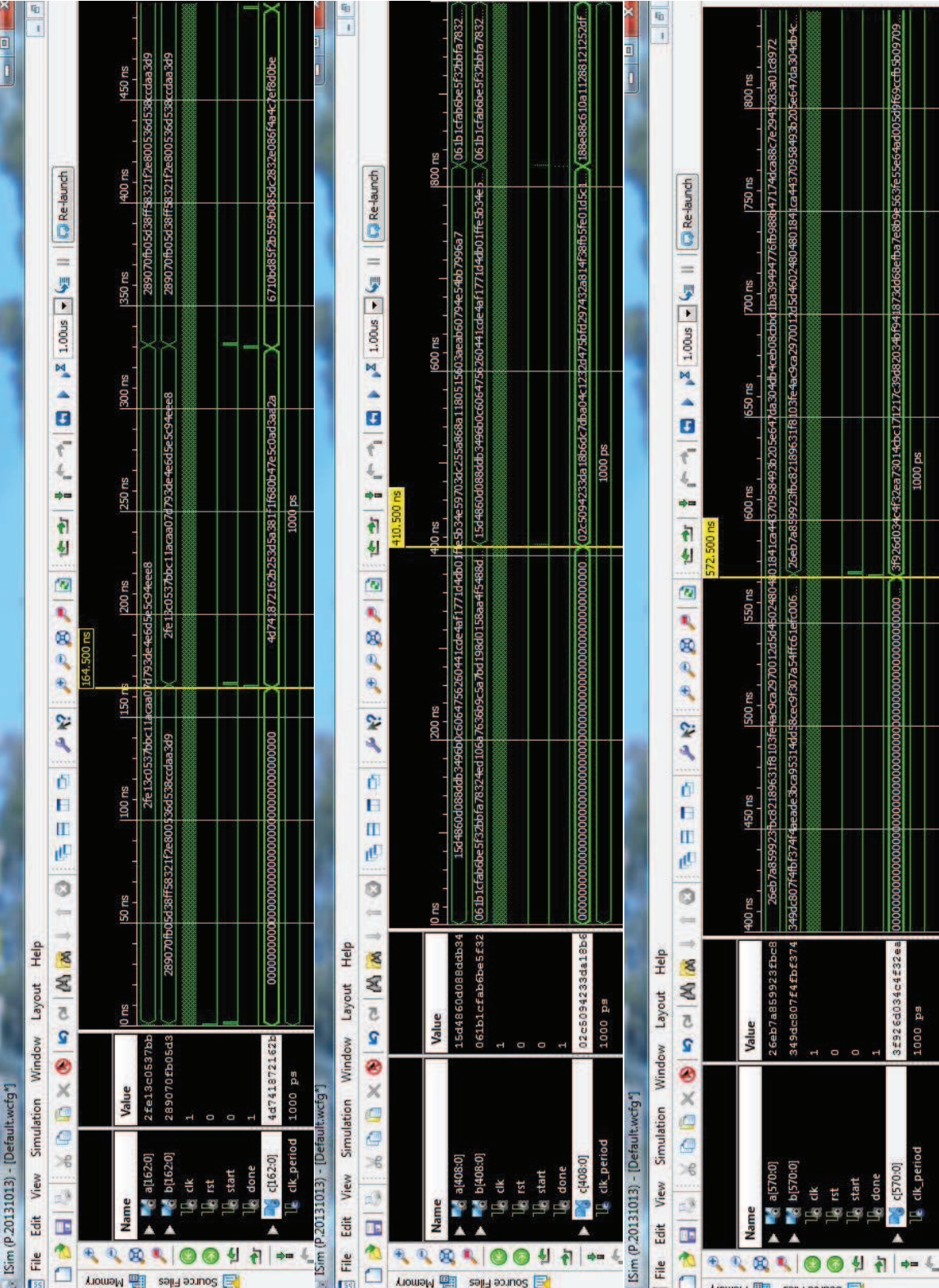


Figure A.1: Simulation waveform for bit-serial multiplier over \mathbb{F}_2^{163} , \mathbb{F}_2^{409} , and \mathbb{F}_2^{571} .

A.1.2 Simulation Results for Bit-Serial Multiplication

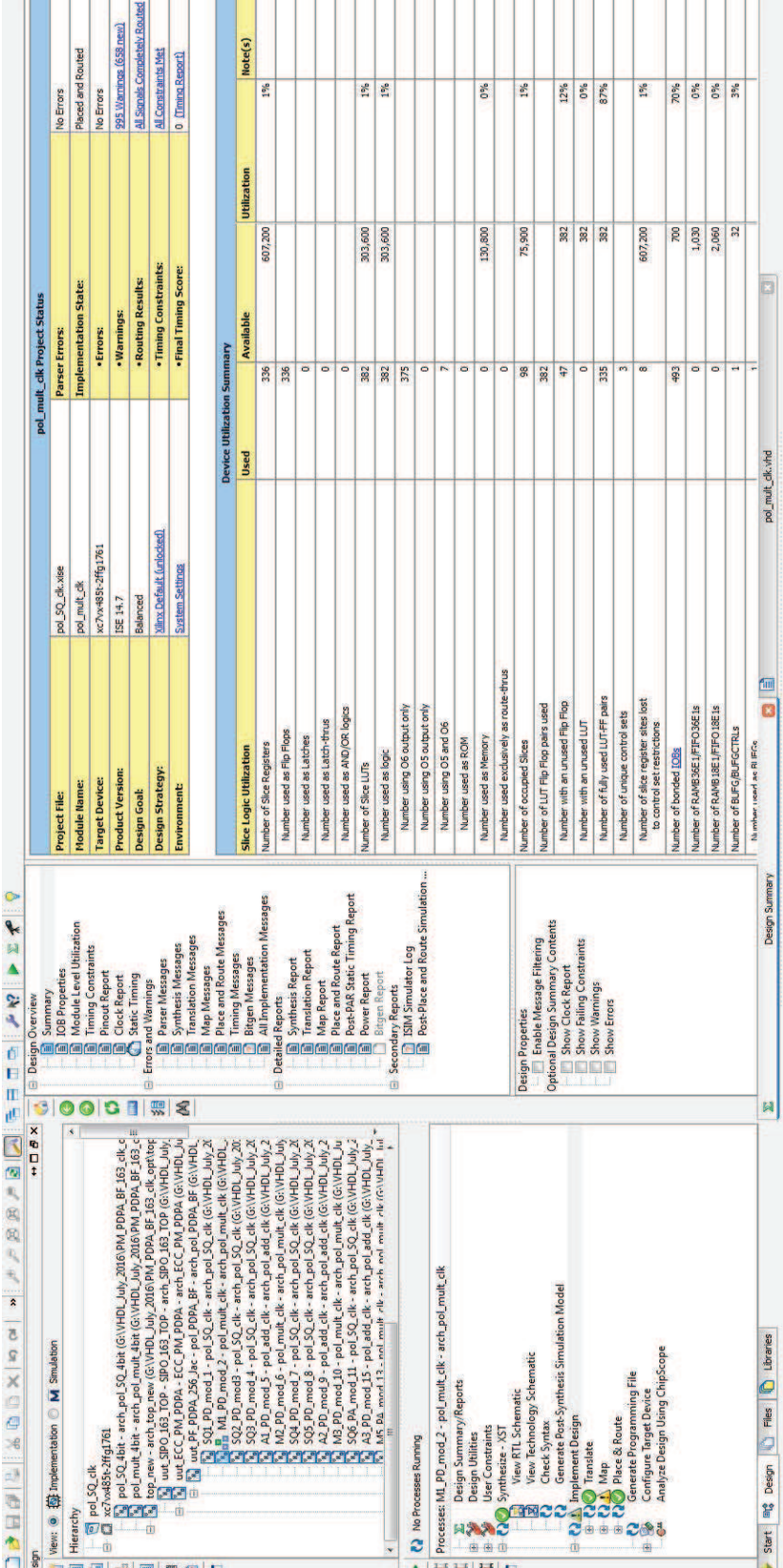


Figure A.2: Implementation results for bit-serial multiplier over \mathbb{F}_2^{163} in Virtex-7 FPGA.

A.1.3 Simulation Waveform and Results for Traditional Digit-Serial Multiplication

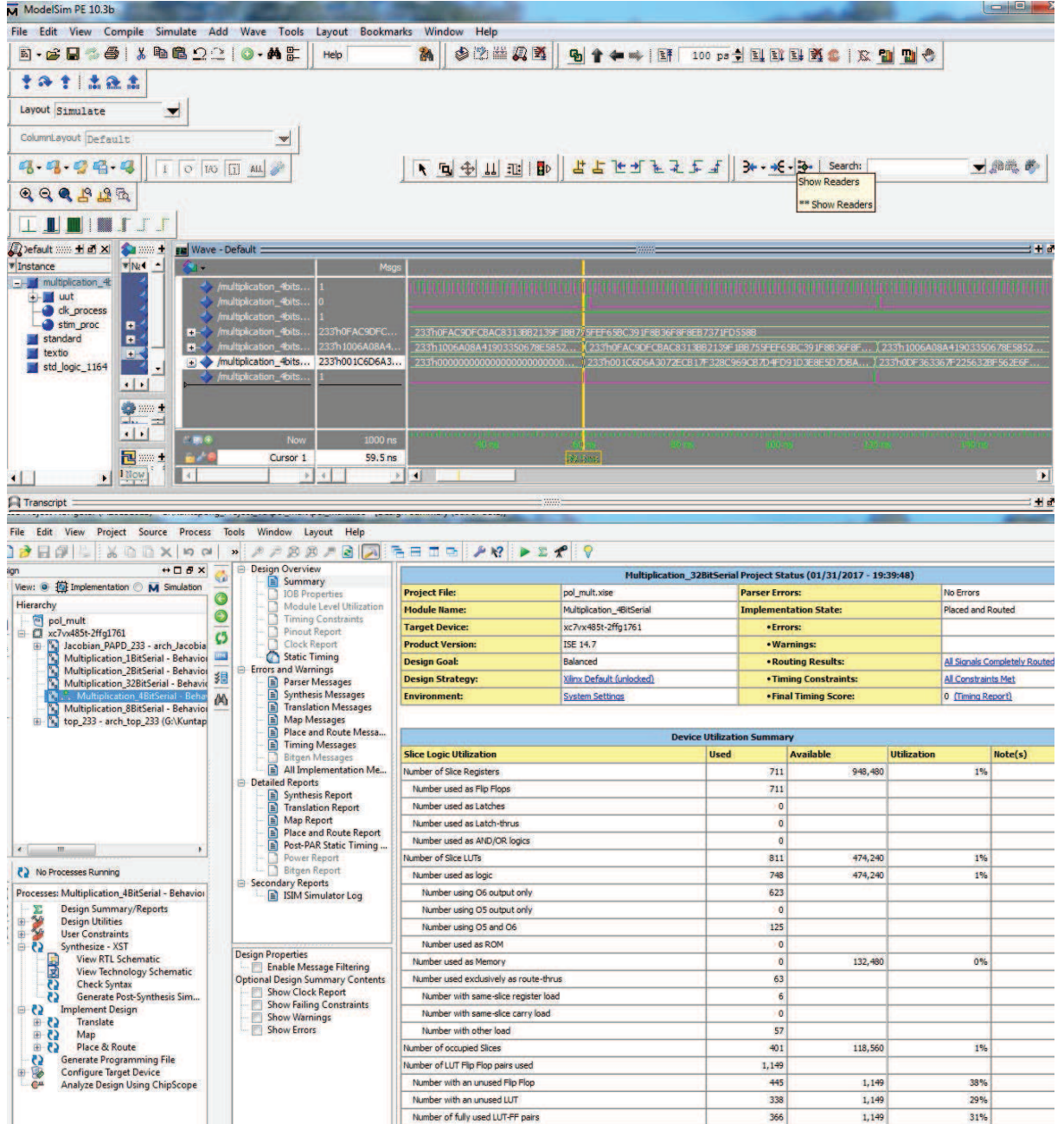


Figure A.3: Simulation waveform and implementation results in Virtex-7 FPGA for traditional digit-serial multiplier over \mathbb{F}_2^{233} with digit size of 4 bits.

A.1.4 Simulation Waveform and Results for Modified Digit-Serial Multiplication

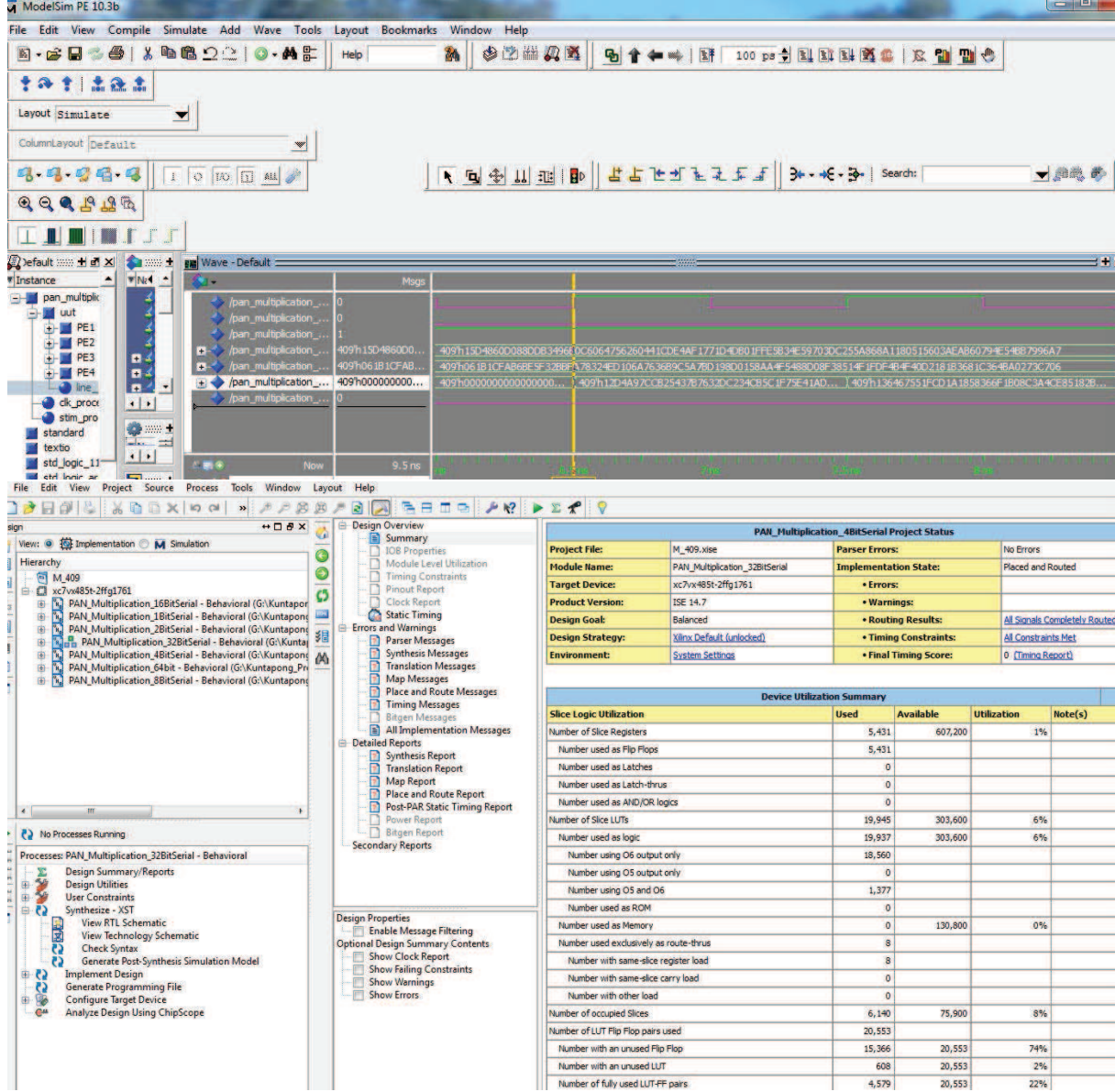


Figure A.4: Simulation waveform and implementation results in Virtex-7 FPGA for modified digit-serial multiplier over \mathbb{F}_2^{283} with digit size of 32 bits.

A.1.5 Simulation Waveform and Results for Inversion

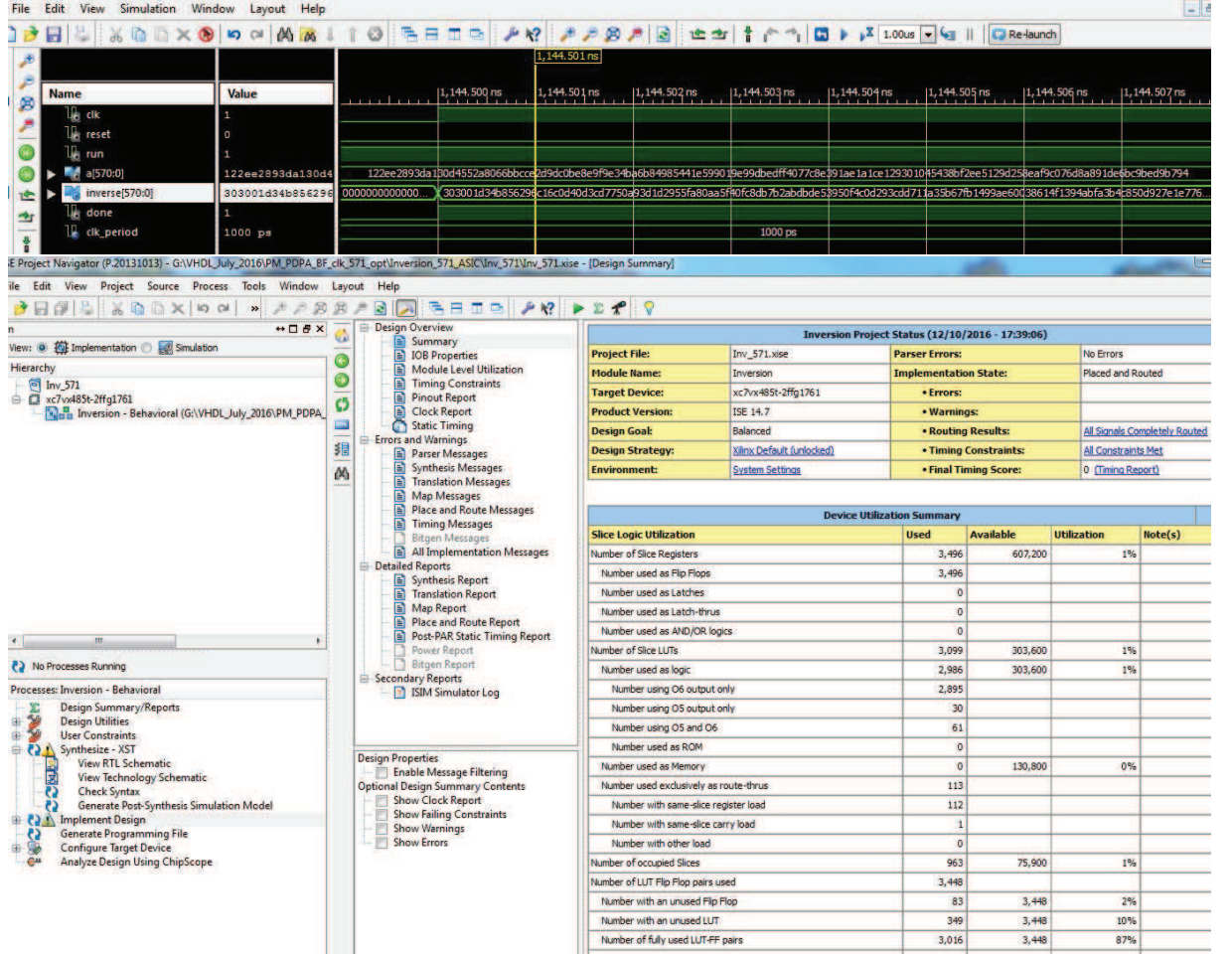


Figure A.5: Simulation waveform and implementation results in Virtex-7 FPGA for finite field inversion over \mathbb{F}_2^{571} .

A.2 Simulation Waveform and Results for ECC Processor over NIST Binary Fields

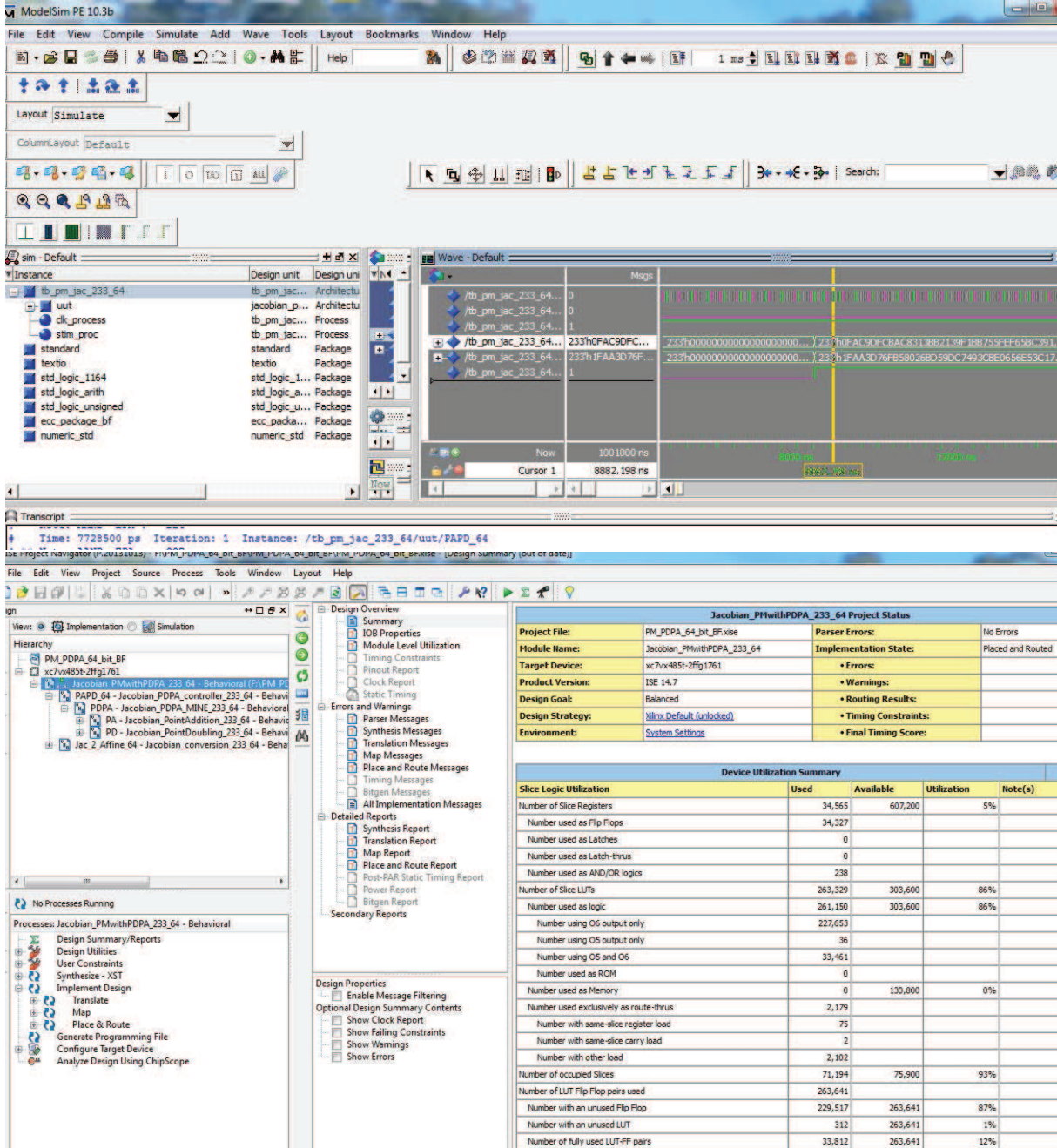


Figure A.6: Simulation waveform and implementation results in Virtex-7 FPGA for point multiplication over \mathbb{F}_2^{233} with digit size of 64 bits.

A.3 Simulation Waveform and Results for ECC Processor over Prime Fields

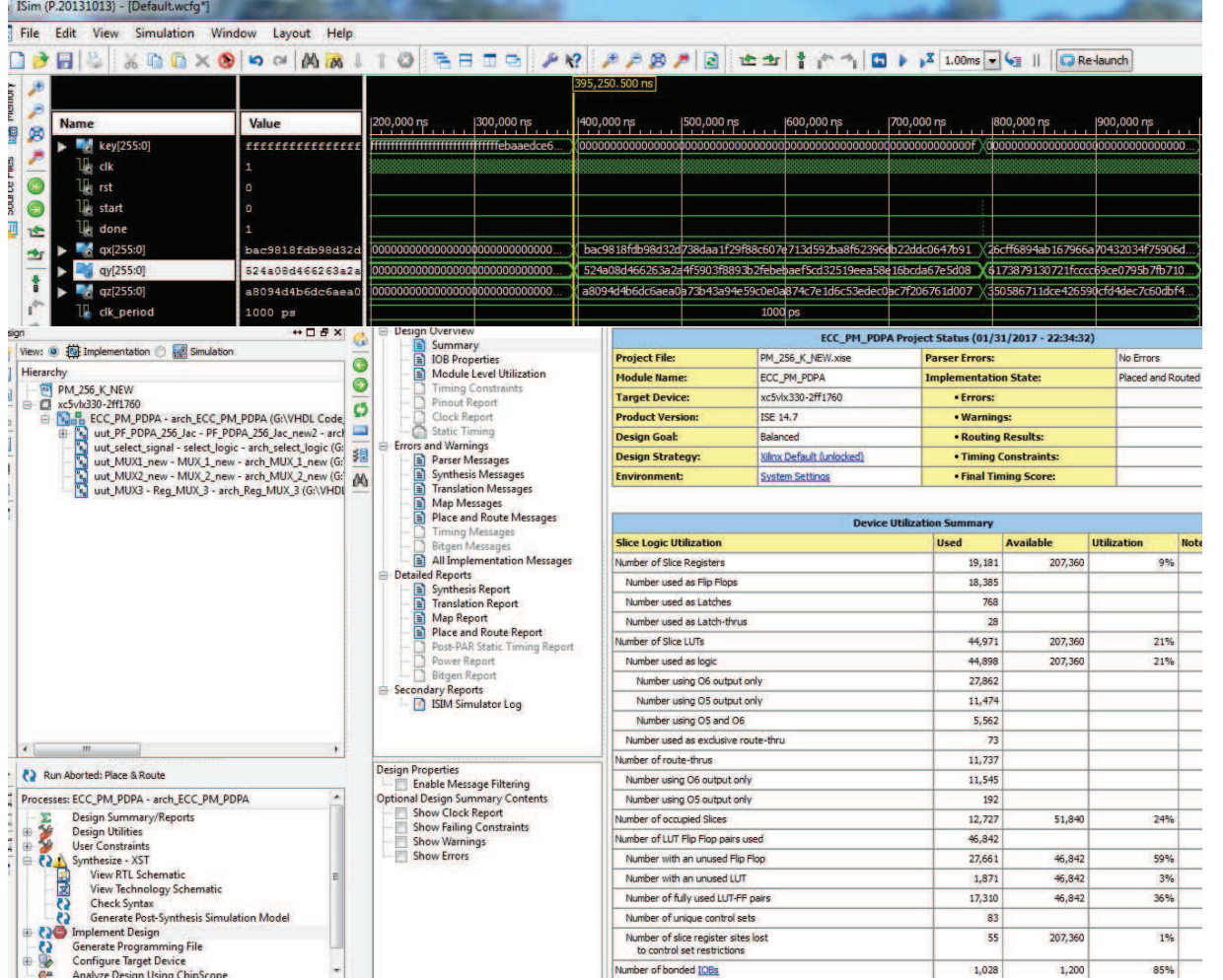


Figure A.7: Simulation waveform and implementation results in Virtex-5 FPGA for point multiplication over \mathbb{F}_{256} in Jacobian coordinates.

Appendix B

TCL Scripts Sample for Finite Field Arithmetic

B.1 Sample TCL scripts for bit-serial multiplier in Design Compiler

```
1 remove_design -all
2 analyze -format vhdl -lib work {/home/piraten/md2823ho/T_Mult_571_ASIC/syn/
   src/ECC_package_BF.vhd \
3 /home/piraten/md2823ho/T_Mult_571_ASIC/syn/src/pol_mult_clk.vhd}
4
5 elaborate pol_mult_clk -architecture arch_pol_mult_clk -library DEFAULT
   -update
6
7 #####
8 #Start of Compile
9 #####
10 create_clock "clk" -name "clk" -period 1
11 set_clock_uncertainty 0.1 clk
```

```

12 set_fix_hold clk
13 set_propagated_clock clk
14 set_input_delay 0 -clock clk [remove_from_collection [all_input] clk]
15 set_output_delay 0 -clock clk [all_outputs]
16 set_max_area 0
17 set_max_dynamic_power 0
18 set_max_leakage_power 0
19 set_load 1.5 [all_outputs]
20
21 compile -map_effort high
22 #####
23 #End of Compile
24 #####
25
26 report_timing > /home/piraten/md2823ho/T_Mult_571_ASIC/syn/reports/
    T_Mult_571_1_bit_serial_timing_DC.rpt
27 report_area -hierarchy > /home/piraten/md2823ho/T_Mult_571_ASIC/syn/reports
    /T_Mult_571_1_bit_serial_area_DC.rpt
28 report_power > /home/piraten/md2823ho/T_Mult_571_ASIC/syn/reports/
    T_Mult_571_1_bit_serial_power_DC.rpt
29
30 change_names -rules verilog -hierarchy > /dev/null
31 write -format verilog -hierarchy -output /net/cas-11/export/home/piraten/
    md2823ho/T_Mult_571_ASIC/syn/netlist/pol_mult_clk.v
32 write_sdf /net/cas-11/export/home/piraten/md2823ho/T_Mult_571_ASIC/syn/
    netlist/pol_mult_clk.sdf
33 write_sdc /net/cas-11/export/home/piraten/md2823ho/T_Mult_571_ASIC/syn/
    netlist/pol_mult_clk.sdc
34 write -format ddc -hierarchy -output /net/cas-11/export/home/piraten/
    md2823ho/T_Mult_571_ASIC/syn/netlist/pol_mult_clk.ddc

```

B.2 Sample TCL scripts for traditional digit-serial multiplier in Design Compiler

```

1 remove_design -all
2 analyze -format vhdl -lib work {/home/piraten/md2823ho/
   Traditional_Multipliers_233_ASIC/syn/src/ECC_package_BF.vhd \
3 /home/piraten/md2823ho/Traditional_Multipliers_233_ASIC/syn/src/
   Multiplication_8BitSerial.vhd}
4
5 elaborate Multiplication_8BitSerial -architecture
   arch_Multiplication_8BitSerial -library DEFAULT -update
6 #####
7 #Start of Compile
8 #####
9 create_clock "clk" -name "clk" -period 2
10 set_clock_uncertainty 0.1 clk
11 set_fix_hold clk
12 #set_propagated_clock clk
13 set_input_delay 0 -clock clk [remove_from_collection [all_input] clk]
14 set_output_delay 0 -clock clk [all_outputs]
15 set_max_area 0
16 set_max_dynamic_power 0
17 set_max_leakage_power 0
18 set_load 1.5 [all_outputs]
19
20 compile -map_effort high
21 #####
22 #End of Compile
23 #####
24 report_timing > /home/piraten/md2823ho/Traditional_Multipliers_233_ASIC/syn
   /reports/T_Mult_233_8_timing_DC.rpt

```

```

25 report_area -hierarchy > /home/piraten/md2823ho/
    Traditional_Multipliers_233_ASIC/syn/reports/T_Mult_233_8_area_DC.rpt
26 report_power > /home/piraten/md2823ho/Traditional_Multipliers_233_ASIC/syn/
    reports/T_Mult_233_8_power_DC.rpt
27
28 change_names -rules verilog -hierarchy > /dev/null
29 write -format verilog -hierarchy -output /net/cas-11/export/home/piraten/
    md2823ho/Traditional_Multipliers_233_ASIC/syn/netlist/
    Multiplication_8BitSerial.v
30 write_sdf /net/cas-11/export/home/piraten/md2823ho/
    Traditional_Multipliers_233_ASIC/syn/netlist/
    Multiplication_8BitSerial.sdf
31 write_sdc /net/cas-11/export/home/piraten/md2823ho/
    Traditional_Multipliers_233_ASIC/syn/netlist/
    Multiplication_8BitSerial.sdc
32 write -format ddc -hierarchy -output /net/cas-11/export/home/piraten/
    md2823ho/Traditional_Multipliers_233_ASIC/syn/netlist/
    Multiplication_8BitSerial.ddc

```

B.3 Sample TCL scripts for modified digit-serial multiplier in Design Compiler

```

1 remove_design -all
2 analyze -format vhdl -lib work {/home/piraten/md2823ho/M_Mult_409_ASIC/syn/
    src/ECC_package_BF.vhd \
3 /home/piraten/md2823ho/M_Mult_409_ASIC/syn/src/PE_32BitSerial.vhd \
4 /home/piraten/md2823ho/M_Mult_409_ASIC/syn/src/
    Modified_Multiplication_32BitSerial.vhd}
5
6 elaborate Modified_Multiplication_32BitSerial -architecture Behavioral
    -library DEFAULT -update

```

```
7 #####
8 #Start of Compile
9 #####
10 create_clock "clk" -name "clk" -period 1
11 set_clock_uncertainty 0.1 clk
12 set_fix_hold clk
13 #set_propagated_clock clk
14 set_input_delay 0 -clock clk [remove_from_collection [all_input] clk]
15 set_output_delay 0 -clock clk [all_outputs]
16 set_max_area 0
17 set_max_dynamic_power 0
18 set_max_leakage_power 0
19 set_load 1.5 [all_outputs]
20
21 compile -map_effort high
22 #####
23 #End of Compile
24 #####
25 report_timing > /home/piraten/md2823ho/M_Mult_409_ASIC/syn/reports/
    M_Mult_409_32_timing_DC.rpt
26 report_area -hierarchy > /home/piraten/md2823ho/M_Mult_409_ASIC/syn/reports
    /M_Mult_409_32_area_DC.rpt
27 report_power > /home/piraten/md2823ho/M_Mult_409_ASIC/syn/reports/
    M_Mult_409_32_power_DC.rpt
28
29 change_names -rules verilog -hierarchy > /dev/null
30 write -format verilog -hierarchy -output /net/cas-11/export/home/piraten/
    md2823ho/M_Mult_409_ASIC/syn/netlist/
    Modified_Multiplication_32BitSerial.v
31 write_sdf /net/cas-11/export/home/piraten/md2823ho/M_Mult_409_ASIC/syn/
    netlist/Modified_Multiplication_32BitSerial.sdf
```

```

32 write_sdc /net/cas-11/export/home/piraten/md2823ho/M_Mult_409_ASIC/syn/
    netlist/Modified_Multiplication_32BitSerial.sdc
33 write -format ddc -hierarchy -output /net/cas-11/export/home/piraten/
    md2823ho/M_Mult_409_ASIC/syn/netlist/
    Modified_Multiplication_32BitSerial.ddc

```

B.4 Sample TCL scripts for inverter in Design Compiler

```

1 remove_design -all
2 analyze -format vhdl -lib work {/home/piraten/md2823ho/
    Inversion_283_BF_ASIC/syn/src/ECC_package_BF.vhd \
3 /home/piraten/md2823ho/Inversion_283_BF_ASIC/syn/src/Inversion_283_BF.vhd}
4
5 elaborate Inversion_283_BF -architecture Arch_Inversion_283_BF -library
    DEFAULT -update
6
7 #####
8 #Start of Compile
9 #####
10 create_clock "clk" -name "clk" -period 1
11 set_clock_uncertainty 0.1 clk
12 set_fix_hold clk
13 ##set_propagated_clock clk
14 set_input_delay 0 -clock clk [remove_from_collection [all_input] clk]
15 set_output_delay 0 -clock clk [all_outputs]
16 set_max_area 0
17 set_max_dynamic_power 0
18 set_max_leakage_power 0
19 set_load 1.5 [all_outputs]
20

```



```
21 compile -map_effort high
22
23 #####
24 #End of Compile
25 #####
26
27 report_timing > /home/piraten/md2823ho/Inversion_283_BF_ASIC/syn/reports/
    Inversion_283_BF_timing_DC.rpt
28 report_area -hierarchy > /home/piraten/md2823ho/Inversion_283_BF_ASIC/syn/
    reports/Inversion_283_BF_area_DC.rpt
29 report_power > /home/piraten/md2823ho/Inversion_283_BF_ASIC/syn/reports/
    Inversion_283_BF_power_DC.rpt
30
31
32 change_names -rules verilog -hierarchy > /dev/null
33 write -format verilog -hierarchy -output /net/cas-11/export/home/piraten/
    md2823ho/Inversion_283_BF_ASIC/syn/netlist/Inversion_283_BF.v
34 write_sdf /net/cas-11/export/home/piraten/md2823ho/Inversion_283_BF_ASIC/
    syn/netlist/Inversion_283_BF.sdf
35 write_sdc /net/cas-11/export/home/piraten/md2823ho/Inversion_283_BF_ASIC/
    syn/netlist/Inversion_283_BF.sdc
36 write -format ddc -hierarchy -output /net/cas-11/export/home/piraten/
    md2823ho/Inversion_283_BF_ASIC/syn/netlist/Inversion_283_BF.ddc
```


Appendix C

Sample TCL Scripts for Elliptic Curve Cryptography Processor Over NIST Binary Fields

C.1 Sample TCL scripts for ECC Processor over \mathbb{F}_2^{571} using bit-serial multiplier in Design Compiler

```
1 remove_design -all
2 analyze -format vhdl -lib work {/home/piraten/md2823ho/ECC_B_571_clk_opt/
   syn/src/ECC_package_BF.vhd \
3 /home/piraten/md2823ho/ECC_B_571_clk_opt/syn/src/pol_add_clk.vhd \
4 /home/piraten/md2823ho/ECC_B_571_clk_opt/syn/src/pol_SQ_clk.vhd \
5 /home/piraten/md2823ho/ECC_B_571_clk_opt/syn/src/pol_mult_clk.vhd \
6 /home/piraten/md2823ho/ECC_B_571_clk_opt/syn/src/Reg_PDPA_new.vhd \
7 /home/piraten/md2823ho/ECC_B_571_clk_opt/syn/src/pol_PDPA_BF.vhd \
8 /home/piraten/md2823ho/ECC_B_571_clk_opt/syn/src/select_logic.vhd \
9 /home/piraten/md2823ho/ECC_B_571_clk_opt/syn/src/MUX_1_new.vhd \
```

```

10 /home/piraten/md2823ho/ECC_B_571_clk_opt/syn/src/MUX_2_new.vhd \
11 /home/piraten/md2823ho/ECC_B_571_clk_opt/syn/src/Reg_MUX_3.vhd \
12 /home/piraten/md2823ho/ECC_B_571_clk_opt/syn/src/ECC_PM_PDPA.vhd \
13 /home/piraten/md2823ho/ECC_B_571_clk_opt/syn/src/SIPO_571_TOP.vhd \
14 /home/piraten/md2823ho/ECC_B_571_clk_opt/syn/src/PISO_571.vhd \
15 /home/piraten/md2823ho/ECC_B_571_clk_opt/syn/src/top_new_571.vhd}
16
17 elaborate top_new_571 -architecture arch_top_new_571 -library DEFAULT
   -update
18
19 #####
20 #Start of Compile
21 #####
22 create_clock "clk" -name "clk" -period 1
23 set_clock_uncertainty 0.1 clk
24 set_fix_hold clk
25 set_input_delay 0 -clock clk [remove_from_collection [all_input] clk]
26 set_output_delay 0 -clock clk [all_outputs]
27 set_max_area 0
28 set_max_dynamic_power 0
29 set_max_leakage_power 0
30 set_load 1.5 [all_outputs]
31
32 compile -map_effort high
33 #####
34 #End of Compile
35 #####
36 report_timing > /home/piraten/md2823ho/ECC_B_571_clk_opt/syn/reports/
   top_new_571_timing_B571_DC.rpt
37 report_area -hierarchy > /home/piraten/md2823ho/ECC_B_571_clk_opt/syn/
   reports/top_new_571_area_B571_DC.rpt

```

```

38 report_power > /home/piraten/md2823ho/ECC_B_571_clk_opt/syn/reports /
    top_new_571_power_B571_DC.rpt
39
40 change_names -rules verilog -hierarchy > /dev/null
41 write -format verilog -hierarchy -output /net/cas-11/export/home/piraten /
    md2823ho/ECC_B_571_clk_opt/syn/netlist/top_new_571.v
42 write_sdf /net/cas-11/export/home/piraten/md2823ho/ECC_B_571_clk_opt/syn /
    netlist/top_new_571.sdf
43 write_sdc /net/cas-11/export/home/piraten/md2823ho/ECC_B_571_clk_opt/syn /
    netlist/top_new_571.sdc
44 write -format ddc -hierarchy -output /net/cas-11/export/home/piraten /
    md2823ho/ECC_B_571_clk_opt/syn/netlist/top_new_571.ddc

```

C.2 Sample TCL scripts for ECC Processor over \mathbb{F}_2^{233} using digit-serial multiplier in Design Compiler

```

1 remove_design -all
2 analyze -format vhdl -lib work {/home/piraten/md2823ho/
    PM_PDPA_233_bit_BF_64/syn/src/ECC_package_BF.vhd \
3 /home/piraten/md2823ho/PM_PDPA_233_bit_BF_64/syn/src/Addition.vhd \
4 /home/piraten/md2823ho/PM_PDPA_233_bit_BF_64/syn/src/Inversion.vhd \
5 /home/piraten/md2823ho/PM_PDPA_233_bit_BF_64/syn/src /
    Multiplication_64BitSerial.vhd \
6 /home/piraten/md2823ho/PM_PDPA_233_bit_BF_64/syn/src /
    Jacobian_PointDoubling_233_64.vhd \
7 /home/piraten/md2823ho/PM_PDPA_233_bit_BF_64/syn/src /
    Jacobian_PointAddition_233_64.vhd \
8 /home/piraten/md2823ho/PM_PDPA_233_bit_BF_64/syn/src /
    Jacobian_PDPA_MINE_233_64.vhd \
9 /home/piraten/md2823ho/PM_PDPA_233_bit_BF_64/syn/src /
    Jacobian_PDPA_controller_233_64.vhd \

```

```

10 /home/piraten/md2823ho/PM_PDPA_233_bit_BF_64/syn/src/
    Jacobian_conversion_233_64.vhd \
11 /home/piraten/md2823ho/PM_PDPA_233_bit_BF_64/syn/src/
    Jacobian_PMwithPDPA_233_64.vhd}
12
13 elaborate Jacobian_PMwithPDPA_233_64 -architecture
    arch_Jacobian_PMwithPDPA_233_64 -library DEFAULT -update
14
15 #####
16 #Start of Compile
17 #####
18 create_clock "clk" -name "clk" -period 1
19 set_clock_uncertainty 0.1 clk
20 set_fix_hold clk
21 set_input_delay 0 -clock clk [remove_from_collection [all_input] clk]
22 set_output_delay 0 -clock clk [all_outputs]
23 set_max_area 0
24 set_max_dynamic_power 0
25 set_max_leakage_power 0
26 set_load 1.5 [all_outputs]
27
28 compile -map_effort high
29 #####
30 #End of Compile
31 #####
32
33 report_timing > /home/piraten/md2823ho/PM_PDPA_233_bit_BF_64/syn/reports/
    Jacobian_PMwithPDPA_B233_64_timing_DC.rpt
34 report_area -hierarchy > /home/piraten/md2823ho/PM_PDPA_233_bit_BF_64/syn/
    reports/Jacobian_PMwithPDPA_B233_64_area_DC.rpt
35 report_power > /home/piraten/md2823ho/PM_PDPA_233_bit_BF_64/syn/reports/

```

```
Jacobian_PMwithPDPA_B233_64_power_DC.rpt
36
37 change_names -rules verilog -hierarchy > /dev/null
38 write -format verilog -hierarchy -output /net/cas-11/export/home/piraten/
    md2823ho/PM_PDPA_233_bit_BF_64/syn/netlist/Jacobian_PMwithPDPA_233_64.v
39 write_sdf /net/cas-11/export/home/piraten/md2823ho/PM_PDPA_233_bit_BF_64/
    syn/netlist/Jacobian_PMwithPDPA_233_64.sdf
40 write_sdc /net/cas-11/export/home/piraten/md2823ho/PM_PDPA_233_bit_BF_64/
    syn/netlist/Jacobian_PMwithPDPA_233_64.sdc
41 write -format ddc -hierarchy -output /net/cas-11/export/home/piraten/
    md2823ho/PM_PDPA_233_bit_BF_64/syn/netlist/
    Jacobian_PMwithPDPA_233_64.ddc
```


Appendix D

Sample TCL Scripts for ASIC-Based Elliptic Curve Cryptography Processor over Prime Fields

D.1 Sample TCL scripts for ECC Processor over \mathbb{F}_{256} in Design Compiler

```
1 remove_design -all
2 analyze -format vhdl -lib work {/home/piraten/md2823ho/ECC_new2_Radix_4_MM/
   syn/src/SIPO_256_TOP.vhd \
3 /home/piraten/md2823ho/ECC_new2_Radix_4_MM/syn/src/ECC_package_PF.vhd \
4 /home/piraten/md2823ho/ECC_new2_Radix_4_MM/syn/src/add_PF_256_bit_new1.vhd
   \
5 /home/piraten/md2823ho/ECC_new2_Radix_4_MM/syn/src/sub_PF_256_bit_new1.vhd
   \
6 /home/piraten/md2823ho/ECC_new2_Radix_4_MM/syn/src/Reg_MM_new.vhd \
7 /home/piraten/md2823ho/ECC_new2_Radix_4_MM/syn/src/MUX.vhd \
```

```

8 /home/piraten/md2823ho/ECC_new2_Radix_4_MM/syn/src/and_mult.vhd \
9 /home/piraten/md2823ho/ECC_new2_Radix_4_MM/syn/src/mult_by_4.vhd \
10 /home/piraten/md2823ho/ECC_new2_Radix_4_MM/syn/src/adder.vhd \
11 /home/piraten/md2823ho/ECC_new2_Radix_4_MM/syn/src/sub1.vhd \
12 /home/piraten/md2823ho/ECC_new2_Radix_4_MM/syn/src/sub2.vhd \
13 /home/piraten/md2823ho/ECC_new2_Radix_4_MM/syn/src/sub3.vhd \
14 /home/piraten/md2823ho/ECC_new2_Radix_4_MM/syn/src/sub4.vhd \
15 /home/piraten/md2823ho/ECC_new2_Radix_4_MM/syn/src/sub5.vhd \
16 /home/piraten/md2823ho/ECC_new2_Radix_4_MM/syn/src/sub6.vhd \
17 /home/piraten/md2823ho/ECC_new2_Radix_4_MM/syn/src/comp.vhd \
18 /home/piraten/md2823ho/ECC_new2_Radix_4_MM/syn/src/MUX_comp.vhd \
19 /home/piraten/md2823ho/ECC_new2_Radix_4_MM/syn/src/MM_Radix_4.vhd \
20 /home/piraten/md2823ho/ECC_new2_Radix_4_MM/syn/src/MSQ_Radix_4.vhd \
21 /home/piraten/md2823ho/ECC_new2_Radix_4_MM/syn/src/Reg_PDPA_new.vhd \
22 /home/piraten/md2823ho/ECC_new2_Radix_4_MM/syn/src/PF_PDPA_256_Jac_new2.vhd
    \
23 /home/piraten/md2823ho/ECC_new2_Radix_4_MM/syn/src/select_logic.vhd \
24 /home/piraten/md2823ho/ECC_new2_Radix_4_MM/syn/src/MUX_1_new.vhd \
25 /home/piraten/md2823ho/ECC_new2_Radix_4_MM/syn/src/MUX_2_new.vhd \
26 /home/piraten/md2823ho/ECC_new2_Radix_4_MM/syn/src/Reg_MUX_3.vhd \
27 /home/piraten/md2823ho/ECC_new2_Radix_4_MM/syn/src/ECC_PM_PDPA.vhd \
28 /home/piraten/md2823ho/ECC_new2_Radix_4_MM/syn/src/PISO_256.vhd \
29 /home/piraten/md2823ho/ECC_new2_Radix_4_MM/syn/src/top_new.vhd}
30 elaborate top_new -architecture arch_top_new -library DEFAULT -update
31 #####
32 #Start of Compile
33 #####
34 set_dont_touch uut/BUF_* true
35 set_dont_touch BUF1_* true
36 set_dont_touch BUF2_* true
37 set_dont_touch uut_*_buf1 true

```

```

38 set_dont_touch uut_*_buf2 true
39 set_dont_use {CORE65LPSVT/HS65_LS*}
40 set_dont_use {LVT_BUF568_RECHAR/LVT*}
41 create_clock "clk" -name "clk" -period 1.5
42 set_clock_uncertainty 0.1 clk
43 set_fix_hold clk
44 #set_propagated_clock clk
45 set_input_delay 0 -clock clk [remove_from_collection [all_input] clk]
46 set_output_delay 0 -clock clk [all_outputs]
47 set_max_area 0
48 set_max_dynamic_power 0
49 set_max_leakage_power 0
50 set_load 1.5 [all_outputs]
51 set_critical_range 0.5 top_new
52 set_cost_priority -default
53 compile -map_effort high
54 #####
55 #End of Compile
56 #####
57 report_timing > /home/piraten/md2823ho/ECC_new2_Radix_4_MM/syn/reports/
    top_new_Radix_4_MM_timing_DC.rpt
58 report_area -hierarchy > /home/piraten/md2823ho/ECC_new2_Radix_4_MM/syn/
    reports/top_new_Radix_4_MM_area_DC.rpt
59 report_power > /home/piraten/md2823ho/ECC_new2_Radix_4_MM/syn/reports/
    top_new_power_Radix_4_MM_DC.rpt
60 change_names -rules verilog -hierarchy > /dev/null
61 write -format verilog -hierarchy -output /net/cas-11/export/home/md2823ho/
    ECC_new2_Radix_4_MM/syn/netlist/top_new.v
62 write_sdf /net/cas-11/export/home/md2823ho/ECC_new2_Radix_4_MM/syn/netlist/
    top_new.sdf
63 write_sdc /net/cas-11/export/home/md2823ho/ECC_new2_Radix_4_MM/syn/netlist/

```

```

top_new.sdc
64 write -format ddc -hierarchy -output /net/cas-11/export/home/md2823ho/
    ECC_new2_Radix_4_MM/syn/netlist/top_new.ddc

```

D.2 Sample TCL scripts for ECC Processor over \mathbb{F}_{256} in Prime Time

```

1 remove_design -all
2 read_verilog /net/cas-11/export/home/md2823ho/ECC_new2_Radix_4_MM/syn/
    netlist/top_new.v
3 current_design top_new
4 create_clock "clk" -name "clk" -period 2
5 report_vcd_hierarchy /net/cas-11/export/home/md2823ho/ECC_new2_Radix_4_MM/
    syn/netlist/top_new.vcd
6 set power_analysis_mode "time_based"
7 read_vcd -strip_path TB_top_new/uut/ /net/cas-11/export/home/md2823ho/
    ECC_new2_Radix_4_MM/syn/netlist/top_new.vcd
8 update_power
9 report_power > /home/piraten/md2823ho/ECC_new2_Radix_4_MM/syn/reports/
    top_new_R_4_Power_PT.rpt
10 report_timing > /home/piraten/md2823ho/ECC_new2_Radix_4_MM/syn/reports/
    top_new_R_4_timing_PT.rpt

```

D.3 Sample TCL scripts to write a SDF file for ECC Processor over \mathbb{F}_{256}

```

1 ## Start PT shell with "pt_shell -64"
2 read_ddc /net/cas-11/export/home/md2823ho/ECC_new2_Radix_4_MM/syn/netlist/
    top_new.ddc
3 write_sdf -map /usr/local-eit/cad2/cmpstm/stm065v536/CORE65LPLVT_5.1/

```

```
behaviour/verilog/CORE65LPLVT.verilog.map /net/cas-11/export/home/
md2823ho/ECC_new2_Radix_4_MM/syn/netlist/top_new_PT.sdf
```

D.4 Sample TCL scripts for ModelSim Simulation of ECC Processor over \mathbb{F}_{256}

```
1 ##Scripts for Modelsim
2 vlog /export/home/md2823ho/ECC_new2_Radix_4_MM/syn/netlist/top_new.v
3 vcom -93 /home/piraten/md2823ho/ECC_new2_Radix_4_MM/sim/src/TB_top_new.vhd
4 vsim -L CORE65LPLVT -L CLOCK65LPLVT -L BUFX568 -L /net/cas-13/export/space/
    eit-oae/msim_libs/PADS_Jun2013 -t ps work.TB_top_new -sdfmax TB_top_new/
    uut=/export/home/md2823ho/ECC_new2_Radix_4_MM/syn/netlist/top_new_PT.sdf
5 #####
6 #ADD SIGNALS
7 #####
8 vcd file /tmp/selim/PM_PDPA_new19/vcd/top_new.vcd
9 vcd add -r TB_top_new/uut_top_new/*
10 ##commnads for vsim ##run 1 us
11 source /tmp/selim/PM_PDPA_new19/sim/scripts/top_postsynthesis.do
12 do /tmp/selim/PM_PDPA_new19/sim/scripts/top_postsynthesis.do
```


Appendix E

List of Acronyms

AES	Advanced Encryption Standard
ALU	Arithmetic Logic Unit
ANSI	American National Standards Institute
ASIC	Application-Specific Integrated Circuit
AT	Area \times Time
ATE	Area \times Time \times Energy
BF	Binary Field
CC	Clock Cycle
CHES	Cryptographic Hardware and Embedded Systems
CMOS	Complementary Metal-Oxide-Semiconductor
DB	Dual Basis
DC	Design Compiler
DES	Data Encryption Standard
3DES	Triple Data Encryption Standard
DH	Diffie-Hellman
DLP	Discrete Logarithm Problem

DPA	Differential Power Analysis
DRC	Design Rule Check
DSA	Digital Signature Algorithm
DSP	Digital Signal Processing
EC	Elliptic Curve
ECC	Elliptic Curve Cryptography
ECP	ECC Processor
ECDLP	Elliptic Curve Discrete Logarithm Problem
ECDSA	Elliptic Curve Digital Signature Algorithm
ECPA	Elliptic Curve Point Addition
ECPADD	Elliptic Curve Point Addition
ECPD	Elliptic Curve Point Doubling
ECPDBL	Elliptic Curve Point Doubling
ECPM	Elliptic Curve Point Multiplication
ECSM	Elliptic Curve Scalar Multiplication
EEA	Extended Euclidean Algorithm
FA	Fault Analysis
FFA	Finite Field Arithmetic
FIPS	Federal Information Processing Standard
FPGA	Field-Programmable Gated Array
FSM	Finite State Machine
gcd	Greatest Common Divisor
GF	Galois Field
GNB	Gaussian Normal Basis
HDL	Hardware Description Language
IEEE	Institute of Electrical and Electronic Engineers

I/O	Input/Output
ISO	International Organization for Standardization
IFP	Integer Factoring Problem
LSB	Least-Significant Bit
LSD	Least-Significant Digit
LUT	Look-Up Table
MM	Modular Multiplication
MR	Modular Reduction
MSB	Most-Significant Bit
MUX	Multiplexer
NAF	Non-Adjacent Form
NB	Normal Basis
NIST	National Institute of Standards and Technology
NSA	National Security Agency
ONB	Optimal Normal Basis
PA	Point Addition
PB	Polynomial Basis
PD	Point doubling
PDA	Personal Digital Assistant
PDPA	point doubling and point addition
PDP	Power-Delay Product
PF	Prime Field
PKC	Public-Key Cryptography
RC4	Rivest Cipher 4
RTL	Register Transfer Level
ROM	Read-Only Memory

RNS	Residue Number System
RSA	Rivest, Shamir, and Adleman PKC Algorithm
SAIF	Switching Activity Interchange Format
SCA	Side Channel Attacks
SDC	Synopsys Design Constraints
SDF	Standard Delay Format
SEC	Standards for Efficient Cryptography
SHA	Secure Hash Algorithm
SPA	Simple Power Analysis
ST	STMicroelectronics
TCL	Tool Command Language
VCD	Voltage Change Dump
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuits
VLSI	Very Large Scale Integration

Bibliography

- [1] N. Koblitz, A. Menezes, and S. Vanstone, “The state of elliptic curve cryptography,” *Des. Codes Cryptography*, vol. 19, no. 2, pp. 173–193, Mar. 2000.
- [2] D. Hankerson, A. J. Menezes, and S. Vanstone, *Guide to Elliptic Curve Cryptography*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2003.
- [3] D. Pamula, “Arithmetic operators on $GF(2^m)$ for cryptographic applications: performance - power consumption - security tradeoffs,” Theses, Université Rennes 1, Dec. 2012.
- [4] M. F. F. Khan and K. Sakamura, “Context-aware access control for clinical information systems,” in *Proc. IEEE IIT*, Mar. 2012, pp. 123–128.
- [5] M. S. Hossain, Y. Kong, E. Saeedi, and N. C. Vayalil, “High-performance elliptic curve cryptography processor over nist prime fields,” *IET Computers & Digital Techniques*, vol. 11, no. 1, pp. 33–42, Dec. 2016.
- [6] J. H. Kong, L.-M. Ang, and K. P. Seng, “A comprehensive survey of modern symmetric cryptographic solutions for resource constrained environments,” *Journal of Network and Computer Applications*, vol. 49, pp. 15 – 50, 2015.
- [7] W. Diffie and M. Hellman, “New directions in cryptography,” *IEEE Trans. Inf. Theor.*, vol. 22, no. 6, pp. 644–654, Sep. 2006.

- [8] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Commun. ACM*, vol. 21, no. 2, pp. 120–126, Feb. 1978.
- [9] T. ElGamal, *A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1985, pp. 10–18.
- [10] V. S. Miller, "Use of elliptic curves in cryptography," in *Proc. CRYPTO 1985*, 1986, pp. 417–426.
- [11] N. Koblitz, "Elliptic curve cryptosystems," *Math. Comput.*, vol. 48, pp. 203–209, 1987.
- [12] H. Cohen, G. Frey, R. Avanzi, C. Doche, T. Lange, K. Nguyen, and F. Vercauteren, *Handbook of Elliptic and Hyperelliptic Curve Cryptography, Second Edition*, 2nd ed. Chapman & Hall/CRC, 2012.
- [13] C. Shu, "Hardware architectures of elliptic curve based cryptosystems over binary fields," Ph.D. dissertation, Fairfax, VA, USA, 2007, aAI3246921.
- [14] G. Orlando, D. C. Paar, and D. J. Orr, "Efficient elliptic curve processor architectures for field programmable logic," Tech. Rep., 2002.
- [15] Z. Liu, X. Huang, Z. Hu, M. K. Khan, H. seo, and L. Zhou, "On emerging family of elliptic curves to secure internet of things: Ecc comes of age," *IEEE Trans. Dependable Secure Comput.*, vol. PP, no. 99, pp. 1–1, 2016.
- [16] R. Amin, S. H. Islam, G. P. Biswas, M. K. Khan, and N. Kumar, "An efficient and practical smart card based anonymity preserving user authentication scheme for tms using elliptic curve cryptography," *Journal of Medical Systems*, vol. 39, no. 11, p. 180, 2015.

- [17] P. Christof and P. Jan, *Understanding Cryptography*. Berlin Heidelberg: Springer-Verlag, 2010.
- [18] A. K. Lenstra and E. R. Verheul, *Selecting Cryptographic Key Sizes*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 446–465.
- [19] D. Karakoyunlu, F. K. Gurkaynak, B. Sunar, and Y. Leblebici, “Efficient and side-channel-aware implementations of elliptic curve cryptosystems over prime fields,” *IET Information Security*, vol. 4, no. 1, pp. 30–43, March 2010.
- [20] Y. R. Hitchcock, “Elliptic curve cryptography for lightweight applications,” Ph.D. dissertation, Queensland University of Technology, 2003.
- [21] Z. Liu, D. Liu, and X. Zou, “An efficient and flexible hardware implementation of the dual-field elliptic curve cryptographic processor,” *IEEE Trans. Ind. Electron.*, vol. PP, no. 99, pp. 1–1, 2016.
- [22] R. G. Kammer, “NIST- National Institute of Standards and Technology, Digital Signature Standard, FIPS Publication 186-2,” 2000.
- [23] “IEEE standard specifications for public-key cryptography,” *IEEE Std 1363-2000*, pp. 1–228, Aug. 2000.
- [24] “X 9.62 public key cryptography for the financial services industry: Elliptic curve digital signature algorithm (ECDSA),” american national standards institute,” 1999.
- [25] M. E. Kaihara, “Studies on modular arithmetic hardware algorithms for public-key cryptography,” Ph.D. dissertation, 2006.
- [26] B. Schneier, *Applied Cryptography (2nd Ed.): Protocols, Algorithms, and Source Code in C*. New York, NY, USA: John Wiley & Sons, Inc., 1995.

- [27] A. J. Menezes, S. A. Vanstone, and P. C. V. Oorschot, *Handbook of Applied Cryptography*, 1st ed. Boca Raton, FL, USA: CRC Press, Inc., 1996.
- [28] “<https://msdn.microsoft.com/en-us/library/ff650720.aspx>,” 2005.
- [29] “<https://en.wikipedia.org/wiki/symmetric-key-algorithm>,” 2016.
- [30] X. Lai and J. L. Massey, “A proposal for a new block encryption standard,” in *Proc. EUROCRYPT '90*. New York, NY, USA: Springer-Verlag New York, Inc., 1991, pp. 389–404.
- [31] R. C. Merkle, “Secure communications over insecure channels,” *Commun. ACM*, vol. 21, no. 4, pp. 294–299, 1978.
- [32] S. Pohlig and M. Hellman, “An improved algorithm for computing logarithms over and its cryptographic significance (corresp.),” *IEEE Trans. Inf. Theor.*, vol. 24, no. 1, pp. 106–110, Sep. 2006.
- [33] D. Hankerson, J. L. Hernandez, and A. Menezes, “Software implementation of elliptic curve cryptography over binary fields,” in *Proc. CHES*. London, UK: Springer-Verlag, 2000, pp. 1–24.
- [34] D. V. Bailey and C. Paar, “Efficient arithmetic in finite field extensions with application in elliptic curve cryptography,” *Journal of Cryptology*, vol. 14, p. 2001, 2000.
- [35] G. Sutter, J. Deschamps, and J. Imana, “Efficient elliptic curve point multiplication using digit-serial binary field operations,” *IEEE Trans. Ind. Electron.*, vol. 60, no. 1, pp. 217–225, Jan. 2013.

- [36] R. Azarderakhsh, “High speed and low-complexity hardware architectures for elliptic curve-based crypto-processors,” Ph.D. dissertation, 2011. [Online]. Available: <http://ir.lib.uwo.ca/etd/308>
- [37] H. Y. Kim, J. Y. Park, J. H. Cheon, J. H. Park, J. H. Kim, and S. G. Hahn, *Fast Elliptic Curve Point Counting Using Gaussian Normal Basis*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 292–307.
- [38] R. Lercier and D. Lubicz, *Counting Points on Elliptic Curves over Finite Fields of Small Characteristic in Quasi Quadratic Time*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 360–373.
- [39] Ö. Eğecioğlu and Ç. K. Koç, *Reducing the Complexity of Normal Basis Multiplication*. Cham: Springer International Publishing, 2015, pp. 61–80.
- [40] G. Orlando and C. Paar, “A high-performance reconfigurable elliptic curve processor for $GF(2^m)$,” Springer-Verlag, 2000, pp. 41–56.
- [41] J. Wolkerstorfer, “Dual-field arithmetic unit for $GF(p)$ and $GF(2^m)$,” in *CHES, Lecture Notes in Computer Science*, vol. 2523, Springer, 2002, pp. 500–514.
- [42] C. K. Koc and T. Acar, “Montgomery multiplication in $gf(2^k)$,” *Designs, Codes and Cryptography*, vol. 14, no. 1, pp. 57–69, 1998.
- [43] J. S. Pan, C. Y. Lee, and P. K. Meher, “Low-latency digit-serial and digit-parallel systolic multipliers for large binary extension fields,” *IEEE Trans. Circuits Syst. I*, vol. 60, no. 12, pp. 3195–3204, Dec. 2013.
- [44] J.-W. Lee, S.-C. Chung, H.-C. Chang, and C.-Y. Lee, “Efficient power-analysis-resistant dual-field elliptic curve cryptographic processor using heterogeneous dual-

- processing-element architecture,” *IEEE Trans. VLSI Syst.*, vol. 22, no. 1, pp. 49–61, Jan. 2014.
- [45] C. Rebeiro, S. S. Roy, and D. Mukhopadhyay, “Pushing the limits of high-speed $\text{GF}(2^m)$ elliptic curve scalar multiplication on FPGAs,” in *Proc. CHES*, 2012, pp. 494–511.
- [46] S. Kumar, T. Wollinger, and C. Paar, “Optimum digit serial $\text{GF}(2^m)$ multipliers for curve-based cryptography,” *IEEE Trans. Comput.*, vol. 55, no. 10, pp. 1306–1311, Oct. 2006.
- [47] M. N. Ismail, “Towards efficient hardware implementation of elliptic and hyperelliptic curve cryptography,” 2012.
- [48] M. Rosing, *Implementing Elliptic Curve Cryptography*. Greenwich, CT, USA: Manning Publications Co., 1999.
- [49] T. Itoh and S. Tsujii, “A fast algorithm for computing multiplicative inverses in $\text{GF}(2^m)$ using normal bases,” *Information and Computation*, vol. 78, no. 3, pp. 171 – 177, 1988.
- [50] E. Savas, M. Naseer, A. A. A. Gutub, and C. K. Koc, “Efficient unified montgomery inversion with multibit shifting,” *IEE Proceedings - Computers and Digital Techniques*, vol. 152, no. 4, pp. 489–498, Jul. 2005.
- [51] H. Brunner, A. Curiger, and M. Hofstetter, “On computing multiplicative inverses in $\text{GF}(2^m)$,” *IEEE Trans. Comput.*, vol. 42, no. 8, pp. 1010–1015, Aug. 1993.
- [52] J.-H. Guo and C.-L. Wang, “Systolic array implementation of euclid’s algorithm for inversion and division in $\text{GF}(2^m)$,” *IEEE Trans. Comput.*, vol. 47, no. 10, pp. 1161–1167, Oct. 1998.

- [53] M. S. Hossain, E. Saeedi, and Y. Kong, "High-speed, area-efficient, FPGA-based elliptic curve cryptographic processor over NIST binary fields," in *IEEE International Conference on Data Science and Data Intensive Systems*, Dec. 2015, pp. 175–181.
- [54] "SEC 2: Recommended elliptic curve domain parameters, standards for efficient cryptography, Certicom Research," 2000.
- [55] K. Javeed and X. Wang, "Speed and area optimized parallel higher-radix modular multipliers." *IACR Cryptology ePrint Archive*, vol. 2016, p. 53, 2016.
- [56] L. Rahimzadeh, M. Eshghi, and S. Timarchi, "Radix-4 implementation of redundant interleaved modular multiplication on FPGA," in *ICEE*, May 2014, pp. 523–526.
- [57] K. Javeed and X. Wang, "Radix-4 and radix-8 booth encoded interleaved modular multipliers over general \mathbb{F}_p ," in *Int. Conf. FPL*, Sept. 2014, pp. 1–6.
- [58] S. Ghosh, D. Mukhopadhyay, and D. Roychowdhury, "Secure dual-core cryptoprocessor for pairings over barreto-naehrig curves on FPGA platform," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 21, no. 3, pp. 434–442, Mar. 2013.
- [59] S. H. Wang, W. C. Lin, J. H. Ye, and M. D. Shieh, "Fast scalable radix-4 montgomery modular multiplier," in *2012 IEEE ISCAS*, May 2012, pp. 3049–3052.
- [60] V. Bunimov and M. Schimmler, "Area and time efficient modular multiplication of large integers," in *Proc. IEEE Int. Conf. ASAP*, Jun. 2003, pp. 400–409.
- [61] D. N. Amanor, C. Paar, J. Pelzl, V. Bunimov, and M. Schimmler, "Efficient hardware architectures for modular multiplication on fpgas," in *International Conference on Field Programmable Logic and Applications, 2005.*, Aug. 2005, pp. 539–542.

- [62] A. F. Tenca and L. A. Tawalbeh, "An efficient and scalable radix-4 modular multiplier design using recoding techniques," in *The Thrity-Seventh Asilomar Conference on Signals, Systems Computers, 2003*, vol. 2, Nov. 2003, pp. 1445–1450.
- [63] J.-H. Hong and C.-W. Wu, "Radix-4 modular multiplication and exponentiation algorithms for the RSA public-key cryptosystem," in *Proceedings 2000. Design Automation Conference. (IEEE Cat. No.00CH37106)*, Jun. 2000, pp. 565–570.
- [64] N. Takagi, "A radix-4 modular multiplication hardware algorithm for modular exponentiation," *IEEE Trans. Comput.*, vol. 41, no. 8, pp. 949–956, Aug. 1992.
- [65] M. Knezevic, F. Vercauteren, and I. Verbauwhede, "Faster interleaved modular multiplication based on barrett and montgomery reduction methods," *IEEE Trans. Comput.*, vol. 59, no. 12, pp. 1715–1721, Dec. 2010.
- [66] P. L. Montgomery, "Modular multiplication without trial division," *Mathematics of Computation*, vol. 44, no. 170, pp. 519–521, 1985.
- [67] P. Barrett, *Implementing the Rivest Shamir and Adleman Public Key Encryption Algorithm on a Standard Digital Signal Processor*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1987, pp. 311–323.
- [68] A. Bosselaers, R. Govaerts, and J. Vandewalle, *Comparison of three modular reduction functions*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1994, pp. 175–186.
- [69] M. S. Hossain and Y. Kong, "FPGA-based efficient modular multiplication for elliptic curve cryptography," in *International Telecommunication Networks and Applications Conference (ITNAC)*, Nov. 2015, pp. 191–195.
- [70] L. A. Tawalbeh, A. F. Tenca, and C. K. Koc, "A radix-4 scalable design," *IEEE Potentials*, vol. 24, no. 2, pp. 16–18, Apr. 2005.

- [71] B. S. Kaliski, "The montgomery inverse and its applications," *IEEE Trans. Comput.*, vol. 44, no. 8, pp. 1064–1065, Aug. 1995.
- [72] P. Choi, J. T. Kong, and D. K. Kim, "Analysis of hardware modular inversion modules for elliptic curve cryptography," in *2015 International SoC Design Conference (ISOCC)*, Nov. 2015, pp. 313–314.
- [73] J.-W. Lee, S.-C. Chung, H.-C. Chang, and C.-Y. Lee, "Efficient power-analysis-resistant dual-field elliptic curve cryptographic processor using heterogeneous dual-processing-element architecture," *IEEE Trans. VLSI Syst.*, vol. 22, no. 1, pp. 49–61, Jan. 2014.
- [74] E. Murat, S. Kardas, and E. Savas, "Scalable and efficient FPGA implementation of montgomery inversion," in *2011 Workshop on Lightweight Security Privacy: Devices, Protocols, and Applications*, Mar. 2011, pp. 61–68.
- [75] S. Ghosh, D. Mukhopadhyay, and D. Roychowdhury, "Petrel: Power and timing attack resistant elliptic curve scalar multiplier based on programmable $GF(p)$ arithmetic unit," *IEEE Trans. Circuits Syst. I, Reg. Papers.*, vol. 58, no. 8, pp. 1798–1812, Aug. 2011.
- [76] J. Vliegen, N. Mentens, J. Genoe, A. Braeken, S. Kubera, A. Touhafi, and I. Verbauwhede, "A compact fpga-based architecture for elliptic curve cryptography over prime fields," in *2010 21st IEEE International Conference on Application-specific Systems Architectures and Processors (ASAP)*, July 2010, pp. 313–316.
- [77] G. Chen, J. Zhu, M. Liu, and W. Zheng, "An improved dual field modular inversion algorithm and vlsi implementation," in *2009 First International Conference on Information Science and Engineering*, Dec. 2009, pp. 1651–1654.

- [78] S. Ma, Y. Hao, Z. Pan, and H. Chen, "Fast implementation for modular inversion and scalar multiplication in the elliptic curve cryptography," in *2008 Second International Symposium on Intelligent Information Technology Application*, vol. 2, Dec. 2008, pp. 488–492.
- [79] C. McIvor, M. McLoone, and J. McCanny, "Hardware elliptic curve cryptographic processor over $gf(p)$," *IEEE Trans. Circuits Syst. I. Reg. Papers.*, vol. 53, no. 9, pp. 1946–1957, Sept. 2006.
- [80] A. Daly, W. Marnane, T. Kerins, and E. Popovici, "An FPGA implementation of a $GF(p)$ ALU for encryption processors," *Microprocessors and Microsystems*, vol. 28, no. 5–6, pp. 253–260, 2004, special Issue on FPGAs: Applications and Designs.
- [81] L. A. Tawalbeh, A. F. Tenca, S. Park, and C. K. Koc, "A dual-field modular division algorithm and architecture for application specific hardware," in *Conference Record of the Thirty-Eighth Asilomar Conference on Signals, Systems and Computers, 2004.*, vol. 1, Nov. 2004, pp. 483–487 Vol.1.
- [82] T. Zhou, X. Wu, G. Bai, and H. Chen, "New algorithm and fast vlsi implementation for modular inversion in galois field $GF(p)$," in *IEEE 2002 International Conference on Communications, Circuits and Systems and West Sino Expositions*, vol. 2, Jun. 2002, pp. 1491–1495 vol.2.
- [83] A. A. A. Gutub, A. F. Tenca, and C. K. Koc, "Scalable VLSI architecture for $GF(p)$ montgomery modular inverse computation," in *Proceedings IEEE Computer Society Annual Symposium on VLSI. New Paradigms for VLSI Systems Design. ISVLSI 2002*, Apr. 2002, pp. 46–51.

- [84] M. S. Hossain and Y. Kong, “High-performance FPGA implementation of modular inversion over \mathbb{F}_{256} for elliptic curve cryptography,” in *IEEE International Conference on Data Science and Data Intensive Systems*, Dec. 2015, pp. 169–174.
- [85] I. F. Blake, G. Seroussi, and N. P. Smart, *Elliptic Curves in Cryptography*. New York, NY, USA: Cambridge University Press, 1999.
- [86] H. Shen, J. Shen, M. K. Khan, and J.-H. Lee, “Efficient rfid authentication using elliptic curve cryptography for the internet of things,” *Wireless Personal Communications*, pp. 1–14, 2016.
- [87] G. Orlando and C. Paar, “A high-performance reconfigurable elliptic curve processor for $\text{GF}(2^m)$,” in *Proc. CHES*, 2000, pp. 41–56.
- [88] D. Schinianakis, A. Fournaris, H. Michail, A. Kakarountas, and T. Stouraitis, “An RNS implementation of an F_p elliptic curve point multiplier,” *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 56, no. 6, pp. 1202–1213, June 2009.
- [89] P. Longa and A. Miri, “Fast and flexible elliptic curve point arithmetic over prime fields,” *IEEE Trans. Comput.*, vol. 57, no. 3, pp. 289–302, 2008.
- [90] S. Ghosh, M. Alam, D. R. Chowdhury, and I. S. Gupta, “Parallel crypto-devices for $\text{GF}(p)$ elliptic curve multiplication resistant against side channel attacks,” *Comput. Electr. Eng.*, vol. 35, no. 2, pp. 329–338, Mar. 2009.
- [91] E. Saeedi, M. S. Hossain, and Y. Kong, “Multi-class svms analysis of side-channel information of elliptic curve cryptosystem,” in *2015 International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS)*, Jul. 2015, pp. 1–6.

- [92] —, “Side channel analysis of an elliptic curve crypto-system based on multi-class classification,” in *2015 6th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*, Jul. 2015, pp. 1–7.
- [93] E. Saeedi, Y. Kong, and M. S. Hossain, “Side-channel attacks and learning-vector quantization,” *Frontiers of Information Technology & Electronic Engineering*, vol. -1, no. -1, 1998.
- [94] S. A. Chaudhry, M. T. Khan, M. K. Khan, and T. Shon, “A multiserver biometric authentication scheme for tmis using elliptic curve cryptography,” *Journal of Medical Systems*, vol. 40, no. 11, p. 230, 2016.
- [95] K. A. Alezabi, F. Hashim, S. J. Hashim, B. M. Ali, and A. Jamalipour, “On the authentication and re-authentication protocols in lte-wlan interworking architecture,” *Transactions on Emerging Telecommunications Technologies*, pp. n/a–n/a, 2016, ett.3031.
- [96] M. F. F. Khan, Y. Takeshi, I. So, M. Bessho, and K. Sakamura, “A secure and flexible electronic-ticket system,” in *Proc. IEEE COMPSAC*, vol. 1, Jul. 2009, pp. 421–426.
- [97] D. Johnson, A. Menezes, and S. Vanstone, “The elliptic curve digital signature algorithm (ECDSA),” *Int. J. Inf. Secur.*, vol. 1, no. 1, pp. 36–63, 2001.
- [98] D. Boneh and M. K. Franklin, “Identity-based encryption from the weil pairing,” in *Proc. CRYPTO '01*. London, UK: Springer-Verlag, 2001, pp. 213–229.
- [99] D. Boneh, B. Lynn, and H. Shacham, “Short signatures from the weil pairing,” *Journal of Cryptology*, vol. 17, no. 4, pp. 297–319, 2004.

- [100] D. F. Aranha, J.-L. Beuchat, J. Detrey, and N. Estibals, “Optimal eta pairing on supersingular genus-2 binary hyperelliptic curves,” in *Proc. CT-RSA '12*. Berlin, Heidelberg: Springer-Verlag, 2012, pp. 98–115.
- [101] J. luc Beuchat, J. Detrey, N. Estibals, E. Okamoto, and F. Rodríguez-henríquez, “Fast architectures for the η t pairing over small-characteristic supersingular elliptic curves,” *IEEE TRANS. COMPUT*, vol. 60, no. 2, pp. 266–281, 2011.
- [102] R. Lidl and H. Niederreiter, *Introduction to Finite Fields and Their Applications*. New York, NY, USA: Cambridge University Press, 1986.
- [103] I. S. Reed and G. Solomon, “Polynomial codes over certain finite fields,” *Journal of the Society for Industrial and Applied Mathematics*, vol. 8, no. 2, pp. 300–304, 1960.
- [104] R. Salarifard, S. Bayat-Sarmadi, and M. Farmani, “High-throughput low-complexity unified multipliers over $\text{GF}(2^m)$ in dual and triangular bases,” *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 63, no. 11, pp. 1944–1953, Nov. 2016.
- [105] J. Xie, P. K. Meher, and Z. H. Mao, “Low-latency high-throughput systolic multipliers over $\text{GF}(2^m)$ for nist recommended pentanomials,” *IEEE Trans. Circuits Syst. I*, vol. 62, no. 3, pp. 881–890, Mar. 2015.
- [106] M. Khairallah and M. Ghoneima, “New polynomial basis versatile multiplier over $\text{GF}(2^m)$ for low-power on-chip crypto-systems,” in *Proc. IEEE ISCAS*, May 2015, pp. 1438–1441.
- [107] R. Azarderakhsh, K. Järvinen, and V. Dimitrov, “Fast inversion in $\text{GF}(2^m)$ with normal basis using hybrid-double multipliers,” *IEEE Trans. Comput.*, vol. 63, no. 4, pp. 1041–1047, Apr. 2014.

- [108] H. Ho, “Design and implementation of a polynomial basis multiplier architecture over $\text{GF}(2^m)$,” *Journal of Signal Processing Systems*, vol. 75, no. 3, pp. 203–208, 2014.
- [109] R. K. Kodali and L. Boppana, “FPGA implementation of energy efficient multiplication over $\text{GF}(2^m)$ for ECC,” in *Proc. IEEE ICACCI*, Sep. 2014, pp. 1815–1821.
- [110] K. C. Loi and S.-B. Ko, “High performance scalable elliptic curve cryptosystem processor for koblitz curves,” *Microprocessors and Microsystems*, vol. 37, no. 4-5, pp. 394–406, 2013.
- [111] K. C. C. Loi and S. B. Ko, “High performance scalable elliptic curve cryptosystem processor in $\text{GF}(2^m)$,” in *IEEE ISCAS*, May 2013, pp. 2585–2588.
- [112] A. Zakerolhosseini and M. Nikooghadam, “Low-power and high-speed design of a versatile bit-serial multiplier in finite fields $\text{GF}(2^m)$,” *Integration, the VLSI Journal*, vol. 46, no. 2, pp. 211 – 217, 2013.
- [113] S. H. Namin, H. Wu, and M. Ahmadi, “Power efficiency of digit level polynomial basis finite field multipliers in $\text{GF}(2^{283})$,” in *Proc. IEEE ICECS*, Dec. 2012, pp. 897–900.
- [114] K. Kobayashi and N. Takagi, “A combined circuit for multiplication and inversion in $\text{GF}(2^m)$,” *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 55, no. 11, pp. 1144–1148, Nov. 2008.
- [115] S. Antao, R. Chaves, and L. Sousa, “Efficient FPGA elliptic curve cryptographic processor over $\text{GF}(2^m)$,” in *ICECE Technology, 2008. FPT 2008. International Conference on*, Dec. 2008, pp. 357–360.

- [116] M. Morales-Sandoval, C. Feregrino-Urbe, R. Cumplido, and I. Algreto-Badillo, "A reconfigurable $GF(2^M)$ elliptic curve cryptographic coprocessor," in *Proc. SPL*, Apr. 2011, pp. 209–214.
- [117] M. N. Hassan and M. Benaissa, "A scalable hardware/software co-design for elliptic curve cryptography on picoblaze microcontroller," in *Proc. IEEE ISCAS*, May 2010, pp. 2111–2114.
- [118] M. Machhout, Z. Guitouni, K. Torki, L. Khriji, and R. Tourki, "Coupled FPGA/ASIC implementation of elliptic curve crypto-processor," *International Journal of Network Security & its Applications (IJNSA)*, vol. 2, no. 2, pp. 100–112, Apr. 2010.
- [119] M. Benaissa and W. M. Lim, "Design of flexible $GF(2^m)$ elliptic curve cryptography processors," *IEEE Trans. VLSI Syst.*, vol. 14, no. 6, pp. 659–662, Jun. 2006.
- [120] C. Shu, K. Gaj, and T. El-Ghazawi, "Low latency elliptic curve cryptography accelerators for nist curves over binary fields," in *Proc. IEEE FPT*, Dec. 2005, pp. 309–310.
- [121] N. Gura, S. C. Shantz, H. Eberle, S. Gupta, V. Gupta, D. Finchelstein, E. Goupy, and D. Stebila, "An end-to-end systems approach to elliptic curve cryptography," in *Proc. CHES*. London, UK: Springer-Verlag, 2003, pp. 349–365.
- [122] P. K. Meher, "On efficient implementation of accumulation in finite field over $GF(2^m)$ and its applications," *IEEE Trans. VLSI Syst.*, vol. 17, no. 4, pp. 541–550, Apr. 2009.
- [123] C. S. Yeh, I. S. Reed, and T. K. Truong, "Systolic multipliers for finite fields $GF(2^m)$," *IEEE Trans. Comput.*, vol. C-33, no. 4, pp. 357–360, Apr. 1984.

- [124] J. Hu, W. Guo, J. Wei, and R. C. C. Cheung, "Fast and generic inversion architectures over $\text{GF}(2^m)$ using modified Itoh-Tsujii Algorithms," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 62, no. 4, pp. 367–371, Apr. 2015.
- [125] W. Chelton and M. Benaissa, "Fast elliptic curve cryptography on FPGA," *IEEE Trans. VLSI Syst.*, vol. 16, no. 2, pp. 198–205, Feb. 2008.
- [126] M. B. I. Reaz, J. Jalil, H. Husian, and F. H. Hasim, "FPGA implementation of elliptic curve cryptography engine for personal communication systems," *WSEAS Tran. on Circuits and Systems*, vol. 11, no. 3, pp. 82–91, Mar. 2012.
- [127] M. N. Hassan and M. Benaissa, "Efficient time-area scalable ECC processor using μ -coding technique," in *Third International Workshop, WAIFI, Arithmetic of Finite Fields, LNCS 6087*, pp. 250–268, Jun. 2010., Jun. 2010, pp. 250–268.
- [128] N. Ghanmy, L. C. Fourati, and L. Kamoun, "Elliptic curve cryptography for WSN and SPA attacks method for energy evaluation," *Journal of Networks*, vol. 9, no. 11, pp. 2943–2950, Nov. 2014.
- [129] M.-D. Shieh, J.-H. Chen, W.-C. Lin, and C.-M. Wu, "An efficient multiplier/divider design for elliptic curve cryptosystem over $\text{GF}(2^m)$," *Journal of Information Science and Engineering* 25, pp. 1555–1573, 2009, pp. 1555–1553, Apr. 2009.
- [130] N. Smyth, M. McLoone, and J. V. McCanny, "An adaptable and scalable asymmetric cryptographic processor." in *ASAP*. IEEE Computer Society, 2006, pp. 341–346.
- [131] J. Park and J.-T. Hwang, "FPGA and ASIC implementation of ECC processor for security on medical embedded system," in *Proc. ICITA*, ser. ICITA '05. Washington, DC, USA: IEEE Computer Society, 2005, pp. 547–551.

- [132] Y. Kong and B. Phillips, "Fast scaling in the residue number system," *IEEE Trans. VLSI Syst.*, vol. 17, no. 3, pp. 443–447, Mar. 2009.
- [133] B. Phillips, Y. Kong, and Z. Lim, "Highly parallel modular multiplication in the residue number system using sum of residues reduction," *Applicable Algebra in Engineering, Communication and Computing*, vol. 21, no. 3, pp. 249–255, 2010.
- [134] Y.-B. Wang, X.-J. Dong, and Z.-G. Tian, "FPGA based design of elliptic curve cryptography coprocessor," in *Third International Conference on Natural Computation, ICNC 2007*, Aug. 2007, pp. 185–189.
- [135] X. Zeng, C. Chen, and Q. Zhang, "A reconfigurable public-key cryptography coprocessor," in *Proceedings of 2004 IEEE Asia-Pacific Conference on Advanced System Integrated Circuits 2004.*, Aug. 2004, pp. 172–175.
- [136] N. Nguyen, K. Gaj, D. Caliga, and T. El-Ghazawi, "Implementation of elliptic curve cryptosystems on a reconfigurable computer," in *Proceedings. 2003 IEEE International Conference on Field-Programmable Technology (FPT), 2003.*, Dec. 2003, pp. 60–67.
- [137] Y. Kong, S. Asif, and M. Khan, "Modular multiplication using the core function in the residue number system," *Applicable Algebra in Engineering, Communication and Computing*, pp. 1–16, Jan. 2015.
- [138] K. C. C. Loi, S. An, and S. B. Ko, "FPGA implementation of low latency scalable elliptic curve cryptosystem processor in $GF(2^m)$," in *Proc. IEEE ISCAS*, Jun. 2014, pp. 822–825.
- [139] S. Liu, L. Ju, X. Cai, Z. Jia, and Z. Zhang, "High performance FPGA implementation of elliptic curve cryptography over binary fields," in *Proc. IEEE TrustCom*, Sept. 2014, pp. 148–155.

- [140] R. Azarderakhsh and A. Reyhani-Masoleh, "High-performance implementation of point multiplication on koblitz curves," *IEEE Trans. Circuits Syst. II*, vol. 60, no. 1, pp. 41–45, Jan. 2013.
- [141] H. Mahdizadeh and M. Masoumi, "Novel architecture for efficient FPGA implementation of elliptic curve cryptographic processor over $\text{GF}(2^{163})$," *IEEE Trans. VLSI Syst.*, vol. 21, no. 12, pp. 2330–2333, Dec. 2013.
- [142] R. Azarderakhsh and A. Reyhani-Masoleh, "Efficient FPGA implementations of point multiplication on binary edwards and generalized hessian curves using gaussian normal basis," *IEEE Trans. VLSI Syst.*, vol. 20, no. 8, pp. 1453–1466, Aug. 2012.
- [143] S. S. Roy, C. Rebeiro, and D. Mukhopadhyay, "A parallel architecture for koblitz curve scalar multiplications on FPGA platforms," in *Proc. Euromicro DSD*, Sep. 2012, pp. 553–559.
- [144] Y. Zhang, D. Chen, Y. Choi, L. Chen, and S.-B. Ko, "A high performance ECC hardware implementation with instruction-level parallelism over $\text{GF}(2^{163})$," *Microprocess. Microsyst.*, vol. 34, no. 6, pp. 228–236, Oct. 2010.
- [145] D. Schinianakis, A. Kakarountas, T. Stouraitis, and A. Skavantzios, "Elliptic curve point multiplication in $\text{GF}(2^n)$ using polynomial residue arithmetic," in *IEEE ICECS*, Dec. 2009, pp. 980–983.
- [146] B. Ansari and M. A. Hasan, "High-performance architecture of elliptic curve scalar multiplication," *IEEE Trans. Comput.*, vol. 57, no. 11, pp. 1443–1453, Nov. 2008.
- [147] C. H. Kim, S. Kwon, and C. P. Hong, "FPGA implementation of high performance elliptic curve cryptographic processor over $\text{GF}(2^{163})$," *J. Syst. Archit.*, vol. 54, no. 10, pp. 893–900, 2008.

- [148] M. S. Hossain, E. Saeedi, and Y. Kong, “High-performance FPGA implementation of elliptic curve cryptography processor over binary field $GF(2^{163})$,” in *International Conference on Information Systems Security and Privacy (ICISSP)*, 2016, pp. 415–422.
- [149] J.-H. Chen, M.-D. Shieh, and W.-C. Lin, “A high-performance unified-field reconfigurable cryptographic processor,” *IEEE Trans. VLSI Syst.*, vol. 18, no. 8, pp. 1145–1158, Aug. 2010.
- [150] J. H. Hong and W. C. Wu, “The design of high performance elliptic curve cryptographic,” in *IEEE Int. Midwest Symp. Circuits Syst.*, Aug. 2009, pp. 527–530.
- [151] Y. K. Lee, K. Sakiyama, L. Batina, and I. Verbauwhede, “Elliptic-curve-based security processor for RFID,” *IEEE Trans. Comput.*, vol. 57, no. 11, pp. 1514–1527, Nov. 2008.
- [152] K. Sakiyama, L. Batina, B. Preneel, and I. Verbauwhede, “Multicore curve-based cryptoprocessor with reconfigurable modular arithmetic logic units over $GF(2^n)$,” *IEEE Trans. Comput.*, vol. 56, no. 9, pp. 1269–1282, Sept. 2007.
- [153] E. S. Kumar and C. Paar, “Are standards compliant elliptic curve cryptosystems feasible on RFID,” in *Proc. of RFIDSec’06*, 2006.
- [154] A. G. Reddy, A. K. Das, E. J. Yoon, and K. Y. Yoo, “A secure anonymous authentication protocol for mobile services on elliptic curve cryptography,” *IEEE Access*, vol. 4, pp. 4394–4407, 2016.
- [155] L. Zhang, S. Tang, and H. Luo, “Elliptic curve cryptography-based authentication with identity protection for smart grids,” *PLOS ONE*, vol. 11, no. 3, pp. 1–15, 03 2016.

- [156] J. Nam, K.-K. R. Choo, S. Han, M. Kim, J. Paik, and D. Won, "Efficient and anonymous two-factor user authentication in wireless sensor networks: Achieving user anonymity with lightweight sensor computation," *PLOS ONE*, vol. 10, no. 4, pp. 1–21, 04 2015.
- [157] Y.-M. Lai, P.-J. Cheng, C.-C. Lee, and C.-Y. Ku, "A new ticket-based authentication mechanism for fast handover in mesh network," *PLOS ONE*, vol. 11, no. 5, pp. 1–18, 05 2016.
- [158] A. G. Reddy, A. K. Das, V. Odelu, and K.-Y. Yoo, "An enhanced biometric based authentication with key-agreement protocol for multi-server architecture based on elliptic curve cryptography," *PLOS ONE*, vol. 11, no. 5, pp. 1–28, 05 2016.
- [159] M. Wang, G. Dai, K.-K. R. Choo, P. P. Jayaraman, and R. Ranjan, "Constructing pairing-friendly elliptic curves under embedding degree 1 for securing critical infrastructures," *PLOS ONE*, vol. 11, no. 8, pp. 1–13, 08 2016.
- [160] Y. Lu, L. Li, H. Peng, and Y. Yang, "An anonymous two-factor authenticated key agreement scheme for session initiation protocol using elliptic curve cryptography," *Multimedia Tools and Applications*, pp. 1–15, 2015.
- [161] Z. U. A. Khan and M. Benaissa, "Low area ecc implementation on FPGA," in *Proc. IEEE ICECS*, Dec. 2013, pp. 581–584.
- [162] F. A. Urbano-Molano, V. Trujillo-Olaya, and J. Velasco-Medina, "Design of an elliptic curve cryptoprocessor using optimal normal basis over $\text{GF}(2^{233})$," in *Proc. IEEE LASCAS*, Feb. 2013, pp. 1–4.
- [163] M. Imran, M. Kashif, and M. Rashid, "Hardware design and implementation of scalar multiplication in elliptic curve cryptography (ECC) over $\text{GF}(2^{163})$ on FPGA," in *Proc. ICICT*, Dec. 2015, pp. 1–4.

- [164] S. Antao, R. Chaves, and L. Sousa, "Compact and flexible microcoded elliptic curve processor for reconfigurable devices," in *Proc. IEEE Symp. FCCM*, Apr. 2009, pp. 193–200.
- [165] J. P. Deschamps and G. Sutter, "Elliptic-curve point-multiplication over $\text{GF}(2^{163})$," in *Proc. IEEE SPL*, Mar. 2008, pp. 25–30.
- [166] G. Zied, M. Mohsen, and T. Rached, "On the hardware design of elliptic curve public key cryptosystems using programmable cellular automata," in *Proc. Intern. Conf. SCS*, Nov. 2008, pp. 1–6.
- [167] K. Sakiyama, E. D. Mulder, B. Preneel, and I. Verbauwhede, "A parallel processing hardware architecture for elliptic curve cryptosystems," in *Proc. IEEE ICASSP*, vol. 3, May 2006, pp. III–III.
- [168] J. Y. Lai and C. T. Huang, "Energy-adaptive dual-field processor for high-performance elliptic curve cryptographic applications," *IEEE Trans. VLSI Syst.*, vol. 19, no. 8, pp. 1512–1517, Aug. 2011.
- [169] —, "A highly efficient cipher processor for dual-field elliptic curve cryptography," *IEEE Trans. Circuits Syst. II*, vol. 56, no. 5, pp. 394–398, May 2009.
- [170] M. Kaihara and N. Takagi, "Bipartite modular multiplication method," *IEEE Trans. Comput.*, vol. 57, no. 2, pp. 157–164, 2008.
- [171] Y. Kong and B. Phillips, "Revisiting sum of residues modular multiplication," *JECE*, vol. 2010, pp. 43:1–43:9, Jan. 2010.
- [172] D. Cheng-hua, L. Yi, and C. Yong-tao, "A 3-stage pipelined large integer modular arithmetic unit for ecc," in *International Symposium on Information Engineering and Electronic Commerce (IEEC '09)*, May 2009, pp. 519–523.

- [173] J. Fan, K. Sakiyama, and I. Verbauwhede, "Montgomery modular multiplication algorithm on multi-core systems," in *IEEE Workshop on Signal Processing Systems*, Oct. 2007, pp. 261–266.
- [174] A. Daly, W. Marnane, T. Kerins, and E. Popovici, "Division in $GF(p)$ for application in elliptic curve cryptosystems on field programmable logic," in *New Algorithms, Architectures and Applications for Reconfigurable Computing*, P. Lysaght and W. Rosenstiel, Eds. Springer US, 2005, pp. 219–229.
- [175] K. C. C. Loi and S. B. Ko, "Scalable elliptic curve cryptosystem fpga processor for nist prime curves," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 23, no. 11, pp. 2753–2756, Nov. 2015.
- [176] H. Marzouqi, M. Al-Qutayri, and K. Salah, "An FPGA implementation of NIST 256 prime field ECC processor," in *Proc. IEEE ICECS*, Dec. 2013, pp. 493–496.
- [177] K. Ananyi, H. Alrimeih, and D. Rakhmatov, "Flexible hardware processor for elliptic curve cryptography over nist prime fields," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 17, no. 8, pp. 1099–1112, Aug. 2009.
- [178] H. Ahmadi and A. Afzali-Kusha, "Low-power low-energy prime-field ecc processor based on montgomery modular inverse algorithm," in *Proc. 12th Euromicro DSD*, Aug. 2009, pp. 817–822.
- [179] J. Fan, K. Sakiyama, and I. Verbauwhede, "Elliptic curve cryptography on embedded multicore systems," *Design Automation for Embedded Systems*, vol. 12, no. 3, pp. 231–242, 2008.
- [180] N. Mentens, K. Sakiyama, L. Batina, B. Preneel, and I. Verbauwhede, "A side-channel attack resistant programmable pkc coprocessor for embedded applications," in *Proc. IC-SAMOS*, July 2007, pp. 194–200.

- [181] K. Sakiyama, N. Mentens, L. Batina, B. Preneel, and I. Verbauwhede, "Reconfigurable modular arithmetic logic unit for high-performance public-key cryptosystems," in *Reconfigurable Computing: Architectures and Applications*, ser. Lecture Notes in Computer Science, K. Bertels, J. Cardoso, and S. Vassiliadis, Eds. Springer Berlin Heidelberg, 2006, vol. 3985, pp. 347–357.
- [182] A. Byrne, N. Meloni, F. Crowe, W. Marnane, A. Tisserand, and E. Popovici, "SPA resistant elliptic curve cryptosystem using addition chains," in *Proc. Int. Conf. ITNG*, Apr. 2007, pp. 995–1000.
- [183] Z. Liu, J. Groszschädl, Z. Hu, K. Jarvinen, H. Wang, and I. Verbauwhede, "Elliptic curve cryptography with efficiently computable endomorphisms and its hardware implementations for the internet of things," *IEEE Trans. Comput.*, vol. PP, no. 99, pp. 1–1, 2016.
- [184] G. Chen, G. Bai, and H. Chen, "A high-performance elliptic curve cryptographic processor for general curves over $GF(p)$ based on a systolic arithmetic unit," *IEEE Trans. Circuits Syst. II*, vol. 54, no. 5, pp. 412–416, May 2007.
- [185] J. Y. Lai and C. T. Huang, "Elixir: High-throughput cost-effective dual-field processors and the design framework for elliptic curve cryptography," *IEEE Trans. VLSI Syst.*, vol. 16, no. 11, pp. 1567–1580, Nov. 2008.
- [186] A. Satoh and K. Takano, "A scalable dual-field elliptic curve cryptographic processor," *IEEE Trans. Comput.*, vol. 52, no. 4, pp. 449–460, Apr. 2003.
- [187] K. K. Duane E. Galbi, "Measuring active power using pt px-a user perspective snug boston," 2010.
- [188] H. M. Edwards, "A normal form for elliptic curves," in *Bulletin of the American Mathematical Society*, pp. 393–422.

- [189] D. J. Bernstein, T. Lange, and R. Rezaeian Farashahi, *Binary Edwards Curves*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 244–265.
- [190] N. P. Smart, “The hessian form of an elliptic curve,” in *Proc. CHES '01*. London, UK, UK: Springer-Verlag, 2001, pp. 118–125.