ISTANBUL TECHNICAL UNIVERSITY ★ INSTITUTE OF SCIENCE AND TECHNOLOGY

HARDWARE IMPLEMENTATIONS OF ECC OVER A BINARY EDWARDS CURVE

M.Sc. Thesis by

Ünal KOCABAŞ, M.Sc.

Department : **Electronics and Communication Engineering**

Programme : **Electronics Engineering**

JULY 2009

ISTANBUL TECHNICAL UNIVERSITY ★ INSTITUTE OF SCIENCE AND TECHNOLOGY

HARDWARE IMPLEMENTATIONS OF ECC OVER A BINARY EDWARDS CURVE

M.Sc. Thesis by

Ünal KOCABAŞ, M.Sc.

(504071224)

Date of Submission : **4 May 2009**

Date of Examin : **22 July 2009**

Supervisor : **Ass. Prof. Dr. Sıddıka Berna ÖRS YALÇIN**

Members of the Examining Committee **Ass. Prof. Dr. Devrim Yılmaz AKSIN**

Ass. Prof. Dr. Osman Kaan EROL

JULY 2009

İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

BİR İKİLİ EDWARDS EĞRİSİNİN DONANIMSAL GERÇEKLEMELERİ

YÜKSEK LİSANS TEZİ

Y.Müh Ünal KOCABAŞ

(504071224)

Tezin Enstitüye Verildiği Tarih : 4 Mayıs 2009

Tezin Savunulduğu Tarih : 22Temmuz 2009

Tez Danışmanı : Ass. Prof. Dr. Sıddıka Berna ÖRS YALÇIN

Diğer Jüri Üyeleri Ass. Prof. Dr. Devrim Yılmaz AKSIN

Ass. Prof. Dr. Osman Kaan EROL

Temmuz 2009

ACKNOWLEDGEMENT

Firstly, I would like to thank my supervisor Ass. Prof. Dr. S. Berna Örs Yalçın who has supported me during my master and has introduced cryptography to me and encouraged me to come to Katholieke Universiteit Leuven for my master thesis.

I would also like to thank my supervisors Dr. Lejla Batina and Prof. Ingrid Verbauwhede for their guidance and support during the thesis and assistance on official problems.

I find it necessary to thank Miroslav Knežević and Vladimir Rožić for their endless support, friendship and advices during my implementation.

I am very grateful to Kerem Varıcı and Özgül Küçük for their sincere friendship and precious favors and corrections.

I am thankful and dedicate my thesis to my family, who are the references of my accomplishments. Their support has encouraged me to work hard and reach the best in my life.

Finally, I would like to thank my love Emanuela Zaraj for her love and support and the happiness she is bringing in my life. It is a great feeling that she is always with me whenever I need her.

July 2009 Ünal KOCABAŞ

TABLE OF CONTENTS

ABBREVATIONS **ix**
LIST OF TABLES **xi**
LIST OF FIGURES **xiii**
LIST OF SYMBOLES **xv**
SUMMARY **xvii**
OZET **xix**

1. INTRODUCTION **1**
1.1. Motivation 1
1.2. Organization of Thesis 2

2. CRYPTOSYSTEMS **3**
2.1. Symmetric-key Cryptosystems 3
2.2. Asymmetric-key Cryptosystems 4
2.2.1. Key Generation 5
2.2.2. Public-key Encryption 5
2.2.3. Digital Signature 6
2.2.4. Diffie-Hellman Key Management 7

3. ESSENTIAL CONCEPTS **9**
3.1. Integers 9
3.1.1. The Integers modulo n 10
3.2. Groups 12
3.3. Rings 13
3.3.1. Polynomial Rings 13
3.4. Fields 15
3.4.1. Finite Fields 15
3.4.1.1. Addition and Subtraction 16
3.4.1.2. Multiplication 16

4. ELLIPTIC CURVE CRYPTOSYSTEMS **19**
4.1. Discrete Logarithm Problem 19
4.1.1. Diffie-Hellman Key Agreement Protocol 20
4.1.2. The El-Gamal Cryptosystem 21
4.1.3. Elliptic Curve Discrete Logarithm Problem 21
4.2. Introduction to Elliptic Curves 23
4.2.1. Weierstrass Equation 24
4.2.2. Point Addition over Finite Fields 25
4.3. Point Multiplication 26
4.4. Projective Coordinates 29

4.4.1. Standard Projective Coordinates 29
4.4.2. Jacobian Coordinates 30

5. EDWARDS CURVES 31
5.1. Binary Edwards Curves 32
5.1.1. Introduction to Binary Edwards Curves 32
5.1.2. Binary Edwards Curves Addition Law 33
5.1.3. Complete Binary Edwards Curves 33
5.1.4. Explicit Addition Formulas 34
5.1.4.1. Affine Addition 34
5.1.4.2. Mixed Addition 34
5.1.4.3. Projective Addition 35
5.1.5. Doubling 36
5.1.5.1. Affine Doubling 37
5.1.5.2. Projective Doubling 38
5.1.6. Differential Addition 39

6. BINARY EDWARDS CURVES IMPLEMENTATION 41
6.1. Implementation of Binary Edwards Curves 41
6.1.1. Processor 43
6.1.1.1. MALU (Modular Arithmetic Logic Unit) Design 43
6.1.1.2. Register File Design 45
6.1.1.3. Shifter Design 48
6.1.1.4. Control Block 49
6.1.2. Bus Manager 51
6.1.3. Memory Units 53
6.1.3.1. ROM 53
6.1.3.2. RAM 53
6.2. Algorithms for Implementation of Binary Edwards Curves 54

7. RESULTS 57
7.1. Power Estimation Methodology 57
7.2. Area, Power Consumption in Different Frequencies 58
7.3. Trade-off 61
7.4. Clock-gating 62

8. CONCLUSION 65

REFERENCES 67

APPENDIX 69

A. Control Sequence of Control Block 69

B. Control Bits of Assign Operation 71

C. Codes for Cell and MALU 73

D. Comparison Between Normal Clocking and Clock-gating **75**

E. Simulation Examples **77**

CIRCULUM VITAE **84**

ABBREVATIONS

AES	:	Advanced Encryption Standard
BEC	:	Binary Edwards Curves
CDHP	:	Computational Diffie-Hellman Problem
DLP	:	Discrete Logarithm Problem
DPA	:	Differential Power Analysis
EC	:	Elliptic Curve
ECC	:	Elliptic Curve Cryptosystem
ECDLP	:	Elliptic Curve Discrete Logarithm Problem
FSM	:	Finite State Machine
GPG	:	GNU Privacy Guard
LSB	:	Least Significant Bit
LUT	:	Look-Up Table
MALU	:	Modular Arithmetic Logic Unit
MSB	:	Most Significant Bit
NESSIE	:	New European Schemes for Signatures, Integrity and Encryption
NIST	:	National Institute of Standards and Technology
PGP	:	Pretty Good Privacy
PKC	:	Public-key Cryptography
RAM	:	Random-access Memory
RFID	:	Radio-Frequency Identification
ROM	:	Read-Only Memory
RSA	:	Rivest-Shamir-Adleman
SPA	:	Simple Power Analysis
SSL	:	Secure Sockets Layer

LIST OF TABLES

		Page No
Table 3.1	Addition in Finite Fields	17
Table 4.1	NESSIE Recommendations	23
Table 5.1	The Speed of Binary Edwards Curves Differential Addition	39
Table 6.1	Example of inversion in $F_{2^{163}}$	55
Table 7.1	Results of implementation in $100kHz$ in $0.13\mu m$ technology	59
Table 7.2	Results of implementation in $400kHz$ in $0.13\mu m$ technology	59
Table 7.3	Results of implementation in $1MHz$ in $0.13\mu m$ technology	60
Table 7.4	Results of implementation in $5MHz$ in $0.13\mu m$ technology	60
Table 7.5	Results of implementation in $20MHz$ in $0.13\mu m$ technology	60
Table 7.6	Results of implementation in $50MHz$ in $0.13\mu m$ technology	60
Table 7.7	Our example design for $d = 4$ and $400kHz$	61
Table 7.8	Results of implementation in $400kHz$ in $0.13\mu m$ technology after clock-gating	63
Table 7.9	Results of implementation in $5MHz$ in $0.13\mu m$ technology after clock-gating	64

LIST OF FIGURES

		Page No
Figure 2.1 :	Symmetric-key Cryptosystems	3
Figure 2.2 :	Key Generation	5
Figure 2.3 :	Public-key Encryption	6
Figure 2.4 :	Public-key Digital Signature Diagram	6
Figure 2.5 :	Diffie-Hellman Key Management Scheme	7
Figure 3.1 :	Modulo Operation over Rijndael Finite Field	17
Figure 4.1 :	Diffie-Hellman Key Agreement Protocol	20
Figure 4.2 :	El-Gamal Cryptosystem	22
Figure 4.3 :	Point Addition of a Point in Elliptic Curve Equation $y^2 = x^3 - 50x + 100$	26
Figure 4.4 :	Doubling of a Point in Elliptic Curve Equation $y^2 = x^3 - 50x + 100$	27
Figure 4.5 :	Hierarchy of Elliptic Curve Cryptosystems	27
Figure 6.1 :	The architecture of Binary Edwards Curves Processor (BEC Processor)	43
Figure 6.2 :	BEC Processor's MALU Architecture	44
Figure 6.3 :	Control Scheme of Cell and MALU with $d = 4$	45
Figure 6.4 :	Register File Architecture	46
Figure 6.5 :	Shifting Operation in *regB* and Data Assigning, Taking Process in *regD*	47
Figure 6.6 :	The Architecture of Shifter Block	48
Figure 6.7 :	The Finite State Machine Diagram of Shifter Component	49
Figure 6.8 :	The Processor of Binary Edwards Curves	50
Figure 6.9 :	The Map of Address Control	52
Figure 6.10:	Architecture of Bus Manager	52
Figure 6.11:	RAM Blocks and Storing Values	53
Figure 6.12:	Required Operations for Binary Edwards Curves Implementation	54
Figure 7.1 :	Power estimation flow	58
Figure 7.2 :	Area Consumption vs. Time	61
Figure 7.3 :	Throughput vs. Power Consumption in $d = 4$	62
Figure 7.4 :	Power Consumption in $5MHz$ with different digit sizes	62
Figure 7.5 :	Process of Clock gating	63
Figure A.1:	The Finite State Machine of Control Block	69
Figure B.1:	Assign Operation Control Map	71
Figure C.1:	Codes for Cell and MALU	73
Figure D.1:	Area Consumption vs. Time	75
Figure D.2:	Frequency vs. Power Consumption in $d = 4$	76
Figure D.3:	Power Consumption in $5MHz$ with different digit sizes	76
Figure E.1:	Assigning Key Value in Modelsim	77
Figure E.2:	Simulation in GEZEL for First Four Projective Addition	78
Figure E.3:	Figure of Simulation in Modelsim for First X3 Value	79

Figure E.4 : Figure of Simulation in Modelsim for First Y3 Value 80
Figure E.5 : Figure of Simulation in Modelsim for First Z3 Value 81
Figure E.6 : Simulation in GEZEL for Final Points After Inversion 82
Figure E.7 : Figure of Simulation in Modelsim for Final Points 83

LIST OF SYMBOLES

$\mathbb{C}$	:	Complex numbers
$\mathbb{Q}$	:	Rational numbers
$\mathbb{R}$	:	Real numbers
$\mathbb{Z}$	:	Integer numbers
$\mathbb{Z}_p$	:	Integer numbers (mod p)
G	:	Group
α	:	Generator of a group
p	:	Prime number
e_k	:	Encryption with key 'k'
d_k	:	Decryption with key 'k'
$\mathbb{F}_q$	:	Finite field over 'q'
E	:	An elliptic curve
$E(K)$	:	An elliptic curve over the field 'K'
λ	:	Point addition constant
E_{B,d_1,d_2}	:	Binary Edwards curve with constants d_1 and d_2
X	:	The X coordinate of a point
Y	:	The Y coordinate of a point
Z	:	The Z coordinate of a point

HARDWARE IMPLEMENTATIONS OF ECC OVER A BINARY EDWARDS CURVE

SUMMARY

A technological progress and its increasing influence on our daily life have made cryptology more evident. Daily applications such as ATM and ID cards, computer passwords, secure e-mail, online banking and online commerce are made via certain cryptology protocols. In early times, the specification of designs depended on high speed and performance. In addition to these, nowadays, low power and small die area implementations have become more important according to the widespread usage of limited power and area specifications.

In this study, Edwards curve is implemented in minimum area specification which is defined over all of the points of the elliptic curves ("completeness") as proposed in April 2008. The statement of the study includes the implementation steps, the optimization over the number of registers and the parallelism of the design to gain speed. The implementation is designed in such a way that it is secure against the simple power analysis (SPA), although it slows down the process speed. Finally, the comparison of power, area and speed are indicated.

BİR İKİLİ EDWARDS EĞRİSİNİN DONANIMSAL GERÇEKLEMELERİ

ÖZET

Teknolojinin gelişmesi ve hayatımıza olan etkisinin artmasıyla, kriptolojinin günlük hayatımıza etkileri de belirgin bir biçimde görülmeye başlamıştır. ATM ve ID kart şifreleri, bilgisayar şifreleri, güvenli e-posta, online bankacılık işlemleri ve online ticaret gibi günlük kullanımlarımız belirli kriptografik protokoller üzerinden gerçekleştirilmektedir. İlk zamanlarda yüksek hız ve yüksek işlem gücü kapasitesine sahip devreler tasarlanmasına çalışılırken, günümüzde enerji ve alan kısıtlamasında olan kullanım alanlarının yaygınlaşmasıyla güç ve alan tasarruflu gerçeklemeler büyük önem kazanmıştır.

Bu çalışmada, Nisan 2008'de sunulan ve eliptik eğrinin tüm noktalarını tanımlayan ("bütünlük") Edwards Eğrilerinin minimum alanda gerçeklenmesine çalışılmıştır. Gerçekleme adımları, register sayısı optimizasyonu ve paralel işlemlerle döngünün hızlandırılması anlatılmıştır. Gerçeklemenin güvenlik durumunun SPA (Simple Power Analysis)'ya dayanıklı tasarımı belirtilmiş ve işlemi yavaşlatmasına rağmen yan-kanal ataklarına karşı dayanıklı olarak tasarlanmıştır. Son olarak, güç, alan ve hız karşılaştırılmaları verilmiştir.

1. INTRODUCTION

1.1 Motivation

Cryptography is one of the oldest fields of technical study which is going back thousands of years. The title cryptography is inspired from Greek words "*kryptós*", which means "hidden, secret" and "*gráfo*", which means "writing". Until recent decades, it has been known as the methods of encryption that use a pen and a paper. More formally, today, cryptography is the art of encoding data in a way that only the intended recipient can decode it, and knows that the message is authenticated and unchanged [1].

Since early ages, empires and governments have been using the cryptography for sending messages in a secure manner. The earliest example of cryptography is found in non-standard hieroglyphs carved into monuments from Egypt's Old Kingdom which are more than 4500 years old. Moreover, cryptography is also used by the Spartans in 5BC. This warrior society developed a cryptographic device to send and receive secret messages. This device, a cylinder called a "*Scytale*", was in the possession of both the sender and the recipient of the message. To prepare the message, a narrow strip of leather, was wound around the Scytale and the message was written across it. To read the message, it was re-wound onto a Scytale of exactly the same diameter. Finally, Caesar's method can be given as an example to the classic cryptography, which relies on shifting the letters by three. In those times, cryptographic methods relied on just the secrecy of algorithm which is called security by obscurity. Today, these ciphers stayed of a historical interest and are not adequate for a real-world situation.

The modern times cryptography is based on mathematical algorithms whose security is based on a hard mathematical problem. Before 1960s, when the computers were not the key primitives of our lives, the cryptography had been used generally for military purposes as a tool to protect national secrets and strategies. The role of cryptography

affected the tactics of the World War I and II, and new boundaries were drawn depending on the developments in the cryptography. After 1960s, the widespread usage of computers and communication systems brought a demand from the public for means to protect information in digital form. The increasing necessity of security services evoked to design DES (Data Encryption Standard) by IBM as a U.S. Federal Information Processing Standard for encrypting unclassified information in 1977 [2]. Afterwards, the fast proceeding of computers led to need a new standard called AES (Advanced Encryption Standard) which was designed by Rijmen and Daemon in 1998. Today, it is most widely used algorithm in the world.

Nowadays, the cryptography is being used in many technologically advanced applications, such as; ATM cards, smart cards, identification cards, secure e-mail, computer passwords, biometrics and electronic commerce.

All modern algorithms use a key to control encryption and decryption. Basically, cryptographic methods can be classified into two main branches like symmetric (private key) cryptosystems and asymmetric (public-key) cryptosystems.

1.2 Organization of Thesis

In this thesis, the first hardware implementation of Binary Edwards Curves has been designed, which includes the feature of completeness and is compatible with RFID-tags. In chapter 2, a brief information about the classification of cryptosystems is given and the idea behind public key cryptosystems is explained. Chapter 3 gives an essential concept of mathematical explanations briefly. The discrete logarithmic problem, introduction to elliptic curves with point addition and point multiplication, projective coordinates and elliptic curve discrete logarithm problem are summarized in chapter 4. Chapter 5 gives Edwards curves and binary Edwards curves with explicit formulas of addition and doubling. In chapter 6 the implementation details of binary Edwards curves and working principles are explained. The efficiency of circuit is discussed. Chapter 7 gives the final results and trade-offs. Finally, chapter 8 concludes binary Edwards curves implementation and efficiency.

2. CRYPTOSYSTEMS

2.1 Symmetric-key Cryptosystems

Symmetric algorithms, also called secret-key algorithms, use the same key for both encryption and decryption. Basically, a sender and a receiver share a secret key "*K*", which is used to encrypt a *plain-text* with "*K*" and an encryption rule. After the receiver receives a *cipher-text*, the same secret key is used to decrypt the *cipher-text* to the *plain-text* with a decryption rule. The security of system is depended on the secrecy of key, therefore the key is not to be leaked to the outside, should be changed often and be sufficiently random. A symmetric-key cryptosystem is illustrated in Figure 2.1.

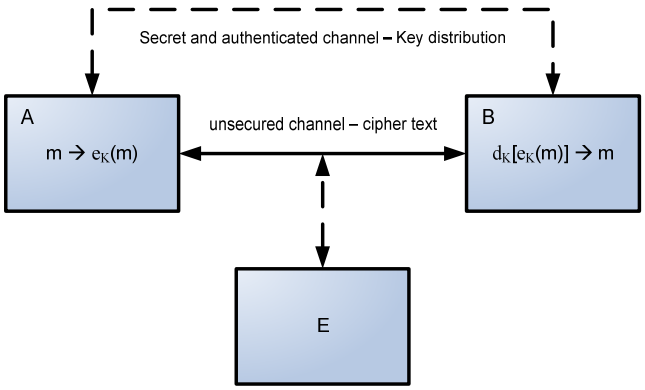

Figure 2.1: Symmetric-key Cryptosystems

In Figure 2.1, an entity A encrypts a message with "*K*" and sends it through an unsecured channel to an entity B. Another entity E, called eavesdropper, listens the unsecured channel. But he/she gets only a *cipher-text* and can not infer a meaningful message. When an entity B receives and decrypts the *cipher-text* by using the same "*K*", the communication is finished securely. One drawback of this scheme is that the shared key, K, must be distributed before the communication occurs.

Symmetric-key algorithms can be further divided into two categories: stream ciphers and block ciphers. Stream ciphers generate an arbitrary long key and the encryption is performed by combining it with the *plain-text* bit-by-bit. In contrast, block ciphers take a block (some fixed number of bits) of *plain-text* and a key, and give an output a block of *cipher-text* of the same size. Moreover, different ciphers use different length of keys and a longer key usually means higher security. The most popular example of a block cipher is Advanced Encryption Standard algorithm (AES), which is approved by NIST in December 2001 and uses $128, 192, 256$-bit blocks [3].

Symmetric-key algorithms are generally much less computationally intensive than asymmetric ones and use shorter keys. According to the [4], both in software and hardware public-key encryption algorithms are two to three orders of magnitude slower than symmetric algorithms. For example, a 1024-bit exponentiation with a thirty-two-bit exponent takes $360\mu s$ on a 1 GHz Pentium III; this corresponds to 2800 cycles/byte; a decryption with a 1024-bit exponent takes 9.8ms or 76000 cycles/byte. This speed should be compared to 15cycles/byte for AES. Although symmetric-key algorithms have many advantages like high efficiency, a problem occurs when a communication system needs to share the same private key through an insecure channel.

The process of selecting, distributing and storing keys is known as key management, and it is difficult to achieve these in secure [5]. In this point, the asymmetric-key cryptosystem is used to establish a secret key, which is then used in a symmetric-key cryptosystem.

2.2 Asymmetric-key Cryptosystems

In contrast to symmetric cryptosystem, asymmetric one has a pair of keys; a public and a private key. This system is also called *public-key cryptosystem*. The public-key cryptosystem consists of a public key, which can be used for encryption and verification of a signature, and a private key, used for decryption and creation of a signature. Everyone publish their public key and keep their private key secret.

The idea of public-key cryptography was introduced in the mid 70's by Diffie and Hellman [6]. In public-key cryptosystem, two different keys are generated,

which are affiliated with each other by trapdoor functions. Trapdoor functions are easy to apply in one direction but extremely difficult to apply in the inverse [7]. Public-key cryptosystems are used in key establishment protocols, data integrity, entity authentication and for encrypting small datas such as credit card numbers and PINs [2].

The generation of keys, encryption and signature schemes are going to be discussed in more details.

2.2.1 Key Generation

First of all, each entity creates its own private and public key. The public-key generation function uses an unpredictable (typically a large randomly chosen) number to generate a valid key pair. The process is shown in Figure 2.2.

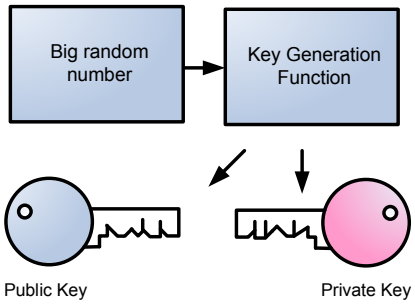

Figure 2.2: Key Generation

After key generation is executed, these keys can be used for the public-key encryption.

2.2.2 Public-key Encryption

In public encryption, every entity has a public key "e" and a corresponding private key "d". As stated before, a public key is used for encryption (Ee(m)), and a private key is used for decryption (Dd(c)). This is illustrated in Figure 2.3. In secure systems, the task of computing "d" with given "e" is computationally infeasible [2].

In Figure 2.3, an entity B wants to send a message "m" to the A, and he/she takes an authentic copy of A's public key "e". Now, B uses the encryption transformation to obtain the cipher-text "$c = Ee(m)$", and transmits "c" to A over an unsecured channel. Anyone can see this cipher-text, but only A, who has the related private key, can decrypt this message. To decrypt "c", A applies the decryption transformation and

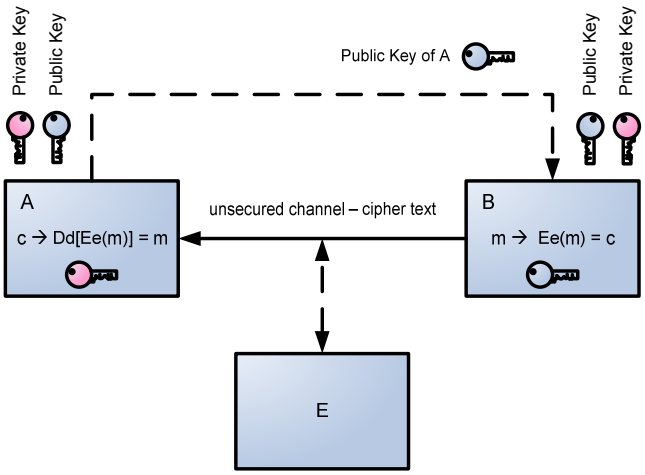

Figure 2.3: Public-key Encryption

obtains the original message "$m = Dd(c)$". In public-key encryption, security rely only on the secrecy of private key [2].

2.2.3 Digital Signature

The main objective of public-key encryption is to provide secrecy or confidentiality. Since A's encryption transformation is on public knowledge, public-key encryption alone does not provide data authentication or data integrity [2]. Anybody can send a cipher-text to the A, and there is no reason for A to believe that the message was sent by the claimed identity unless a digital signature is used. The setting of public-key cryptosystem also allows the application of digital signatures. These settings are shown in Figure 2.4 to generate a signature.

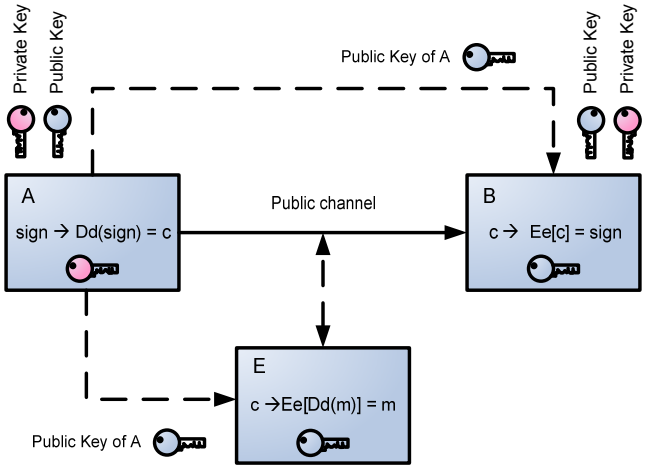

Figure 2.4: Public-key Digital Signature Diagram

In this protocol shown in Figure 2.4, an entity A signs a message with its private key "d" and sends it to an entity B. To verify the signature, B has to look up the public key "e" of A and compute "$Ee(Dd(sign)) = sign$". Here, anybody can see and decrypt the signature of A, but no one can just copy signature of the A and send the messages to B as A. Since the signature can only be created by A's secret key, it's validity depends on the security of the private key.

Finally, public-key cryptosystems can also be used to generate shared secret key without an authenticated and secure channel.

2.2.4 Diffie-Hellman Key Management

In 1976, Whitfield Diffie and Martin Hellman published a key management scheme [6]. In this scheme, each entity generates own public and private key pair, and distribute their public key. After obtaining an authentic copy of each others public keys, A and B can compute a shared key offline for a symmetric cipher in the diagram is shown in Figure2.5.

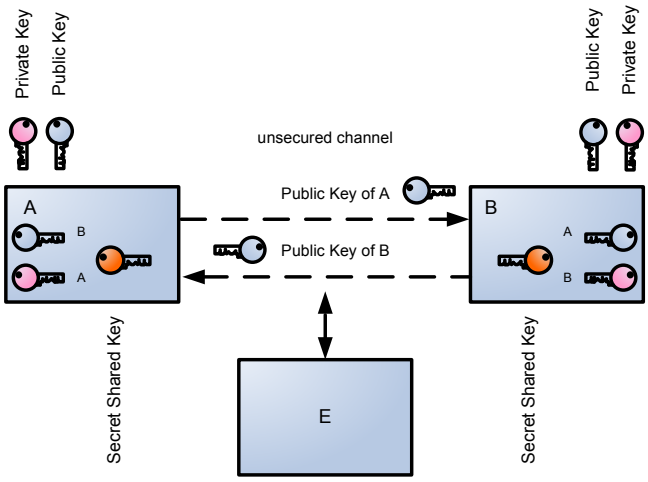

Figure 2.5: Diffie-Hellman Key Management Scheme

After all, public-key cryptosystems can be compared by means of their security, key lengths, speed and implementation issues. In terms of security, the hardness of the underlying mathematical problem determines the intractability of the system [7].

In short, modern cryptosystems take the advantages of both asymmetric and symmetric algorithms. Asymmetric algorithms are used at the first stages to provide authenticated channel and key distribution, and then symmetric key algorithms are used for encryption. For instance, this type of hybrid approach is used in SSL, PGP and GPG, etc [2].

3. ESSENTIAL CONCEPTS

In this chapter, we present mathematical background of this study and introduce a few basic definitions. These definitions will be useful to support ideas of the later chapters. First of all, we give the properties and definitions of integers then we will establish other subsections over these basic informations.

3.1 Integers

The set of integers $\{\ldots, -3, -2, -1, 0, 1, 2, 3, \ldots\}$ is denoted by the symbol $\mathbb{Z}$. For a given finite set A, the number of elements of A is denoted by $\sharp A$. The following definitions involve the basic properties of integers that we use these explanations to define some operations on the next sections.

Definition 3.1 : (*Division algorithm for integers*) If a and b are integers with $b \geq 1$, then ordinary long division of a by b yields integers q (*the quotient*) and r (*the remainder*) such that;

$$a = q.b + r, \ \ where \ \ 0 \leq r < b \tag{3.1}$$

The remainder of the division is denoted as *a mod b*, and the quotient is denoted as *a div b*. In the other denotation, *a div b* $= [a/b]$ and *a mod b* $= a$ - $b.[a/b]$.

Definition 3.2 : (*Greatest common divisor*) An integer c is a common divisor of a and b if $c \mid a$ and $c \mid b$. Moreover, a non-negative integer d is the greatest common divisor of integers a and b, denoted $d = \gcd(a,b)$, if

(i) d is a common divisor of a and b,

(ii) whenever $c \mid a$ and $c \mid b$, then $c \mid d$.

Equivalently, $\gcd(a,b)$ is the largest positive integer that divides both a and b, with the exception that $\gcd(0,0) = 0$. a,$b \in \mathbb{Z}$ are called relatively prime if and only if $\gcd(a,b) = 1$.

Definition 3.3 : (*Least common multiple*) A non-negative integer d is the least common multiple of integers a and b, denoted d = lcm(a,b), if

(i) $a \mid d$ and $b \mid d$,

(ii) whenever $a \mid c$ and $b \mid c$, then $d \mid c$.

Equivalently, *lcm(a,b)* is the smallest non-negative integer divisible by both a and b. In other denotation, lcm(a,b)=$a.b/\gcd(a,b)$.

3.1.1 The Integers modulo n

The following definitions of integers are given in modulo n. Let n be a positive integer. The integers modulo n denoted as $\mathbb{Z}_n$, is the set of integers $\{0,1,2,\ldots,n-1\}$. Addition, subtraction and multiplication in $\mathbb{Z}_n$ are performed in modulo n.

Definition 3.4 : (*Congruency*) If a and b are integers, then a is said to be *congruent* to b *modulo* n, written $a \equiv b \pmod{n}$, if n divides $(a-b)$. The integer n is called the *modulus* of the congruence. The properties of congruence are given for all $a,\ a_1,\ b,\ b_1,\ c \in \mathbb{Z}$.

1. $a \equiv b$ (mod n) if and only if a and b leave the same remainder when divided by n.
2. (*reflexivity*) $a \equiv a \pmod{n}$
3. (*symmetric*) If $a \equiv b \pmod{n}$ then $b \equiv a \pmod{n}$
4. (*transitivity*) If $a \equiv b \pmod{n}$ and $b \equiv c \pmod{n}$, then $a \equiv c \pmod{n}$
5. If $a \equiv a_1 \pmod{n}$ and $b \equiv b_1 \pmod{n}$, then $a+b \equiv a_1 + b_1 \pmod{n}$ and $a.b \equiv a_1 \,.\, b_1 \pmod{n}$.

Definition 3.5 : (*Multiplicative inverse*) Let $a \in \mathbb{Z}_n$. The multiplicative inverse of a modulo n is an integer $x \in \mathbb{Z}_n$ such that $a.x \equiv 1 \pmod{n}$. If such an x exists, then a is said to be invertible; the inverse of a is denoted by a^{-1}. This condition can be provided if and only if *gcd(a,n) = 1*.

The multiplicative inverse operation is used in our implementation depending on Fermat's theorem. To clarify the Fermat theorem, Euler phi function and multiplicative group of $\mathbb{Z}_n$ is defined in following definitions.

Definition 3.6 : (Euler phi function) For $n \geq 1$, let $\phi(n)$ denote the number of integers in the interval $[1,n]$ which are relatively prime to n. The function ϕ is called the *Euler phi function*. The properties of Euler phi function are given;

(i) If p is a prime, then $\phi(p) = p-1$.

(ii) The Euler phi function is multiplicative. That is, if *gcd(m,n) = 1*, then $\phi(mn) = \phi(m).\phi(n)$.

(iii) If $n = p_1^{e_1}.p_2^{e_2}\ldots p_k^{e_k}$ is the prime factorization of n, then

$$\phi(n) = n(1-\frac{1}{p_1})(1-\frac{1}{p_2})\ldots(1-\frac{1}{p_k}). \tag{3.2}$$

Definition 3.7 : The multiplicative group of $\mathbb{Z}_n$ is $\mathbb{Z}_n^* = \{a \in \mathbb{Z}_n \mid gcd(a,n) = 1\}$. In particular, if n is a prime, then $\mathbb{Z}_n^* = \{a \mid 1 \leq a \leq n-1\}$. Moreover the order of $\mathbb{Z}_n^*$ is defined to be the number of elements in $\mathbb{Z}_n^*$, namely $\mid \mathbb{Z}_n^* \mid$ [2].

From the Euler phi function that $\mid \mathbb{Z}_n^* \mid = \phi(n)$. Note also that if $a \in \mathbb{Z}_n^*$ and $b \in \mathbb{Z}_n^*$, then $a.b \in \mathbb{Z}_n^*$, and so $\mathbb{Z}_n^*$ is closed under multiplication.

Fact Let $n \geq 2$ be an integer.

(i) (*Euler's theorem*) If $a \in \mathbb{Z}_n^*$, then $a^{\phi(n)} \equiv 1$ (mod n).

(ii) If n is a product of distinct primes, and if $r \equiv s \pmod{\phi(n)}$, then $a^r \equiv a^s$ (mod n) for all integers a. In other words, when working modulo such an n, exponents can be reduced modulo $\phi(n)$.

A special case of Euler's theorem is Fermat's (little) theorem. Let p be a prime.

(i) (*Fermat's theorem*) If $gcd(a,p) = 1$, then $a^{p-1} \equiv 1 \pmod{p}$.

(ii) If $r \equiv s \pmod{p-1}$, then $a^r \equiv a^s \pmod{p}$ for all integers a. In other words, when working modulo a prime p, exponents can be reduced modulo $p-1$.

(iii) In particular, $a^p \equiv a \pmod{p}$ for all integers a.

These properties is used in our design to calculate the inversion of Z coordinate. The method of finding inverse of Z coordinate is given in Table 6.1.

Definition 3.8 : Let $\alpha \in \mathbb{Z}_n^*$. If the order of α is $\phi(n)$, then α is said to be a *generator* or a *primitive element* of $\mathbb{Z}_n^*$. If $\mathbb{Z}_n^*$ has a generator, then $\mathbb{Z}_n^*$ is said to be *cyclic* [2].

The properties of generators of $\mathbb{Z}_n^*$ are given;

(i) $\mathbb{Z}_n^*$ has a generator if and only if $n = 2,\ 4,\ p^k$ or $2p^k$, where p is an odd prime and $k \geq 1$. In particular, if p is a prime, then $\mathbb{Z}_n^*$ has a generator.

(ii) If α is a generator of $\mathbb{Z}_n^*$, then $\mathbb{Z}_n^* = \{\alpha^i \ mod\ n \mid 0 \leq i \leq \phi(n) - 1\}$.

(iii) Suppose that α is a generator of $\mathbb{Z}_n^*$. Then $b = \alpha^i \bmod n$ is also a generator of $\mathbb{Z}_n^*$ if and only if $gcd(i, \phi(n)) = 1$. It follows that if $\mathbb{Z}_n^*$ is cyclic, then the number of generators is $\phi(\phi(n))$.

(iv) $\alpha \in \mathbb{Z}_n^*$ is a generator of $\mathbb{Z}_n^*$ if and only if $\alpha^{\phi(n)/p} \neq 1 \pmod{n}$ for an each prime divisor p of $\phi(n)$.

3.2 Groups

Definition 3.9 : A group $(G, *)$ consists of a set G with a binary operation $*$ on G satisfying the following three axioms [2].

1. The group operation is *associative*. That is, $a*(b*c) = (a*b)*c$ for all $a, b, c \in G$.
2. There is an element $1 \in G$, called the *identity element*, such that $a*1 = 1*a = a$ if all $a \in G$.
3. For each $a \in G$ there exists an element $a^{-1} \in G$, called the *inverse of a*, such that $a*a^{-1} = a^{-1}*a = 1$.
4. A group G is *abelian* (or *commutative*) if, furthermore, $a*b = b*a$ for all $a, b \in G$.

The notation of $(G, *)$ is used to represent multiplicative group, the identity element is represented by 1 and the inverse of a is denoted as a^{-1}. If the group operation is addition with the notation $(G, +)$, then the group is said to be an *additive* group, the identity element is denoted by 0, and the inverse of a is denoted $-a$.

If G is a finite group, then the number of elements of G is called the order of G and it is denoted as $|G|$. An element of group G, $a \in G$. The order of a is defined to be the least positive integer t such that $a^t = 1$, provided that such an integer exists. If such a 't' does not exist, then the order of a is defined to be ∞.

Definition 3.10 : A group G is *cyclic* if there is an element $\alpha \in G$ such that for each $b \in G$, there is an integer i with $b = \alpha^i$. Such an element α is called a *generator* of G.

For example, the set $\mathbb{Z}_n = \{0, 1, 2, \ldots, n-1\}$ is a cyclic group of order n under addition modulo n, i.e. $a + b \equiv r \; mod \; n$, where $r < n$ (r is the remainder when $a + b$ is divided by n) [2].

3.3 Rings

Definition 3.11 : A ring $(R, +, \times)$ consists of a set R with two binary operations denoted $+$ (addition) and $\times$ (multiplication) on R, satisfying the following axioms.

1. $(R, +)$ is an abelian group with identity denoted 0.
2. The operation $\times$ is associative. That is, $a \times (b \times c) = (a \times b) \times c$ for all $a, b, c \in R$.
3. There is a multiplicative identity denoted 1, with $1 \neq 0$, such that $1 \times a = a \times 1 = a$ for all $a \in R$.
4. The operation $\times$ is distributive over $+$. That is, $a \times (b + c) = (a \times b) + (a \times c)$ and $(b + c) \times a = (b \times a) + (c \times a)$ for all $a, b, c \in R$.

The ring is a commutative ring if $a \times b = b \times a$ for all $a, b \in R$. From the third axiom, if R has an identity element, then it is said to be a unitary ring or a ring with unity element [2].

3.3.1 Polynomial Rings

Definition 3.12 : If R is a commutative ring, then a polynomial in the indeterminate x over the ring R is an expression of the form

$$f(x) = a_n x^n + \cdots + a_2 x^2 + a_1 x + a_0 \tag{3.3}$$

where each $a_i \in R$ and n is a positive integer. Here, the element a_i is called the coefficient of x^i in $f(x)$. The largest integer m for which $a_m \neq 0$ is called the degree of $f(x)$, denoted deg $f(x)$; a_m is called the leading coefficient of $f(x)$ [2].

Definition 3.13 : The polynomial ring $R[x]$ is formed by the set of all polynomials in the indeterminate x having coefficients from R. The standard polynomial addition and multiplication operations are performed with coefficient arithmetic in the ring R [2].

Given two polynomials,

$$f(x) = \sum_{i=0}^{n} a_i x^i \text{ and } g(x) = \sum_{i=0}^{n} b_i x^i$$

we define the **sum** of $f(x)$ and $g(x)$ as

$$f(x) + g(x) = \sum_{i=0}^{n} (a_i + b_i) x^i \tag{3.4}$$

Given two polynomials,

$$f(x) = \sum_{i=0}^{n} a_i x^i \text{ and } g(x) = \sum_{j=0}^{m} b_j x^j$$

we define the **product** of $f(x)$ and $g(x)$ as

$$f(x)g(x) = \sum_{k=0}^{n+m} (c_k) x^k, \textit{ where } c_k = \sum_{i=j=k} a_i b_j \tag{3.5}$$

We give an example to show addition and multiplication operations on the polynomial ring $\mathbb{Z}[x]$. Let $f(x) = x^3 + x + 1$ and $g(x) = x^2 + x$ be elements of our polynomial ring. The addition of two elements is,

$$f(x) + g(x) = x^3 + x^2 + 1 \tag{3.6}$$

and

$$f(x) \times g(x) = x^5 + x^4 + x^3 + x \tag{3.7}$$

Definition 3.14 : (*Division algorithm for $F[x]$*) Let $f(x)$,$g(x) \in F[x]$, with $g(x) \neq 0$. Then there exist unique polynomials $q(x), r(x) \in F[x]$ such that

$$f(x) = q(x)g(x) + r(x) \tag{3.8}$$

where the degree of $r(x)$ is less than the degree of $g(x)$. The polynomial $q(x)$ is called the *quotient*, while $r(x)$ is called the *remainder*. If $r(x)$ is the zero polynomial (*i.e. r(x)=0*), then $g(x)$ is said to be a divisor of $f(x)$. A non-constant polynomial $f(x)$ is said to be *irreducible over F* if it has no divisor of lower degree than $f(x)$ in $F[x]$ [2].

Again, we give an example to practice polynomial division on the polynomial ring. Let $f(x) = x^6 + x^5 + x^3 + x^2 + x + 1$ and $g(x) = x^4 + x^3 + 1$ in $\mathbb{Z}[x]$. Polynomial long division of $f(x)$ by $g(x)$ yields, $g(x) = x^2.h(x) + (x^3 + x + 1)$.

Hence $f(x)$ mod $g(x) = x^3 + x + 1$ and $f(x)$ div $g(x) = x^2$

3.4 Fields

Definition 3.15 : A field is a commutative ring in which all non-zero elements have multiplicative inverses [2].

The characteristic of a field is 0, if it is defined by addition over integer numbers that $\overbrace{1+1+\ldots+1}^{m\ times} \neq 0$ for any $m \geq 1$. Otherwise, the characteristic of the field is the least positive integer m such that $\sum_{i=1}^{m} 1 = 0$.

Moreover, $\mathbb{Z}_p$ is a field under the usual operations of addition and multiplication in modulo p, if and only if p is a prime number. Then $\mathbb{Z}_p$ has characteristic p [2].

3.4.1 Finite Fields

Definition 3.16 : A finite field is a field F which contains a finite number of elements. The order of F is the number of elements in F. The properties of a finite field F can be given with following axioms [2].

(i) If F is a finite field, then F contains p^m elements for some prime p and integer $m \geq 1$.

(ii) For every prime power order p^m, there is a unique finite field of order p^m. This field is denoted by $\mathbb{F}_{p^m}$, or $GF(p^m)$. The characteristic of $\mathbb{F}_{p^m}$ is p.

Definition 3.17 : A finite field $\mathbb{F}_q$ is given with the order $q = p^m$, p is a prime, the non-zero elements of $\mathbb{F}_q$ form a group under multiplication called multiplicative group of $\mathbb{F}_q$, denoted by $\mathbb{F}_q^*$ [2].

$\mathbb{F}_q^*$ is a cyclic group of order $q-1$. Hence $a^q = a$ for all $a \in \mathbb{F}_q$.

A polynomial basis representation is commonly used to represent the elements of a finite field $GF(q)$, where $q = p^m$. There exists an irreducible polynomial,$f(x)$, of degree m over $GF(p)$, then the polynomial representation of the finite field, $GF(p^m)$, can be given in the following form.

$$g(x) = a_{m-1}x^{m-1} + a_{m-2}x^{m-2} + \ldots + a_1x^1 + a_0 \tag{3.9}$$

where $\{0 \leq a_{m-1}, a_{m-2}, \ldots, a_1, a_0 \leq p-1\}$ and the greatest degree of field is $m-1$.

Addition: The representation of an addition in $GF(p^m)$ can be performed by adding the coefficient of same degrees in modulo p.

Multiplication : If $g(x), h(x) \in GF(p^m)$, then the product $g(x)h(x)$ can be formed by first multiplying $g(x)$ and $h(x)$ as polynomials by the ordinary method with modulo p for coefficients, and then taking the remainder after polynomial division by $f(x)$.

Multiplicative inversion: In $GF(p^m)$, it can be computed by using Fermat's Little theorem which is stated in Definition 3.7.

In our design, we use GF(2^{163}). So, the order of the finite field is of the form p^m, where p is a prime number called the characteristic of the field and 2, and m is a positive integer and 163.

We can explain effective polynomial representation to clarify operations over GF(2^{163}). A particular case in GF(p) is GF(2), where addition is exclusive OR (XOR) and multiplication is AND. Moreover, elements of GF(2^{163}) may be represented as polynomials of degree less than 163 over GF(2). Operations are then performed modulo $R(x)$ where $R(x)$ is an *irreducible polynomial* of degree 163 over GF(2). The addition of two polynomials P and Q is done as stated before; multiplication is done as follows: $W = P.Q$, then compute the remainder modulo $R(x)$. In our design, irreducible polynomial is set to $x^{163} + x^7 + x^6 + x^3 + 1$. It is possible to express elements of GF(2^{163}) as binary numbers, with each term in a polynomial represented by one bit in the corresponding element's binary expression.

3.4.1.1 Addition and Subtraction

Addition and subtraction are performed by adding or subtracting two of these polynomials together, and reducing the result modulo the characteristic. In a finite filed with characteristic 2 as ours, addition and subtraction are identical, and are accomplished using the XOR operation. Table 3.1 gives some examples over GF(2^{163}). Notice that under regular addition of polynomials, the sum would contain a term $2x^6$, but this term becomes $0x^6$ and is dropped when the answer is reduced modulo 2.

3.4.1.2 Multiplication

Multiplication in a finite field is multiplication modulo an irreducible reducing polynomial used to define the finite field. We give an example of multiplication over Rijndael's finite field.

Table 3.1: Addition in Finite Fields

p_1	p_2	$p_1 + p_2$ (normal algebra)	$p_1 + p_2$ in $GF(2^163)$
x^3+x+1	x^3+x^2	$2x^3+x^2+x+1$	x^2+x+1
x^4+x^2	x^6+x^2	$x^6+x^4+2x^2$	x^6+x^4
$x+1$	x^2+1	x^2+x+2	x^2+x
x^3+x	x^2+1	x^3+x^2+x+1	x^3+x^2+x+1
x^2+x	x^2+x	$2x^2+2x$	0

Rijndael uses a characteristic 2 finite field with 8 terms, which can also be called the GF(2^8). The following reducing polynomial is given for multiplication:
$x^8+x^4+x^3+x+1$.

For example, {53}.{CA} = {01} in Rijndael's field, it can be calculated as following steps;

$$
\begin{aligned}
&(x^6+x^4+x+1)(x^7+x^6+x^3+x)\\
&=x^{13}+x^{12}+x^9+x^7+x^{11}+x^{10}+x^7+x^5+x^8+x^7+x^4+x^2+x^7+x^6+x^3+x\\
&=x^{13}+x^{12}+x^{11}+x^{10}+x^9+x^8+x^6+x^5+x^4+x^3+x^2+x \quad mod\ x^8+x^4+x^3+x+1\\
&=1
\end{aligned}
\tag{3.10}
$$

Modulo operation can be demonstrated through long division, remainder gives the result value. Notice that *EXOR* is applied in the example and not arithmetic subtraction.

```
_____________
 11111101111110 (mod) 100011011
^100011011
 _____________
  1110000011110
 ^100011011
  _____________
   110110101110
  ^100011011
   _____________
    10101110110
   ^100011011
    _____________
     0100011010
    ^000000000
     _____________
      100011010
     ^100011011
      _____________
       00000001
```

Figure 3.1: Modulo Operation over Rijndael Finite Field

4. ELLIPTIC CURVE CRYPTOSYSTEMS

Widely usage of public cryptosystems in communication triggered the invention of new mathematical algorithms. The first proposals of the elliptic curves were made by Koblitz in [8] and Miller in [9] for the use in public-key cryptography (PKC). In order to introduce a public key cryptosystem based on elliptic curves, firstly we describe discrete logarithm problem, Diffie-Hellman problem, Diffie-Hellman key agreement and El-Gamal cryptosystem to discuss later in ECDLP. Properties of elliptic curves are discussed later.

4.1 Discrete Logarithm Problem

The discrete logarithm is the inverse of exponentiation in a finite cyclic group [10]. For a given cyclic group G with a group operation "*" and a generator " a", exponentiation in G is defined by

$$a^x = a * a * \ldots * a. \tag{4.1}$$

Suppose that $\beta = \alpha^x$, then the discrete logarithm of β is x and is written as

$$log_\alpha \beta = x. \tag{4.2}$$

Actually, the discrete logarithm of β is not unique as it can only be found modulo the order of α in $\mathbb{F}$. If α is a generator as specified above, then the logarithm is found modulo the order of the group

$$log_\alpha \beta = x \ (mod \ p). \tag{4.3}$$

where " p" is the group order.

Definition 4.1 : (*Discrete logarithm problem*) Given a prime p, a generator α of $\mathbb{Z}_p^*$ and an element $\beta \in \mathbb{Z}_p^*$, find the integer x, $0 \leq x \leq p-2$, such that $\beta = \alpha^x \ mod \ p$ [2]. The DLP in $\mathbb{Z}_p$ is considered to be difficult or intractable if p has at least 150 digits and p-1 has at least one large prime factor (as close to p as possible) [11]. These criteria for p are safeguards against the known attacks on DLP.

4.1.1 Diffie-Hellman Key Agreement Protocol

The problem of computing discrete logarithms was just a mathematical curiosity until Diffie and Hellman described a method of exchanging cryptographic keys which relies on DLP in 1976 [6]. The Diffie-Hellman key agreement protocol provides sharing secret key parts over an insecure channel between two parties, A and B, which is given in Figure 4.1 and works as follows:

1. A and B agree on group G and generator α. These choices can be public.
2. A chooses an exponent x ($0 \leq x \leq p-2$) randomly, computes α^x, and sends this value to the B. The exponent x must be kept private.
3. B chooses an exponent y ($0 \leq y \leq p-2$) randomly, computes α^y, and sends this value to A. The exponent y must be kept private. B then computes, using the value α^x received from A, $K_b=(\alpha^x)^y$.
4. When A receives α^y from B, A computes $K_a=(\alpha^y)^x$.

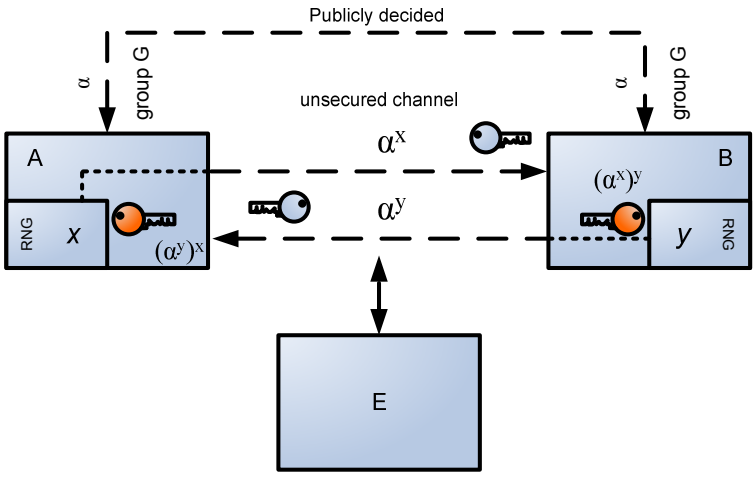

Figure 4.1: Diffie-Hellman Key Agreement Protocol

A and B now share the common secret key α^{xy}. If third party does not know any of the random choices, then DLP will keep the secret key α^{xy} in secure. An attacker could decrypt A's message if B's random secret key y could be computed from $\beta \equiv \alpha^y \ (mod\ p)$ and α which are publicly known [12].

In Figure 4.1, third part E could listen α^x, α^y $(mod\ p)$; the security of this protocol is based on the assumption of computing α^{xy}, common shared secret key, with these public values is as hard as obtaining the value y from $\beta \equiv \alpha^y$ $(mod\ p)$ in DLP. In brief, this protocol is secure as long as the DLP is intractable.

4.1.2 The El-Gamal Cryptosystem

The El-Gamal cryptosystem in $\mathbb{Z}_p^*$, which also uses discrete logarithm, is presented with the following equations, given in Figure 4.2 [13].

Let p be a prime such that the DLP in $\mathbb{Z}_p$ is intractable, and let $\alpha \in \mathbb{Z}_p^*$ be a primitive element, where p and α are publicly known. Each user creates their private keys, x, y and calculates α^x, α^y. After the calculation is completed, all are published to public as;

$$\beta_x \equiv \alpha^x\ (mod\ p),\ \beta_y \equiv \alpha^y\ (mod\ p). \tag{4.4}$$

where β is recipients published value.

Before, sending a message, user must choose a random number $k \in \mathbb{Z}_{p-1}$ and the message, $m \in \mathbb{Z}_p^*$, is sent as:

$$(s_1, s_2) = (\alpha^k\ mod\ p, m\beta_y^k\ mod\ p) \tag{4.5}$$

After, receiving the message, the recipient decrypts text as follows:

$$s_2(s_1^y)^{-1} \equiv m\beta^k(\alpha^{ky})^{-1} \equiv m\alpha^{yk}(\alpha^{ky})^{-1} \equiv m\ mod\ p, \tag{4.6}$$

where y is recipients secret key.

4.1.3 Elliptic Curve Discrete Logarithm Problem

The hardness of the elliptic curve discrete problem is essential for the security of all elliptic curve cryptographic systems.

Definition 4.2 : Given an elliptic curve E defined over a finite field $\mathbb{F}_q$, a point $P \in E(\mathbb{F}_q)$ of order n, and a point $Q \in < P >$, find the integer $k \in \{0,\ n-1\}$ such that $Q = kP$. The integer k is called the discrete logarithm of Q to the base P. ECDLP is defined to be the problem of finding the logarithm k for a given P and Q.

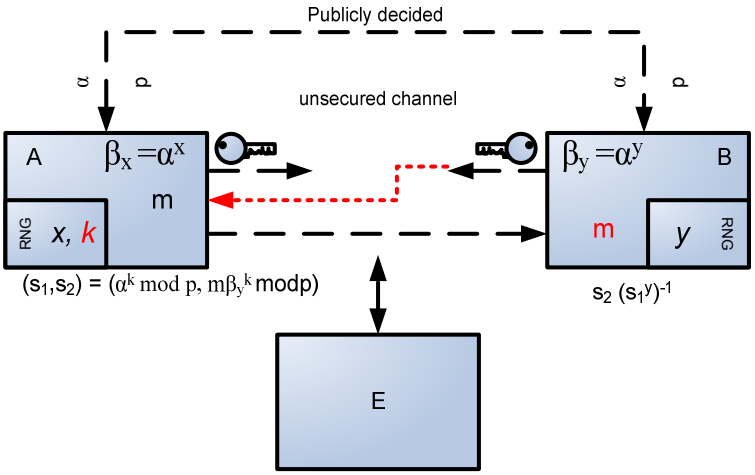

Figure 4.2: El-Gamal Cryptosystem

The number of rational points on a curve E over a finite field $\mathbb{F}_q$ is denoted by $\sharp E(\mathbb{F}_q)$. The ECDLP is really hard unless $\sharp E(\mathbb{F}_q)$ is "smooth", i.e., a product of small primes. This number is shown on the following theorem.

Theorem 4.1 :(*Hasse*) Let E be an elliptic curve over $\mathbb{F}_q$. Then

$$q+1-2\sqrt{q} \leq E(\mathbb{F}_q) \leq q+1+2\sqrt{q} \tag{4.7}$$

The quantity t, defined by $\sharp E(\mathbb{F}_q) = q+1-t$ is called *the trace of Frobenius* [14]. Hasse's theorem implies $|t| \leq 2\sqrt{q}$ [12].

The elliptic curve parameters should be carefully selected in order to resist all known attacks on ECDLP. If the estimated time of searching k is long enough to think about the worth of the information in secrecy, then attacker will give up attacking.

Some known attacks, their running times and precautions can be considered with following situations. Firstly, the most naive algorithm to solve the ECDLP is *exhaustive search* which is computing in every step *P, 2P, 3P* ... one by one until reaching Q value. The running time is approximately n steps in the worst case and $n/2$ steps on average. Therefore, this method can be circumvented by selecting elliptic curve parameters large enough to represent an infeasible amount of computation as $n \geq 2^{80}$. Secondly, there are lots of algorithms to attack ECDLP, but the most general known one is the combination of the Pohlig-Hellman algorithm and the Polard's rho algorithm [12], which has an exponential running time of $O(\sqrt{p})$ where p is the largest

prime divisor of n. If the elliptic curve parameters are chosen so that n is divisible by a prime number p sufficiently large, then it will be an infeasible amount of computation ($e.g., p > 2^{160}$), so ECDLP will resist to this kind of attack [12]. Finally, the important issue is choosing the parameters of elliptic curve very carefully, so that ECDLP could resist to all attacking method known.

In conclusion, as a comparison of the cryptosystems on security, the NESSIE consortium in [15], recommends sufficient security for the next 5-10 years, the use of 1536-bit keys for RSA and DL based public key schemes, and 160-bit for elliptic curve discrete logarithms. This recommendation is based on an assumed equivalence between 512-bit RSA keys and 56-bit keys, and an extrapolation of that is given in Table 4.1.

Table 4.1: NESSIE Recommendations

Equivalent symmetric key size	56	64	80	112	128	160
Elliptic curve key size	112	128	160	224	256	320
Modulus length (pq)	512	768	1536	4096	6000	10000
Modulus length (p^2q)	570	800	1536	4096	6000	10000

4.2 Introduction to Elliptic Curves

Elliptic curve cryptography is a public-key cryptosystem which is believed to be intractable because of hardness of finding discrete logarithm in a finite group. In public-key cryptography, for example the RSA algorithm, the product of two large prime numbers are used as the puzzle: a user picks two large random primes as private key, and publish their product as public key. While finding large primes and multiplying them is easy, its inverse process factoring is believed to be hard. But, improvements on technology lead to longer bits to provide intractability. It is generally recommended RSA public keys to be at least 1024 bits in length to render integer factoring algorithms infeasible [16]. On the other hand, for given P and Q, finding k such that $kP = Q$ in elliptic curve needs less bits to provide intractability. The size of group determines the difficulty of the problem. It is believed that smaller group can be used to obtain the same level of security as RSA-based systems. When RSA is compared with ECC, ECC needs shorter parameters and signatures, ECC is faster than RSA on some platforms and needs lower power consumption [17].

Now, we discuss the properties of elliptic curves with the equation 4.8.

4.2.1 Weierstrass Equation

Let K be a field. For example, K can be the finite field of $\mathbb{F}_q$, the prime field $\mathbb{Z}_p$, the field $\mathbb{R}$ of the real numbers, the filed $\mathbb{Q}$ of rational numbers, or the field $\mathbb{C}$ of complex numbers [13].

An elliptic curve over a field K is defined by the Weierstrass equation:

$$y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6 \tag{4.8}$$

over this field and the point $\mathscr{O}$ at infinity, where a_1, a_2, a_3, a_4, $a_6 \in K$. The elliptic curve E over K is denoted *E(K)*.

All the solutions of the above equation together with a point at infinity form an Abelian group, with the point at infinity as identity element. If the coordinates x and y are chosen from a finite field, the solutions form a finite Abelian group.

For fields of various characteristics, the Weierstrass equation can be transformed into different forms by a linear change of variables [13]. For instance,

Characteristic $\neq 2,3$ **:** Let K be a field of characteristics $\neq 2,3$, and let $x^3 + ax + b$ (where $a,b \in K$) be a cubic polynomial with the condition that $4a^3 + 27b^2 \neq 0$ which ensures that the polynomial has no multiple roots. An elliptic curve over K is the set of points (x,y) with $x,y \in K$ that satisfy the equation.

$$y^2 = x^3 + ax + b \tag{4.9}$$

and the element denoted by $\mathscr{O}$ is called the point at infinity.

Characteristic 2 **:** If K is a field of characteristic 2, then there are two types of elliptic curves:

An elliptic curve of zero j-invariant is the set of points satisfying

$$y^2 + a_3y = x^3 + a_4x + a_6 \tag{4.10}$$

(where $a_3, a_4, a_6 \in \mathbb{F}_q, a_3 \neq 0$) and $\mathscr{O}$, the point at infinity.

j-invariant of E over K is an element of K determined by a_1, a_2, a_3, a_4 and a_6 [18].

An elliptic curve of nonzero j-invariant is the set of points satisfying

$$y^2 + xy = x^3 + a_2x^2 + a_6 \tag{4.11}$$

(where $a_2, a_6 \in \mathbb{F}_q, a_6 \neq 0$) and $\mathcal{O}$, the point at infinity.

4.2.2 Point Addition over Finite Fields

Let P_1 and P_2 be two points on an elliptic curve E and we define a third point $P_1 + P_2$ so that *E(K)* defines an abelian group with this addition operation. If $P_1 \neq P_2$, then the line which goes through P_1 and P_2 intersects the curve on a third point Q. If $P_1 = P_2$ then the tangent of *E(K)* at P_1 intersects the curve on a second point Q. In every group structure, there must be a neutral element with respect to the group operation, so this line and Q does not define a group structure in this condition. Therefore, we find a point of intersection where the curve meets the line connecting Q and the point infinity (neutral element) with a third point which we call this point $P_1 + P_2$ or $2P_1$. This situation can be provided by a vertical line, which is drawn through the point Q. A vertical line intersects *E(K)* at 3 points: (x, y),$(x, -y)$ an 0. Hence, the point at infinity 0 serves as the additive identity element, other two points are their inverses in addition. $P_1 + P_2 + Q = 0$ or $P_1 + P_2 = -Q$, the inverse of Q. In figure 4.3 and 4.4, addition in different points and doubling in one point is illustrated respectively. These elliptic curves are drawn over real numbers with the equation $y^2 = x^3 - 50x + 100$.

Given two points $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$, $P_1 \neq P_2$, the sum $P_3 = P_1 + P_2 = (x_3, y_3)$ can be computed as;

$$\lambda = \begin{cases} \frac{y_1 - y_2}{x_1 - x_2} & P_1 \neq P_2 \\ \frac{3x_1^2 + 2a_2x_1 + a_4 - a_1y_1}{2y_1 + a_1x_1 + a_3} & P_1 = P_2 \end{cases} \tag{4.12}$$

$$x_3 = \lambda^2 - a_1\lambda - a_2 - x_1 - x_2,\ y3 = (x_1 - x_3)\lambda - y_1 - a_1x_3 - a_3 \tag{4.13}$$

In general, the basic operation for ECC algorithms is point or scalar multiplication, shown as $Q = kP$, where k is an integer, P and Q are EC points. The efficiency of point multiplication is mainly determined by the implementation of the finite field arithmetic. The point operation can be calculated in many different ways, for example by using two different double-and-add algorithm and Montgomery ladder algorithm which are executed by point addition and doubling. Algorithms are given in Section 4.3. The

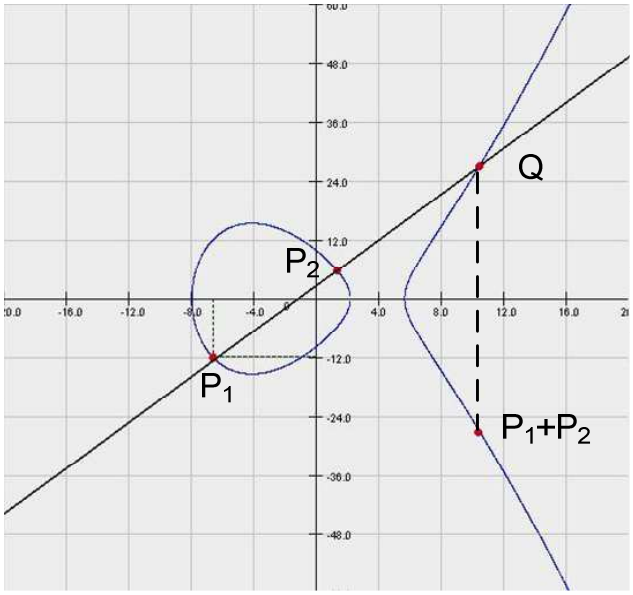

Figure 4.3: Point Addition of a Point in Elliptic Curve Equation $y^2 = x^3 - 50x + 100$

lowest hierarchical level is composed of finite filed operations: addition, subtraction, multiplication and inversion.

There are many types of coordinates in which an elliptic curve can be represented. In the above equations affine coordinates are used, but so-called *projective coordinates* have some implementation advantages. The main conclusion is that point addition can be done in projective coordinates using only field multiplications, with no inversions required. In addition to this, inversion is only needed ones, at the end of the point multiplication operation, to convert back to affine coordinates.

4.3 Point Multiplication

In this section, we consider the methods of computing kP, where k is an integer and P is a point on elliptic curve E defined over $\mathbb{F}_q$. This operation is called point multiplication, and it consumes almost all of the execution time on elliptic curve cryptographic protocols. Basically, $Q = k.P$ is calculated by adding the point P to itself k times. Algorithm 1 and 2 are the basic repeated double-and-add methods which process the bits of k from right to left and left to right, respectively. Algorithm 3 is Montgomery ladder which is computationally balanced and independent of k_i , thus

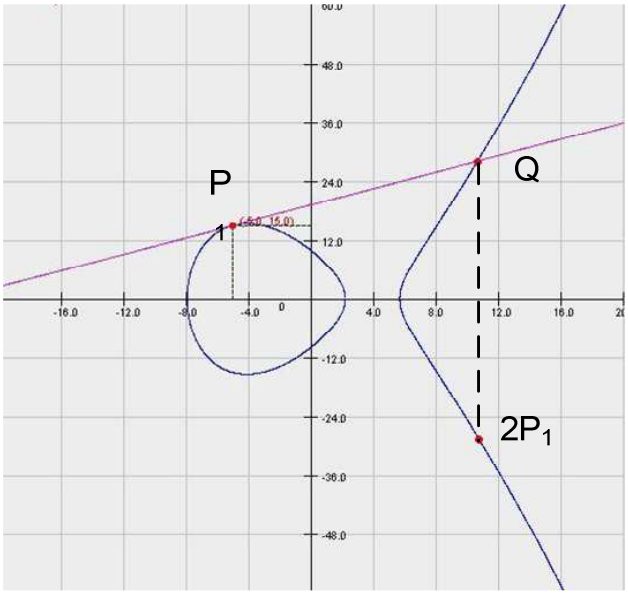

Figure 4.4: Doubling of a Point in Elliptic Curve Equation $y^2 = x^3 - 50x + 100$

it is more secure against simple power analysis (SPA). It will be discussed in Section 6.1.1.4 in details.

After the method of point multiplication is chosen, one lower level of hierarchy is selecting point addition and point doubling algorithms. These algorithms use finite field arithmetic: addition, subtraction, multiplication and inversion, with respect to the control of the top level. The hierarchy of a basic elliptic curve cryptosystem is illustrated in Figure 4.5.

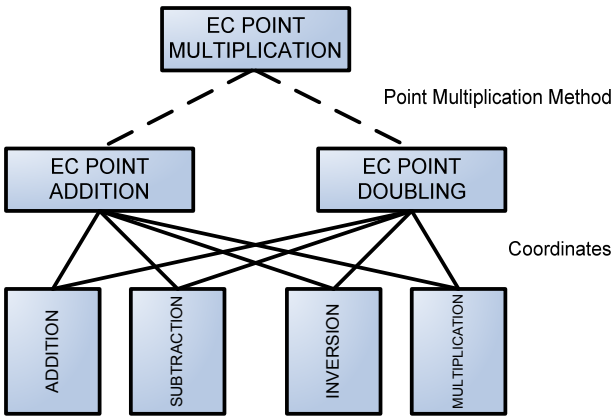

Figure 4.5: Hierarchy of Elliptic Curve Cryptosystems

Algorithm 1 : Right-to-left Binary Method for point multiplication [12]

Require: EC point $P = (x, y)$, integer k, $0 < k < M$,
$k = (k_{t-1}, k_{t-2}, \ldots, k_0)_2$, $P \in E(\mathbb{F}_q)$
Ensure: $Q = [k]P$
$Q \leftarrow \infty$
for i *from* 0 *to* $t-1$ **do**
if $k_i = 1$ **then**
$Q \leftarrow Q + P$
end if
$P \leftarrow 2P$
end for
return(Q)

Algorithm 2 : Left-to-right Binary Method for point multiplication [12]

Require: EC point $P = (x, y)$, integer k, $0 < k < M$,
$k = (k_{t-1}, k_{t-2}, \ldots, k_0)_2$, $P \in E(\mathbb{F}_q)$
Ensure: $Q = [k]P$
$Q \leftarrow \infty$
for i *from* $t-1$ *downto* 0 **do**
$Q \leftarrow 2Q$
if $k_i = 1$ **then**
$Q \leftarrow Q + P$
end if
end for
return(Q)

Algorithm 3 : Montgomery Ladder for point multiplication [12]

Require: EC point $P = (x, y)$, integer k, $0 < k < M$,
$k = (k_{t-1}, k_{t-2}, \ldots, k_0)_2$, $k_{t-1} = 1$ $P \in E(\mathbb{F}_q)$
Ensure: $Q = [k]P$
$P_1 \leftarrow P$, $P_2 \leftarrow 2P$
for i *from* $t-2$ *downto* 0 **do**
if $k_i = 1$ **then**
$P_1 \leftarrow P_1 + P_2$, $P_2 \leftarrow 2P_2$
else
$P_2 \leftarrow P_1 + P_2$, $P_1 \leftarrow 2P_1$
end if
end for
return(P_1)

In brief, an elliptic curve cryptography can be implemented either faster or more secure up to the choice of the proper methods for application. Moreover, with respect to the finite field choice, for example characteristic 2 as ours, addition and subtraction are identical, and are accomplished using the XOR operation. Furthermore, inversion can be neglected by using projective coordinates and only used once at the end of the point multiplication. Specific work on elliptic curve algorithms can be found in [19].

4.4 Projective Coordinates

Formulas for adding two points on an elliptic curve were presented in Section 4.2. For all curves defined, the formulas for point addition and point doubling require inversions, multiplications and additions. If inversion in K consumes much more time and power than multiplication, then using projective coordinate representation, to reduce the number of inversion to one, can be more advantageous. The following sections consider two different projective coordinates in brief.

Let K be a field, and let c and d be positive integers. One can define an equivalence relation $\sim$ on the set $K^3(0,0,0)$ of nonzero triples over K by $(X_1,Y_1,Z_1) \sim (X_2,Y_2,Z_2)$ if $X_1 = \lambda^c X_2$, $Y_1 = \lambda^d Y_2$, $Z_1 = \lambda Z_2$ for some $\lambda \in K^*$.

The equivalence class containing $(X,Y,Z) \in K^3(0,0,0)$ is

$$(X : Y : Z) = (\lambda^c X, \lambda^d Y, \lambda Z) : \lambda \in K^*. \tag{4.14}$$

$(X : Y : Z)$ is called a projective point, and (X,Y,Z) is called a representative of $(X : Y : Z)$. The projective of Weierstrass equation (4.8) of an elliptic curve E defined over K is obtained by replacing x by X/Z^c and y by Y/Z^d, and clearing denominators [12].

4.4.1 Standard Projective Coordinates

Let $c = 1$ and $d = 1$. Then the projective form of the Weierstrass equation

$$E : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6 \tag{4.15}$$

defines over K is

$$Y^2Z + a_1XYZ + a_3YZ^2 = X^3 + a_2X^2Z + a_4XZ^2 + a_6Z^3 \tag{4.16}$$

The only point on the line at infinity that lies on E is $(0:1:0)$ [12]. This projective point corresponds to the point $\mathscr{O}$ in Equation 4.8.

4.4.2 Jacobian Coordinates

Let $c = 2$ and $d = 3$. The projective point $(X:Y:Z),\ Z \neq 0$, corresponds to the affine point $(X/Z^2, Y/Z^3)$ [12]. The projective form of the Weierstrass equation

$$E : y^2 = x^3 + ax + b \tag{4.17}$$

defines over K is

$$Y^2 = X^3 + aXZ^4 + bZ^6 \tag{4.18}$$

The point at infinity $\mathscr{O}$ corresponds to $(1:1:0)$, while the negative of $(X:Y:Z)$ is $(X:-Y:Z)$ [12].

5. EDWARDS CURVES

The main operations in the elliptic curve cryptography are single-scalar multiplication $(k, P \rightarrow kP)$ and double-scalar multiplication $(k, l, P, Q \rightarrow mP + nQ)$. For instance, Miller proposed carrying these points in Jacobian coordinates, so each point is represented by three values (x, y, z) which corresponds $(x/z^2, y/z^3)$ on a curve $y^2 = x^3 + a_4x + a_6$ [9]. Up to now, the fastest algorithm for point addition uses 16 field multiplications, specifically 11M+5S. Studies on getting faster addition and doubling on elliptic curves are going on [20].

A new form for elliptic curves was added to the mathematical literature with Edwards curves. Edwards showed in [21] that all elliptic curves over number fields can be transformed to $x^2 + y^2 = c^2(1 + x^2y^2)$, with $(0, c)$ as the neutral element and with a simple and a symmetric addition law.

$$(x_1, y_1), (x_2, y_2) \rightarrow \left(\frac{x_1y_2 + y_1x_2}{c(1 + x_1x_2y_1y_2)}, \frac{y_1y_2 - x_1x_2}{c(1 - x_1x_2y_1y_2)}\right). \tag{5.1}$$

Similarly, all elliptic curve equations can be converted to the Edwards form. Some of them require field extensions, but mostly these are used transformations which are defined over the original number field or the finite field. Moreover, in [20] the notation of Edwards form is expanded to include all curves $x^2 + y^2 = c^2(1 + dx^2y^2)$, where $cd(1 - dc^4) \neq 0$, so that it is possible to capture a larger class of elliptic curves over the original field.

In brief, Edwards form breaks the Jacobian speed barrier stated before and is the new speed leader for multi-scalar multiplication. In addition to these, Edwards curve has an extra feature that the addition formulas are *complete*. This means that the formulas work over all point pairs on the curve with no exceptions for doubling, neutral element, negatives, etc [20]. The following section discusses completeness of Edwards curves over the characteristic 2, which are called binary Edwards curves. By introducing this curve, the advantages of binary field over hardware implementations can be available.

In section 6, the implementation of binary Edwards curves for RFID tags will be shown.

5.1 Binary Edwards Curves

This section contains complete addition formulas for binary elliptic curves, i.e., addition formulas that work for all input pairs, with no exceptional cases. First, the need for Edwards curves is explained, and then the theorems and formulas will be shown in order.

5.1.1 Introduction to Binary Edwards Curves

The points on a Weierstrass-form elliptic curve

$$y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6 \tag{5.2}$$

include not only the affine point (x_1, y_1), but also an extra point at infinity serving as neutral element. The standard formulas for elliptic curve to compute a sum $P_1 + P_2$ fail if P_1, P_2, or $P_1 + P_2$ is at infinity, or if P_1 is equal to P_2. Each of these possibilities should be tested separately before generating any elliptic curve cryptosystem. A complete addition algorithm is produced by combining several incomplete addition formulas.

In [10], the new curve shape for ordinary elliptic curves over field of characteristic 2 is introduced, and is shown that the affine points are non-singular. Binary Edwards curves properties are defined in Weierstrass form in Definition 5.1.

Definition 5.1 : (*Binary Edwards Curve*) Let k be a field with $char(k) = 2$. Let d_1, d_2 be elements of k with $d_1 \neq 0$ and $d_2 \neq d_1^2 + d_1$, then the binary Edwards curve with coefficients d_1 and d_2 is the affine curve [22]:

$$E_{B,d_1,d_2} = d_1(x+y) + d_2(x^2+y^2) = xy + xy(x+y) + x^2y^2 \tag{5.3}$$

This curve is symmetric in x and y and thus it has the property that if (x_1, y_1) is a point on the curve then so is (y_1, x_1). The point $(0,0)$ will be the neutral element of the addition law, while $(1,1)$ will have order 2 [22].
The non-singularity of each binary Edwards curve is proven in Theorem 5.1.

Theorem 5.1(Non-singularity). *Each binary Edwards curve is non-singular [22].*

Proof. By definition, the curve E_{B,d_1,d_2} has $d_1 \neq 0$ and $d_2 \neq d_1^2 + d_1$. The partial

derivatives of the curve equation are $d_1 + y + y^2$ and $d_1 + x + x^2$. A singular point (x_1, y_1) must have $d_1 + y_1 + y_1^2 = 0$ and $d_1 + x_1 + x_1^2 = 0$ and therefore $(x_1 + y_1)^2 = x_1 + y_1$, implying $x_1 = y_1$ or $x_1 = y_1 + 1$.

The case $x_1 = y_1$ implies $0 = x_1^2 + x_1^4$ by the curve equation and therefore $d_1^2 = x_1^2 + x_1^4 = 0$, contradicting the hypothesis that $d_1 \neq 0$.

The case $x_1 = y_1 + 1$ implies $d_1 + d_2 = y_1^2 + y_1^4$ by the curve equation and therefore $d_1^2 = y_1^2 + y_1^4 = d_1 + d_2$, which contradicts the hypothesis that $d_2 \neq d_1^2 + d_1$.

5.1.2 Binary Edwards Curves Addition Law

Binary Edwards curves, E_{B,d_1,d_2}, addition law is given as in follows, and it is proven that the addition law corresponds to the elliptic curve in Weierstrass form similarly. It can be used for doubling with two identical inputs. The sum of two points (x_1, y_1), (x_2, y_2) on E_{B,d_1,d_2} is the point (x_3, y_3) defined as follows:

$$x_3 = \frac{d_1(x_1 + x_2) + d_2(x_1 + y_1)(x_2 + y_2) + (x_1 + x_1^2)(x_2(y_1 + y_2 + 1) + y_1 y_2)}{d_1 + (x_1 + x_1^2)(x_2 + y_2)}, \quad (5.4)$$

$$y_3 = \frac{d_1(y_1 + y_2) + d_2(x_1 + y_1)(x_2 + y_2) + (y_1 + y_1^2)(y_2(x_1 + x_2 + 1) + x_1 x_2)}{d_1 + (y_1 + y_1^2)(x_2 + y_2)}. \quad (5.5)$$

If the denominators $d_1 + (x_1 + x_1^2)(x_2 + y_2)$ and $d_1 + (y_1 + y_1^2)(x_2 + y_2)$ are non-zero then the sum (x_3, y_3) is a point on E_{B,d_1,d_2}: i.e., $d_1(x_3 + y_3) + d_2(x_3^2 + y_3^2) = x_3.y_3 + x_3.y_3(x_3 + y_3) + x_3^2.y_3^2$ [22].

Here, if the points are inserted like $(0,0)$ into the addition law, it is shown that $(0,0)$ is the neutral element. Similarly, $(x_1, y_1) + (1,1) = (x_1 + 1, y_1 + 1)$; in particular $(1,1) + (1,1) = (0,0)$. Furthermore $(x_1, y_1) + (y_1, x_1) = (0,0)$, so $-(x_1, y_1) = (y_1, x_1)$ [22].

5.1.3 Complete Binary Edwards Curves

The complete binary Edwards curves conditions and requirements are given in Definition 5.2.

Definition 5.2 : Let k be a field with $char(k) = 2$. Let d_1, d_2 be elements of k with $d_1 \neq 0$. Assume that no element $t \in k$ satisfies $t^2 + t + d_2 = 0$. Then the addition law on the binary Edwards curve $E_{B,d_1,d_2}(k)$ is complete, then the complete binary Edwards curves with coefficients d_1 and d_2 is the affine curve [22].

$$E_{B,d_1,d_2} = d_1(x + y) + d_2(x^2 + y^2) = xy + xy(x + y) + x^2 y^2. \quad (5.6)$$

There is no conflict in notation or terminology, and no difference from binary Edwards curve E_{B,d_1,d_2}. The complete case has the extra requirement that $t^2+t+d_2 \neq 0$ for all $t \in k$, not just for $t = d_1$. If k is the finite field F_{2^n} then an equivalent requirement is that $T_r(d_2) = 1$, where T_r is the absolute trace of F_{2^n} over F_2. In [10], more information is given about generality of E_{B,d_1,d_2}.

5.1.4 Explicit Addition Formulas

In this section, we present explicit formulas for affine addition, projective addition and mixed addition on the binary Edwards curves. The formulas are not as fast as Weierstrass equation; on the other hand curves have the advantage of being unified and completeness for suitable $T_r(d_2)$ values [22].

5.1.4.1 Affine Addition

The following formulas, given (x_1, y_1) and (x_2, y_2) on the binary Edwards curve E_{B,d_1,d_2}, compute the sum $(x_3, y_3) = (x_1, y_1) + (x_2, y_2)$ if it is defined:

Algorithm 4 : Affine Addition

$w_1 = x_1 + y_1$,
$w_2 = x_2 + y_2$,
$A = x_1^2 + x_1$,
$B = y_1^2 + y_1$,
$C = d_2 w_1 w_2$,
$D = x_2 y_2$,
$x_3 = y_1 + (C + d_1(w_1 + x_2) + A(D + x_2))/(d_1 + Aw_2)$,
$y_3 = x_1 + (C + d_1(w_1 + y_2) + B(D + y_2))/(d_1 + Bw_2)$.

These formulas use $2I + 8M + 2S + 3D$, where I is the cost of inversion, M is the cost of multiplication, S is the cost of squaring, D is the cost of a multiplication by a curve parameter. The 3D here are two multiplications by d_1 and one multiplication by d_2 [22].

For complete binary Edwards curves the denominators $(d_1 + A.w_2) = d_1 + (x_1^2 + x_1)(x_2 + y_2)$ and $(d_1 + B.w_2) = d_1 + (y_1^2 + y_1)(x_2 + y_2)$ cannot be zero.

5.1.4.2 Mixed Addition

Given $(X_1 : Y_1 : Z_1)$ and (x_2, y_2) on the binary Edwards curve E_{B,d_1,d_2}, the following formulas compute the sum $(X_3 : Y_3 : Z_3) = (X_1 : Y_1 : Z_1) + (x_2, y_2)$ if it is defined: Note

Algorithm 5 : Mixed Addition

$W_1 = X_1 + Y_1$,
$w_2 = x_2 + y_2$,
$A = x_2^2 + x_2$,
$B = y_2^2 + y_2$,
$D = W_1.Z_1$,
$E = d_1.Z_1^2$,
$H = (E + d_2 D).w_2$,
$I = d_1.Z_1$,
$U = E + A.D$,
$V = E + B.D$,
$Z_3 = U.V$,
$X_3 = Z_3.y_2 + (H + X_1(I + A(Y_1 + Z_1))).V$,
$Y_3 = Z_3.x_2 + (H + Y_1(I + B(X_1 + Z_1))).U$.

that, these formulas use $13M + 3S + 3D$. For complete binary Edwards curves the product $Z_3 = Z_1^4(d_1 + (x_2^2 + x_2)(x_1 + y_1))(d_1 + (y_2^2 + y_2)(x_1 + y_1))$ cannot be zero [22].

5.1.4.3 Projective Addition

The following formulas, given $(X_1 : Y_1 : Z_1)$ and $(X_2 : Y_2 : Z_2)$ on the binary Edwards curve E_{B,d_1,d_2}, compute the sum $(X_3 : Y_3 : Z_3) = (X_1 : Y_1 : Z_1) + (X_2 : Y_2 : Z_2)$.

Algorithm 6 : Projective Addition I

$W_1 = X_1 + Y_1$,
$W_2 = X_2 + Y_2$,
$A = X_1.(X_1 + Z_1)$,
$B = Y_1.(Y_1 + Z_1)$,
$C = Z_1.Z_2$,
$D = W_2.Z_2$,
$E = d_1.C.C$,
$H = (d_1 Z_2 + d_2 W_2).W_1.C$,
$I = d_1.C.Z_1$,
$U = E + A.D$,
$V = E + B.D$,
$S = U.V$,
$X_3 = S.Y_1 + (H + X_2(I + A(Y_2 + Z_2))).V.Z_1$,
$Y_3 = S.X_1 + (H + Y_2(I + B(X_2 + Z_2))).U.Z_1$,
$Z_3 = S.Z_1$.

These formulas use $21M + 1S + 4D$. The 4D are three multiplications by d_1 and one multiplication by d_2. For complete binary Edwards curves $Z_3 = Z_1^5.Z_2^4(d_1 + (x_2^2 + x_2)(x_1 + y_1))(d_1 + (y_2^2 + y_2)(x_1 + y_1))$ cannot be zero. Note that, these formulas are going to be used to implement binary Edwards curves in projective coordinate and will

be discussed in Section 6 in details. The constant values can be more general than the following projective addition formulas, thus we used these formulas in our design [22].

The following formulas are given for small d_1 and d_2 values, that they are faster than previous one:

Algorithm 7 : Projective Addition II

$A = X_1.X_2$,
$B = Y_1.Y_2$,
$C = Z_1.Z_2$,
$D = d_1.C$,
$E = C^2$,
$F = d_1^2.E$,
$G = (X_1 + Z_1).(X_2 + Z_2)$,
$H = (Y_1 + Z_1).(Y_2 + Z_2)$,
$I = A + G$,
$J = B + H$,
$K = (X_1 + Y_1).(X_2 + Y_2)$,
$U = C.(F + d_1K.(K + I + J + C))$,
$V = U + D.F + K.(d_2(d_1E + G.H + A.B) + (d_2 + d_1)I.J)$
$X_3 = V + D.(A + D).(G + D)$,
$Y_3 = V + D.(B + D).(H + D)$,
$Z_3 = U + (d_2 + d_1)C.K^2$.

These formulas use $18M + 2S + 7D$. One can alternatively compute F as D^2, replacing $1D$ with $1S$. For the complete binary Edwards curves the denominator Z_3 cannot be zero [22].

The following formulas become simpler in case $d_1 = d_2$:

These formulas use $16M + 1S + 4D$. As stated above, one can replace $1D$ with $1S$. For complete binary Edwards curves the denominator Z_3 cannot be zero.

5.1.5 Doubling

The fast doubling formulas on the Edwards curve E_{B,d_1,d_2} is presented in this section. Affine coordinates and inversion-free projective coordinates are given respectively. In addition to these, the formulas are complete if the curve is complete. The literature on doubling formulas for binary Edwards curves is reviewed and the speeds of two different doubling forms are compared in this section.

Algorithm 8 : Projective Addition III

$A = X_1.X_2$,
$B = Y_1.Y_2$,
$C = Z_1.Z_2$,
$D = d_1.C$,
$E = C^2$,
$F = d_1^2.E$,
$G = (X_1 + Z_1).(X_2 + Z_2)$,
$H = (Y_1 + Z_1).(Y_2 + Z_2)$,
$I = A + G$,
$J = B + H$,
$K = (X_1 + Y_1).(X_2 + Y_2)$,
$L = d_1.K$,
$U = C.(F + L.(K + I + J + C))$,
$V = U + D.F + L.(d_1E + G.H + A.B)$,
$X_3 = V + D.(A + D).(G + D)$,
$Y_3 = V + D.(B + D).(H + D)$,
$Z_3 = U$.

5.1.5.1 Affine Doubling

Let (x_1, y_1) be a point on E_{B,d_1,d_2}, and assume that the sum $(x_1, y_1) + (x_1, y_1)$ is defined. Computing $(x_3, y_3) = (x_1, y_1) + (x_1, y_1)$ we obtain;

$$\begin{aligned} x_3 &= \frac{d_2(x_1 + y_1)^2 + (x_1 + x_1^2)(x_1 + y_1^2)}{d_1 + (x_1 + y_1)(x_1 + x_1^2)} \\ &= \frac{d_1(x_1 + y_1) + x_1y_1 + x_1^2(1 + x_1 + y_1)}{d_1 + x_1y_1 + x_1^2(1 + x_1 + y_1)} \\ &= 1 + \frac{d_1(1 + x_1 + y_1)}{d_1 + x_1y_1 + y_1^2(1 + x_1 + y_1)}, \end{aligned} \tag{5.7}$$

where the second line uses that $d_2(x_1 + y_1)^2 + x_1^2y_1^2 + x_1y_1^2 = d_1(x_1 + y_1) + x_1y_1 + x_1^2y_1$ for all points on E_{B,d_1,d_2} [22]. Likewise we have

$$y_3 = 1 + \frac{d_1(1 + x_1 + y_1)}{d_1 + x_1y_1 + y_1^2(1 + x_1 + y_1)}. \tag{5.8}$$

Note that, the affine formulas is computed with one inversion, as the product of the denominators of x_3 and y_3 is

$$
\begin{aligned}
&(d_1 + x_1y_1 + x_1^2(1+x_1+y_1))(d_1 + x_1y_1 + y_1^2(1+x_1+y_1)) \\
&\quad = d_1^2 + (x_1^2+y_1^2)(d_1(1+x_1+y_1) + x_1y_1(1+x_1+y_1) + x_1^2y_1^2) \\
&\quad = d_1^2 + (x_1^2+y_1^2)(d_1 + d_2(x_1^2+y_1^2)) \\
&\quad = d_1(d_1 + x_1^2 + y_1^2 + (d_2/d_1)(x_1^4+y_1^4)),
\end{aligned} \tag{5.9}
$$

where the curve equation is used again. This leads to the doubling formulas

$$x_3 = 1 + \frac{d_1 + d_2(x_1^2+y_1^2) + y_1^2 + y_1^4}{d_1 + x_1^2 + y_1^2 + (d_2/d_1)(x_1^4+y_1^4)}, \tag{5.10}$$

$$y_3 = 1 + \frac{d_1 + d_2(x_1^2+y_1^2) + x_1^2 + x_1^4}{d_1 + x_1^2 + y_1^2 + (d_2/d_1)(x_1^4+y_1^4)}, \tag{5.11}$$

which needs $1I+2M+4S+2D$. For complete binary Edwards curves all denominators here are nonzero [22].

If $d_1 = d_2$ some multiplications can be grouped as follows:
$A = x_1^2, \quad B = A^2, \quad C = y_1^2, \quad D = C^2, \quad E = A + C,$
$F = 1/(d_1 + E + B + D), \quad x_3 = (d_1E + A + B).F, \quad y_3 = x_3 + 1 + d_1F.$
These formulas use only $1I + 1M + 4S + 2D$.

5.1.5.2 Projective Doubling

In this sub-section, explicit formulas of projective doubling is given to compute $2(X_1 : Y_1 : Z_1) = (X_3 : Y_3 : Z_3)$:
$A = X_1^2, \quad B = A^2, \quad C = Y_1^2, \quad D = C^2, \quad E = Z_1^2,$
$F = d_1E^2, \quad G = (d_2/d_1)(B+D), \quad H = A.E,$
$I = C.E, \quad J = H + I, \quad K = G + d_2J,$
$X_3 = K + H + D, \quad Y_3 = K + I + B, \quad Z_3 = F + J + G.$

These formulas use $2M + 6S + 3D$. The $3D$ are multiplications by $d_1, d_2/d_1$ and d_2. For complete binary Edwards curves the denominator Z_3 is nonzero.

If $d_1 = d_2$ the squaring can be computed as follows :
$W_1 = X_1 + Y_1, E + (W_1(W_1 + Z_1))^2,$
$X_3 = ((\sqrt{d_1}W_1 + X_1)Z_1 + X_1^2)^2, Y_3 = X_3 + E, Z_3 = E + d_1(Z_1^2)^2.$

These formulas use $2M + 5S + 2D$. For complete binary Edwards curves the denominator Z_3 is nonzero [22].

Comparing the literature vs. binary Edwards curves:

These doubling formulas for complete Edwards curves are the first complete doubling formulas in literature. All other doubling formulas in the literature have exceptional cases. Moreover, it is presented in [22] that there are two improvements on doubling formulas of Lopez-Dahab coordinates for binary curves in Weierstrass form. Also the improved formulas against Kim and Kim represented formulas is given in [23] and [22].

5.1.6 Differential Addition

"Differential addition" means computing $Q+P$ given $Q, P, Q - P$: e.g., computing $(2m+1)P$ given $(m+1)P$, mP and P. In [22], it is analyzed that the cost of formulas in affine coordinates is expensive, if the inversion operation is expensive. Thus, unless there is limit to storage space, it is better to represent the points in projective form (i.e., as a ratio of two elements). In Table 5.1, the achieved speeds in [22] is given.

Table 5.1: The Speed of Binary Edwards Curves Differential Addition

	General case	$d_2 = d_1$
Affine diff addition	1I + 3M + 1S + 1D	1I + 1M + 2S + 1D
Affine diff addition + doubling	2I + 4M + 3S + 2D	2I + 1M + 3S + 2D
Mixed diff addition	6M + 1S + 2D	5M + 1S + 1D
Mixed diff addition + doubling	6M + 4S + 4D	5M + 4S + 2D
Projective diff addition	8M + 1S + 2D	7M + 1S + 1D
Projective diff addition + doubling	8M + 4S + 4D	7M + 4S + 2D

The reason, why the differential addition is interesting, is relied on Montgomery's fast formulas for u-coordinate differential addition in non-binary elliptic curves presentation $v^2 = u^3 + a_2.u^2 + u$ in [24]. An application, Montgomery ladder is suggested to compute $u(mP)$, $u((m+1)P)$ given $u(P)$. It is mentioned in Section 4 that the Montgomery ladder has many advantages like: it is fast; controller part fits into extremely small hardware; its uniform double-and-add structure makes it secure against simple side-channel attacks. Therefore, it is used in our design to protect against SPA and also projective addition formulas are used to implement EC point addition. More details related to differential addition can be found in [22].

6. BINARY EDWARDS CURVES IMPLEMENTATION

Previous sections show that the communication over unsecured channels is an extremely hard problem. Symmetric-key algorithms can be used to generate highly secure and fast systems. As mentioned previously, the drawback of symmetric cryptography is key management and distribution before getting into secure channel to communicate. To solve this problem, public-key cryptography was proposed to arrange key management. If public-key and symmetric-key cryptography are used together, then we can provide fast, more secure and efficient systems. Using elliptic curves is one of the recent methods to create protocols for the key management. Moreover, several algorithms are proposed to improve the features of elliptic curves. Recently, Edwards curves were proposed [21] and it was shown that every point on curve is valid for point addition. Afterwards, binary Edwards curves was proposed.

In this work, a binary Edwards curve implementation has been designed. This is the first implementation of a binary Edwards curve on hardware, and the proposed design is compact over finite fields, so that it can be used for RFID's. In elliptic curve algorithms most of the calculations is concentrated on point multiplication ($Q = k.P$), then different protocols can be implemented on it easily. The proposed design was written in GEZEL hardware design language [25], and its results are tested with Synopsys Design Vision [26]. Moreover, the projective coordinates are used to neglect inversion during finite fields computation, and also modular addition is used instead of doubling. Therefore, the design is more secure against Simple Power Analyses (SPA) [27]. Register number of the design is reduced from 8 to 5 to consume less area, and several hardware tricks are used to finish the calculations in less clock cycles.

6.1 Implementation of Binary Edwards Curves

As an implementation step, first of all, the strategy of design is arranged. Control block and the finite state machine of the point multiplication are the critical parts of

the design. Therefore, projective addition algorithm steps, given in algorithm 9, are arranged to calculate the results in an efficient way. The point multiplication is tested to see that algorithm is working for $k = 5$ (in binary 101) with respect to test vectors. After the reference finite state machine is provided with 8 registers in register file, different finite state machines are written to reduce the number of registers. Finally, register file is established with five registers.

Algorithm 9 : Projective Addition

$W_1 = X_1 + Y_1, \quad \Longrightarrow W_2 = X_2 + Y_2,$
$W_2 = X_2 + Y_2, \quad \Longrightarrow D = W_2.Z_2,$
$A = X_1.(X_1 + Z_1), \quad \Longrightarrow C = Z_1.Z_2,$
$B = Y_1.(Y_1 + Z_1), \quad \Longrightarrow E = d_1.C.C,$
$C = Z_1.Z_2, \quad \Longrightarrow I = d_1.C.Z_1,$
$D = W_2.Z_2, \quad \Longrightarrow A = X_1.(X_1 + Z_1),$
$E = d_1.C.C, \quad \Longrightarrow W_1 = X_1 + Y_1,$
$H = (d_1Z_2 + d_2W_2).W_1.C, \quad \Longrightarrow H = (d_1Z_2 + d_2W_2).W_1.C,$
$I = d_1.C.Z_1, \quad \Longrightarrow U = E + A.D,$
$U = E + A.D, \quad \Longrightarrow B = Y_1.(Y_1 + Z_1),$
$V = E + B.D, \quad \Longrightarrow V = E + B.D,$
$S = U.V, \quad \Longrightarrow S = U.V,$
$X_3 = S.Y_1 + (H + X_2(I + A(Y_2 + Z_2))).V.Z_1, \quad \Rightarrow Z_3,$
$Y_3 = S.X_1 + (H + Y_2(I + B(X_2 + Z_2))).U.Z_1, \quad \Rightarrow Y_3,$
$Z_3 = S.Z_1. \quad \Longrightarrow X_3.$

The main architecture of the binary Edwards curves processor design is shown in Figure 6.1. It consists of a processor, a bus manager, a ROM and RAM blocks. The starting points (P_X, P_Y), key *(k)* and equation constants (d_1, d_2) are stored in the ROM. Interval values and result points (X_3, Y_3, Z_3) of modular addition are kept in the RAM0. RAM1 and RAM2 store P_1 and P_2 values of the Montgomery ladder, respectively. The bus manager controls the connection between ROM-processor, RAMs-processor, processor-RAMs, RAMs-RAMs according to the address and assign bits.

The processor part of the design consists of a control block, a register file, a modular arithmetic logic unit (MALU) and a shifter. The control block has the finite state machine data path that arranges the inputs, outputs and connects the components according to the addition and multiplication. Moreover, the last inversion process is also controlled by same control block, based on Fermat Little Theorem [7] and several multiplications which are given in Table 6.1. The following subsections indicate the components in details in three sub-groups; processor, bus manager and memory units.

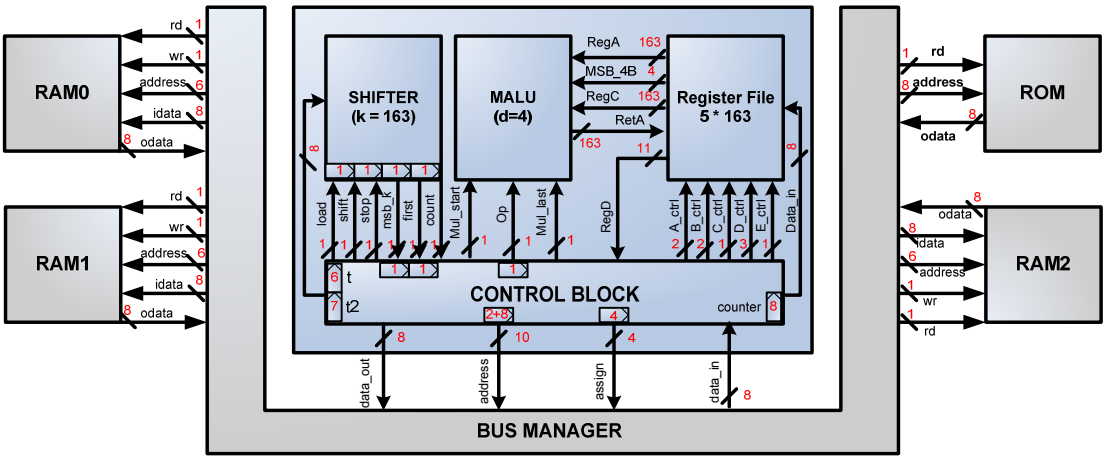

Figure 6.1: The architecture of Binary Edwards Curves Processor (BEC Processor)

6.1.1 Processor

A processor is the main part of the design. It assigns the necessary inputs from storage to the register and it controls the addition and multiplication operations in finite fields. Afterwards, according to the double and add algorithm of Montgomery ladder, is given in Algorithm 10, the processor sends the intermediate values to the storage parts over the bus. Moreover, the next point is calculated according to key value (k), while a counter is counting shift operation of the key. Finally, the counter gives the finish signal and calculation stops. $Q = k.P$ is calculated.

6.1.1.1 MALU (Modular Arithmetic Logic Unit) Design

The first MALU architecture was initially proposed in [28]. It can perform both addition and multiplication operations over finite fields as shown in Figure 6.2. Operations are performed as shown by Equation 6.1 [28].

$$\begin{aligned} A(x) &= B(x)\ .\ C(x)\ mod\ P(x) \qquad if\ cmd = 1 \\ A(x) &= A(x) + C(x)\ mod\ P(x) \qquad if\ cmd = 0 \end{aligned} \tag{6.1}$$

where $A(x) = \sum a_i.x^i, B(x) = \sum b_i.x^i, C(x) = \sum c_i.x^i$ and $P(x) = x^{163} + x^7 + x^6 + x^3 + 1$.

In the MALU, the cost of the field multiplication and addition is $\frac{163}{d}$ and one clock cycle for the digit size d, respectively. In every cell, multiplication and addition can share the same XOR array. The MALU can be scaled easily to different digit sizes by using cells in serial.

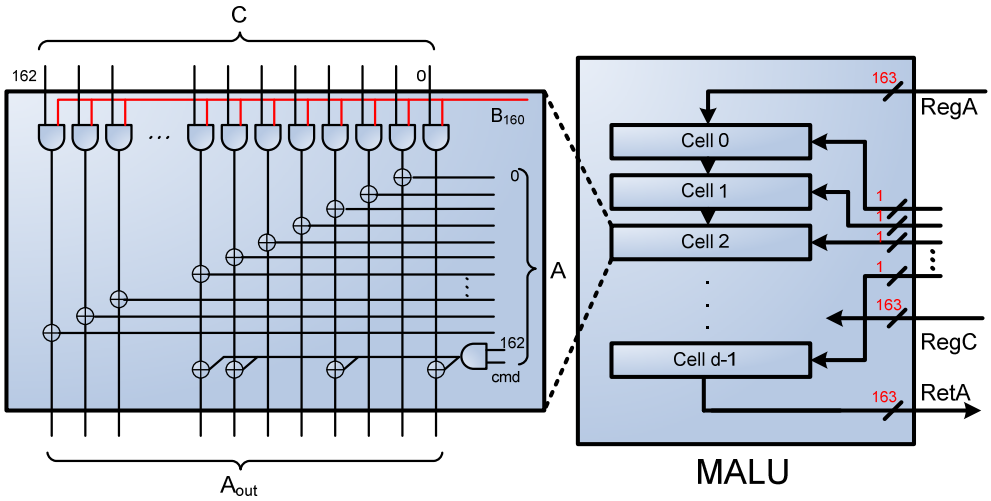

Figure 6.2: BEC Processor's MALU Architecture

The MALU does not contain internal registers, everything works in sequential. As shown in the Equation 6.1 [29], the register file keeps the interval value (*RetA*) and MALU does the calculations. When the MALU performs a multiplication, each digit of multiplication must be provided to the MALU. That means in every cycle, *regB* must be shifted to left by d bits and most significant digits turn back to the least significant bits as a circular shifting. Note that, the shift operation must be circular and the last shifting must be a remainder of $\frac{163}{d}$ so that *regB* turns back to the initial value at the end of the multiplication.

In Figure 6.2, *cmd* signal commands perform multiplication or addition as shown in equation 6.1 [29]. The position of the XOR-gates in the latter array depends on the irreducible polynomial. In this case, the polynomial $P(x) = x^{163} + x^7 + x^6 + x^3 + 1$ is used. In case of a finite field multiplication, the reduction needs to be done if the *MSB* of A is "1". For the finite field addition, *cmd* signal provides the reduction that will not be performed [28].

The data path of the MALU is an MSB serial F_{2^n} multiplier with digit size d. The MALU sums up three types of inputs which are B_i C, A_{MSB} $P(x)$ and A. Afterwards, it gives the intermediate result, *RetA*, by computing $RetA = (A + B_i.C + A_{MSB}.P(x))$. The multiplication operation can be obtained by providing the next input A as *RetA* by

repeating this computation for n times. The addition operation can be obtained at the same hardware with some additional tricks. Input value A is shifted 1-bit left inside the '*CELL*' as shown in Figure 7.4. If 'A' value is shifted to the right before entering the '*cell_3*' and does the XOR as an addition operation for binary fields inside the '*cell_3*, then $C[0] \oplus A[0]$ can be concatenated to the A, it is illustrated in Figure 7.4 on the '*MALU*'. Thus, addition can be done with the same hardware in one clock cycle.

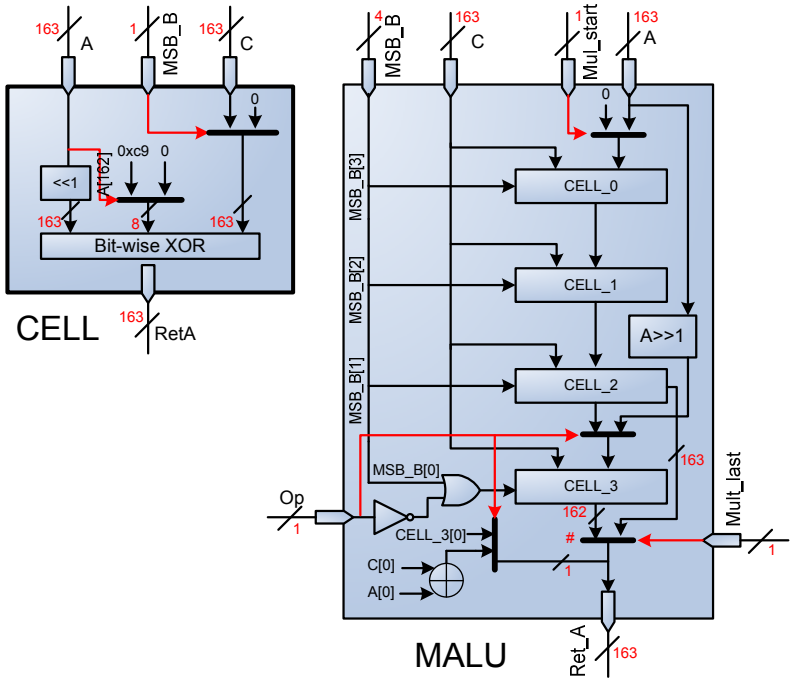

Figure 6.3: Control Scheme of Cell and MALU with $d = 4$

In our design, the optimum digit size is decided as 4 according to the trade-offs between area consumption and processing time. Therefore, the Figure 7.4 illustrates the MALU for $d = 4$.

Here, notice that the shifting process is done in 40 times 4 and 1 time 3 cycles. The *Mult_last* signal controls the multiplexer to choose the output of *cell_3* or *cell_2*, and it selects *cell_2* if it is the last round of multiplication. Also, *mul_start* signal controls *regA* value and sets it to "0" when multiplication starts. The GEZEL code of the Cell and MALU are given in Appendix-C.

6.1.1.2 Register File Design

The register file is the memory part of the processor and temporary values of projective addition are stored in it. The *MALU* uses three registers as operands and the result value. The architecture of register file is shown in Figure 6.4. A circular shift register is

used to reduce the complexity of the multiplexer. In a randomly accessible register file, the area complexity is directly proportional with the square of the number of registers, since every register has inputs as a number of register; on the other hand, the area complexity of the multiplexer in circular shift register is a constant.

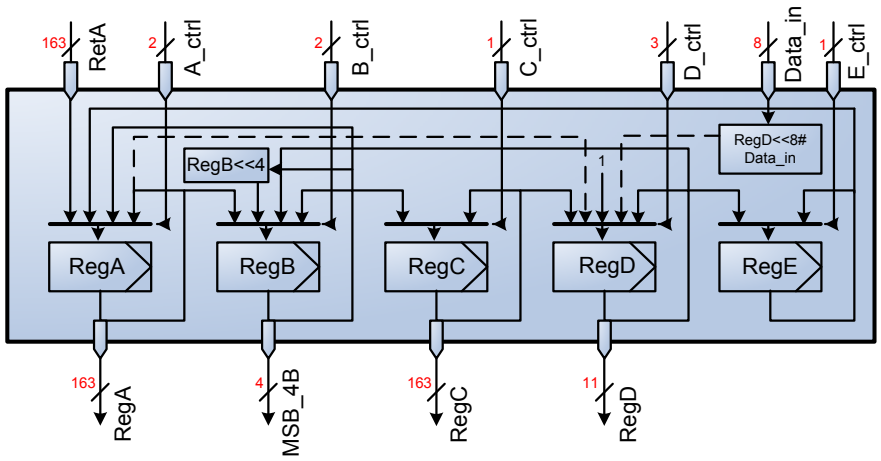

Figure 6.4: Register File Architecture

Although the register file is a circular shift register file as given in Figure 6.4, each register is independently controlled for the efficient management. In our first design, *regA* was used for assigning new values and as an accumulator of operations. In the progress of designing, it is noticed that the assigning operation can be processed simultaneously with multiplication operation if another register is used to assign values. Since, multiplication operation takes from 24 to 163 clock cycles in digit sizes from 7 to 1 respectively and assigning from outside to the register file the data takes only 21 clock cycles. Consequently, *regD*, a spare register, is used for assigning the data to reduce the total processing time.

In our sample design, we stated before that forty times 4-bit and once 3-bit shifting is needed. If we add one more bit to the *regB*, we can ignore 3-bit shifting and regular 4-bit shifting works as shown in Figure 6.5. In first shifting operation, *MSB_4B* of *regB* $B[162], B[161], B[160], B[159]$ goes to $B[2], B[1], B[0], B[163]$ respectively and so on, for the same values, second shifting will be obtained as $B[2], B[1], B[0], B[163]$ to $B[6], B[5], B[4], B[3]$. After 41 cycles, regB gets the initial value back. So, the fifth input of the multiplexer is ignored (Reg B « 3) and the multiplexer can be controlled by only in 2-bit. This feature can be done for the other digit sizes as well. The difference is only remainder of bits after regular shifting done. *RegC* and *regE* do not need any

multiplexers. These registers are only connected with their own or previous ones value and this can be controlled by clock-gating with enable signal which is generated from "*C_ctrl*" and "*E_ctrl*" signals.

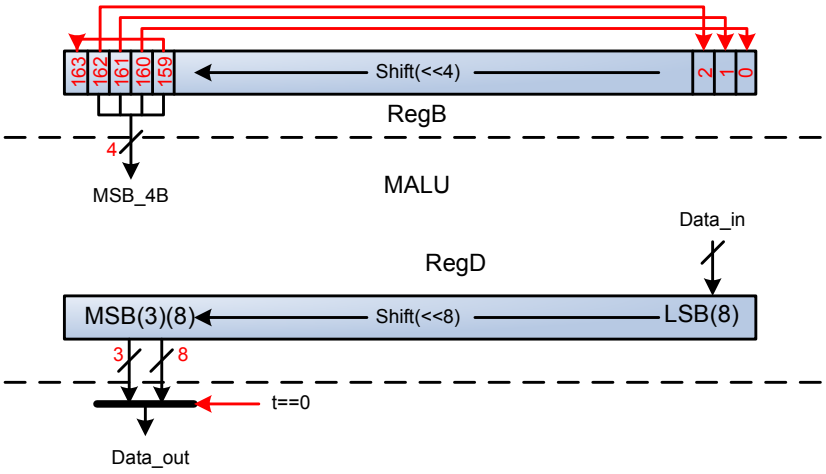

Figure 6.5: Shifting Operation in *regB* and Data Assigning, Taking Process in *regD*

In our final design, the register file inputs are only connected from external memory (ROM or RAM) to the *regD*. Since, as shown in Figure 6.5, the data is assigned by 8-bit inputs, *RegD* performs 8-bit shifts to keep the previous loaded data. Moreover, other inputs of the *regD* are: one from itself, one from *regC* as circular shift register, one from *regA* to connect outside easily like shortcut, one from outside and the last one is '1' to add projective Z-coordinate as initial value as shown in Figure 6.4.

Finally, shifting is done as shown in Figure 6.5. The data is assigned to the *LSB* bits, then shift and concatenate operations are used to fill the *regD*. The output is taken from *MSB* bits of *regD* by the following steps;

1. For $t = 0$ (counter), the first output is taken as most significant 3-bit.
2. For $t = 1$, one 8-bit shifting is omitted and multiplexer chooses the bits between *regD[159:152]*.
3. For $2 \leqslant t \leqslant 21$, every cycle multiplexer chooses the bits between *regD[159:152]* and regD is shifted 8-bit to the left.

6.1.1.3 Shifter Design

In our processor, key value (k) is stored in an internal register as a shifter component. This component has its own finite state machine and is operated by a controller with *load*, *shift* and *stop* signals. The same 8-bit shifting operation is used when the point multiplication is started and k is assigned to the processor only once. The figure 6.6 illustrates the control of shifter.

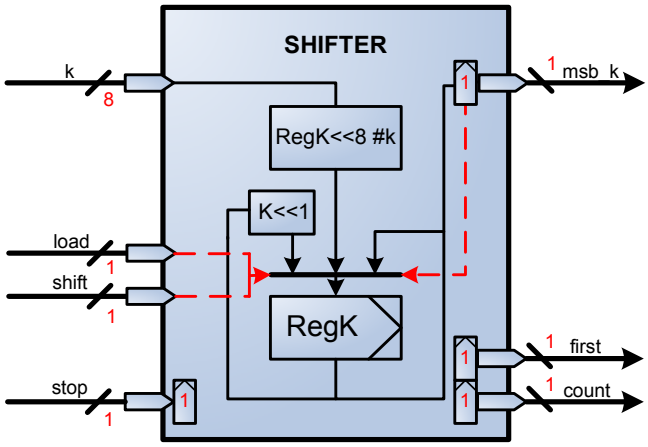

Figure 6.6: The Architecture of Shifter Block

In Figure 6.6 the *load* signal controls the multiplexer to choose 8-bit shifting and concatenation on *regK*. Since it is just a signal, it should be indicated in every cycle as '0' or '1'. The *shift* signal controls the multiplexer to choose 1 bit shifted value to the *regK*. Shifting operation is not used after every projective addition operation, so that each bit is used twice to calculate new P_1 and P_2 values, in the Montgomery ladder. The *stop* signal controls the shifting operation to stop. Moreover, Montgomery ladder requires the most significant bit equals '1' to start the operation. When the assigning operation is over, the first step is searching the *MSB* until it equals '1' as shifting to the left, so that the *MSB* of key can be zero. The finite state machine of shifter is illustrated in Figure 6.7. *MSB_k* value is checked, if it is '0' then shift to the left by one bit is active and the output flag '*count*' is raised, else shifting operation is passive and the output '*first*' is raised. It means, when the *MSB* of k value equals '1', the point multiplication can be started. Note that, *count* flag just increments the counter inside of the control block when shifting operation is processed.

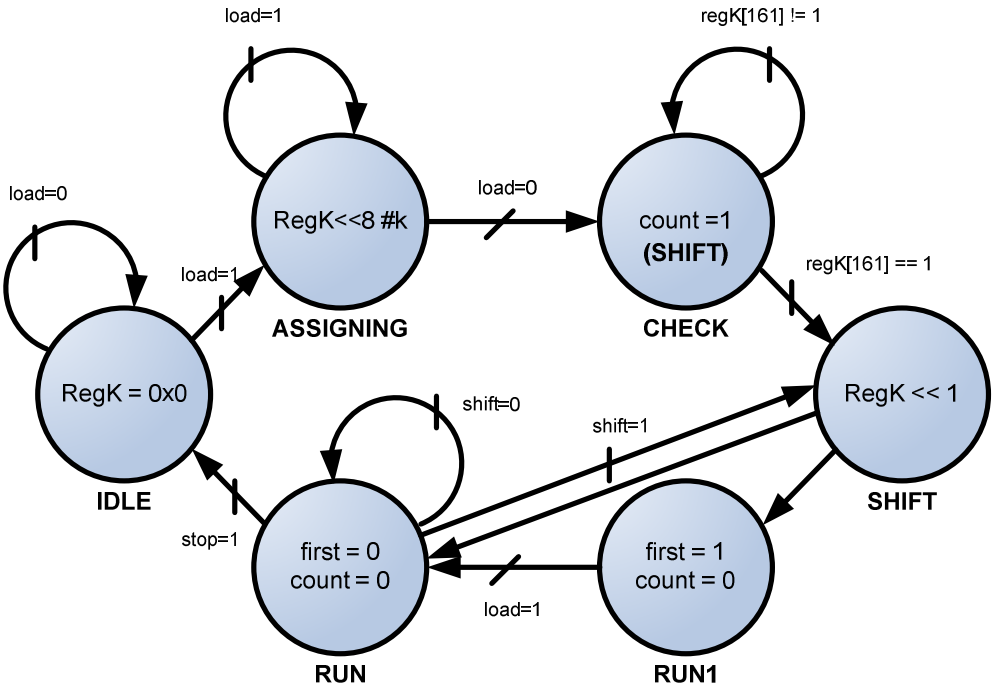

Figure 6.7: The Finite State Machine Diagram of Shifter Component

6.1.1.4 Control Block

In our design, the control block works as the brain of all decisions. It has two different finite state machines to manage projective addition and to convert the result back to the affine coordinates at the final step (inversion). It controls not only the processor but also the outside of the processor as shown in Figure 6.8.

Inside the processor, the control block controls the register file; it keeps the values or performs circular shifting or assigning new value or shifting *regB* during multiplication and keeps return value *A* or assigning values and runs multiplication operation in parallel, with only 8-bit. 3-bit controls the *MALU* to define the operation type, manages the multiplication with respect to counter (*register t*). It also manages shifter blocks finite state machine with 3-bit.

Outside of the processor, the control block controls the interface between memory unit to bus and bus to processor with the *address* and *assign* signals. It receives the values from '*data_in*' input port and puts the values to the memory from '*data_out*' output port.

There are several kinds of register types in the controller block as given in Figure 6.8. One type is just counters, 6-bit register '*t*' counts the multiplication steps, 7-bit register

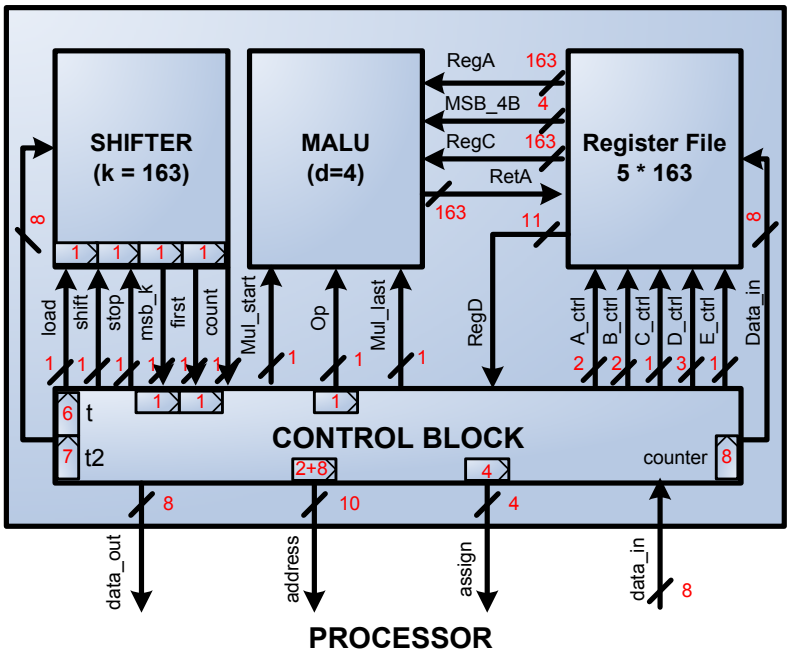

Figure 6.8: The Processor of Binary Edwards Curves

'*t2*' counts the inversion steps which are given in Table 6.1 and 8-bit register '*counter*' counts the key shifting inside shifter, and manages shifting process. One bit registers keep the values that are controlled by a conditional statement in the finite state machine. 10-bit register keeps the address in 8-bit and the read or write operation in 2-bit. Finally, 4-bit register keeps the operation of assign controlling, whether reads from ROM or RAM, writes to RAM, writes from RAM0 to RAM1, RAM2 and manages reading values from P_1 or P_2 depending on the key.

In controller block, there is a sequential finite state machine for repeating one modular projective addition which is given in Figure A.1 in Appendix A and in Algorithm 9. Operations are initialized with the start signal that is created inside in our design, but this signal can also be triggered from the outside. Then controller block manages the bus and ROM to get the k value from ROM to shifter block until 't' counts 21. When *MSB* signal is received from shifter block, it starts to get the first time values from ROM according to Montgomery ladder which is given in Algorithm 10 [24]. Initially, in "*firsttime*" period, the $P_2 \leftarrow 2P$ value is calculated, *assign* signal is set the connection over buses to put these values to RAM2. Note that (as stated before), RAM1 stores P_1 values and RAM2 stores P_2 values. Shifter shifts the 'k' to the left by 1-bit and checks *MSB*, equals '0' or '1'. So, the next point addition operation occurs according to the

MSB and '*assign*' signal which will be explained later. After all bits of k are generated by Montgomery ladder, the control block manages the inversion operation of Z_1 to convert the projective coordinates to affine one.

Algorithm 10 Montgomery Ladder for Point Multiplication

Require: EC point $P = (x, y)$, integer k, $0 < k < M$,
$k = (k_{t-1}, k_{t-2}, \ldots, k_0)_2$, $k_{t-1} = 1$ $P \in E(F_q)$
Ensure: $Q = [k]P$
$P_1 \leftarrow P$, $P_2 \leftarrow 2P$
for i *from* $t-2$ *downto* 0 **do**
if $k_i = 1$ **then**
$P_1 \leftarrow P_1 + P_2$, $P_2 \leftarrow 2P_2$
else
$P_2 \leftarrow P_1 + P_2$, $P_1 \leftarrow 2P_1$
end if
end for
return(P_1)

Consequently, there is an important point in the establishment of *assign* control. It is obvious that the main problem is same finite state machine must work on different calculations. The arrangements of the inputs and outputs are controlled by the help of this four bit control unit, *assign*, which is given in Equation 6.2;

$$assign[3:0] = read \sharp\ online \sharp\ ctrl_{state} \sharp\ msb_k \tag{6.2}$$

where the *read bit* controls which value from which RAM will be assigned, the *online bit* controls carrying values from RAM0 to RAM1 or RAM2 over bus, while FSM of control block is running. The details are given in subsection RAM. Moreover, the $ctrl_{state}$ controls in which state we are (first or second state of Montgomery ladder operations). Finally, the msb_k controls the operations of Montgomery ladder. The assign control map is given in Table B.1 in Appendix B in details.

6.1.2 Bus Manager

The bus manager is a connection between the processor to memory units and outside of the chip. The bus manager gets 10-bit address and 4-bit assign control to manage which ports will be connected. Four most significant bits of the address control are read and write signals of memory units. The most significant two bits are considered

read ROM, read RAM and write RAM. The other two bits indicate which RAM will read or write. The address map of this four bit is given in Figure 6.9.

Address[9]	Address[8]	Address[7]	Address[6]	operation
0	0	0	0	default
0	1	0	0	read ROM
0	1	0	1	read ROM
0	1	1	0	read ROM
0	1	1	1	read ROM
1	0	0	0	write RAM0
1	0	0	1	write RAM0
1	0	1	0	write RAM2
1	0	1	1	write RAM1
1	1	0	0	read RAM0
1	1	0	1	read RAM0
1	1	1	0	read RAM2
1	1	1	1	read RAM1

Figure 6.9: The Map of Address Control

The bus manager has also a register which stores 6 of address bits, to work in parallel, to read from RAM0 and to write to the related RAM one cycle later which is decided by *assign* control. Therefore, we design this RAM blocks separately. In the Montgomery ladder, there is a selection between P_1 or P_2 values according to the step. When the results are obtained, they should be carried to the related places for reuse in the following step by using separated RAMs with same addresses. This register just helps us as a buffer, to use the same address bits from control block. Figure 6.10 shows the architecture of the bus manager.

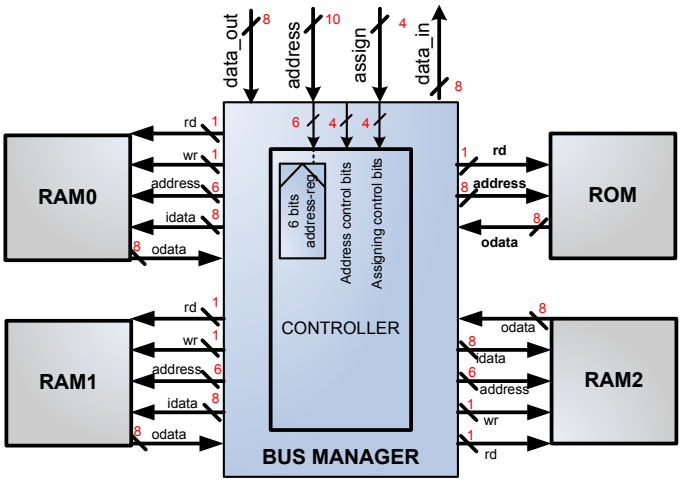

Figure 6.10: Architecture of Bus Manager

6.1.3 Memory Units

6.1.3.1 ROM

In ROM design, look-up tables (LUT) are used to store main point coordinates (X, Y), constant values in binary Edwards curves equation (d_1, d_2) and key value (k). It can be addressed by 8-bit. It is assumed that the connection between ROM and processor is secure against eavesdropping.

6.1.3.2 RAM

'*Ipblock*' feature of GEZEL [25] is used to design RAM blocks. The reason of using separate three RAM block is explained in section 6.1.2. In the "*Ipblock* feature, the word length and the size of the RAM is defined as shown in algorithm 11.

Algorithm 11 Ipblock for RAM design

$ipblock\ RAM_{EC0}(in\ address\ :\ ns(8);$
$in\ wr,\ rd\ :\ ns(1);\ in\ idata\ :\ ns(8);\ out\ odata\ :\ ns(8))\{$
$ipparm\ "wl = 8";$
$ipparm\ "size = 128";\}$

RAM0 is used for interval values and it stores four 163-bit values. After every projective point addition operation finished, new coordinates reveal in order of X, Y, Z and interval value. These first three values are carried to the related RAM for reuse in the following steps of the Montgomery ladder, and it is given in Figure 6.11.

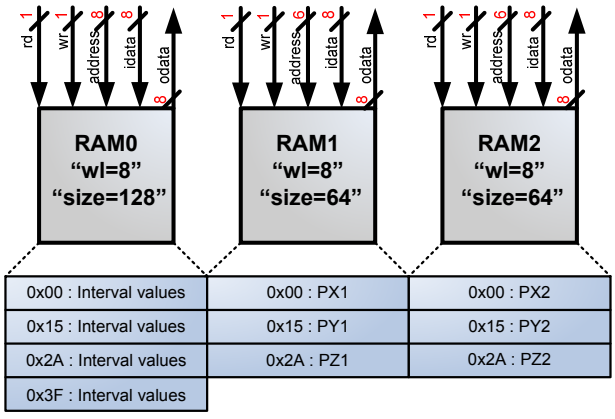

Figure 6.11: RAM Blocks and Storing Values

6.2 Algorithms for Implementation of Binary Edwards Curves

The main operation of the binary Edwards curves is the scalar multiplication. The hierarchical structure of required operations for implementation of BEC is given in Figure 6.12. The scalar multiplication is at the top level and we use the Montgomery Ladder algorithm which is given in Algorithm 10. The next lower level are composed of projective addition and inversion operations, but as stated before, we used projective coordinates to neglect inversion operation in every step of the point addition. So, we use inversion operation at the end of process to get the affine coordinates. The binary Edwards curves projective addition algorithm is used with general d_1 and d_2 constant values for implementing point addition which is given in Algorithm 6. The lowest level consists of the finite field arithmetic operations such as addition and multiplication. Finally, as stated in Section 3 (Essential Concepts), the modular inversion is performed using *Fermat's little theorem*, which is given in Theorem 6.2 and in Theorem 6.3 [7].

Theorem 6.2 : If p is prime, then

$a^p \equiv a(mod p)$.

Theorem 6.3 : *Fermat's little theorem* If p is prime and $p \nmid a$, then

$a^{p-1} \equiv 1(mod p)$.

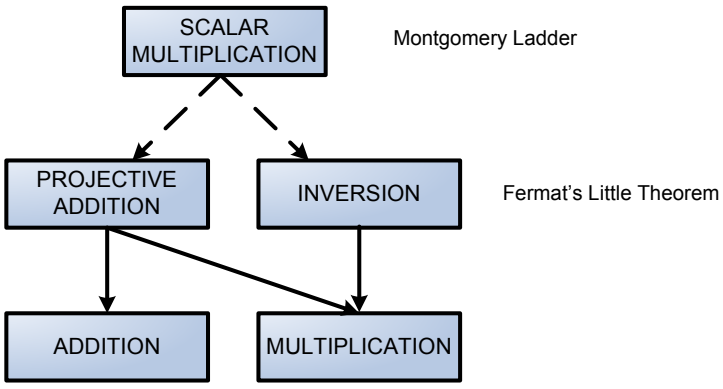

Figure 6.12: Required Operations for Binary Edwards Curves Implementation

In Fermat's little theorem, the prime number p refers to 2^{163} in our design. Then in order to calculate the inverse of Z coordinate, we must calculate the value $a^{2^{163}-2} \equiv a^{-1}$. This can be performed by multiplications and squarings. Note that, we did not implement doubling algorithm in our design, instead we used the multiplication with

same operands to implement doubling. In [30], Itoh and Tsujii proposed an efficient technique to calculate the inverse in an optimal way.

From Fermat's little theorem, we have $a^{-1} = a^{2^n-2} = (a^{2^{n-1}-1})^2$, for all $a \in F_{2^n}$ [7]. In our design, n equals to 163, so $n-1$ is even number. Then we can write:

$$a^{2^{n-1}-1} = a^{(2^{\frac{n-1}{2}}-1)(2^{\frac{n-1}{2}}+1)} = (a^{2^{\frac{n-1}{2}}-1})^{2^{\frac{n-1}{2}}} a^{2^{\frac{n-1}{2}}-1}. \tag{6.3}$$

In our case for $F_{2^{163}}$ we get:

$$a^{-1} = a^{2^{163}-2} = (a^{2^{162}-1})^2 = ((a^{2^{81}-1})^{2^{81}} a^{2^{81}-1})^2, \tag{6.4}$$

which means that we need to use formula $a^{2^{n-1}-1}$, but now $n-1$ is odd number [7]. In this case:

$$a^{2^{n-1}-1} = a.a^{2^{n-1}-2} = a(a^{2^{n-2}-1})^2. \tag{6.5}$$

The inversion operation can be computed by 171 multiplication by repeating this process. All process is given in Table 6.1.

Table 6.1: Example of inversion in $F_{2^{163}}$

Calculation	Number of Multiplication
$a^{2^2-1} = a.a^2$	2M
$a^{2^4-1} = (a^{2^2}-1)^{2^2}.a^{2^2-1}$	3M
$a^{2^5-1} = a.(a^{2^4-1})^2$	2M
$a^{2^{10}-1} = (a^{2^5-1})^{2^5}.a^{2^5-1}$	6M
$a^{2^{20}-1} = (a^{2^10-1})^{2^{10}}.a^{2^{10}-1}$	11M
$a^{2^{40}-1} = (a^{2^{20}-1})^{2^{20}}.a^{2^{20}-1}$	21M
$a^{2^{80}-1} = (a^{2^{40}-1})^{2^{40}}.a^{2^{40}-1}$	41M
$a^{2^{81}-1} = a.(a^{2^{80}-1})^2$	2M
$a^{2^{162}-1} = (a^{2^{81}-1})^{2^{81}}.a^{2^{81}-1}$	82M
$a^{-1} = a^{2^{163}-2} = (a^{2^{162}-1})^2$	1M
Total	171M

7. RESULTS

The implementation of binary Edwards curves is written in GEZEL [25] hardware description language and is optimized for low area consumption. We attempted to implement Algorithm 9 as fast as possible. Assigning operations are hidden within the multiplication operation. In one multiplication time, the processor does not only assigning operation but also stores the values from *regD* to memory unit while digit size is not overlapping the time limit. For instance, multiplication operation takes 41 clock cycles for $d = 4$, and single assigning operation takes 21 clock cycles by 8-bit per cycle. Therefore, if digit size is below four, it will be possible to do assigning and storing operations parallel with the multiplication operation. $0.13\mu m$ standard cell library is used as the technology library. For the synthesis, the VHDL codes, which are automatically generated by GEZEL, were used. The design was synthesized by Synopsys Design Vision [26] at different frequencies, but especially 400 kHz comparisons are given. RTL level and gate level simulations were done by GEZEL *fdlsim* feature and Modelsim, example simulations are added to the Appendix E. All power consumption estimations were also synthesized on Synopsys Power Compiler tool and these measurements were carried out by the switching activity which was captured by the gate level simulations in ModelSim.

7.1 Power Estimation Methodology

In our first estimation method, the power consumption is estimated just by Synopsys Design Vision without any switching activity. Then, the similar results are seen for different digit sizes. It is observed that the power consumption is only affected by frequency changes. In order to estimate power consumption of the designs as accurate as possible, power simulations were done in the gate-level by using the switching activity. The power estimation hierarchy is shown in Figure 7.1.

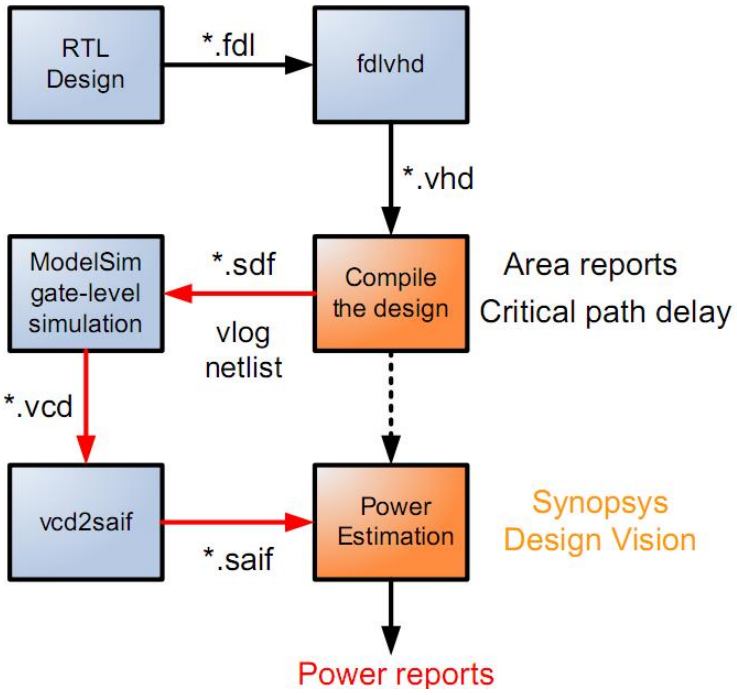

Figure 7.1: Power estimation flow

After compiling the design with Synopsys, area and timing reports are obtained. In Figure 7.1, dashed arrow shows first estimation type without switching activity. Red arrows show the method to get the files. First, a verilog netlist and a standard delay format (SDF) file are produced in order to be used in simulations. The SDF file contains the timing information of the cells in the netlist. After this step, the netlist is simulated in ModelSim and a value change dump (VCD) file is created by the simulator. The VCD file keeps the switching activity of the signals of the netlist. Then, this VCD file is converted to a switching activity interchange format (SAIF) file, which is used by the Synopsys Power Compiler. As the final step, Power Compiler generates the power estimation reports by using the netlist information and SAIF file. After this measurement setup is applied to the power consumption estimation, results are changed by the changing on digit sizes.

7.2 Area, Power Consumption in Different Frequencies

Six different implementation of binary Edwards curves with different digit sizes are synthesized in order to find the best solution. Firstly, design is implemented with $d = 3$ digit size. Afterwards, digit size is changed and resynthesized. The area and power consumptions, longest critical path delay and processing clock cycle are given

in 100kHz, 400kHz, 1MHz, 5MHz, 20MHz and 50MHz in Table 7.1, 7.2, 7.3, 7.4, 7.5 and 7.6, respectively. Equivalent gate numbers of the designs are calculated by dividing the total area of the circuit by the area of one 2-input NAND gate.

In Table 7.2, it can be observed from the results that combinational area is increasing with digit size but non-combinational area stays the same. Non-combinational area does not change, since all these results depend on the same five register design. The difference between these results is in the number of *CELL* which consists of *AND* and *XOR* gates, there is no additional register coming from *CELL*. Moreover, the power consumption increases according to the higher digit size and higher frequencies. Critical path of the design changes depend on the compiler, but basically values are around 14-15 ns. For instance, if we add an extra cell in our MALU in serial, then we expect to see, the path of the signal will increase. *Slack* means that the rest of the period of a clock cycle after delay is removed. Our example design ($d = 4$) has around 14 ns delay, which means our design will worked up to $70MHz$. The area consumption of our example design is given in Table 7.7 in details.

Table 7.1: Results of implementation in $100kHz$ in $0.13\mu m$ technology

	Area Est. (kG)			Power Est. (μW)		Time Est. (ns)		
digit	Total	Comb.	Non-comb.	$T_{P_{dyn}}$	P_{cell}	Critical Path	Slack	cycle
d=1	13.08	6.81	6.27	2.27	1.84	15.08	2484	1499716
d=2	14.23	7.96	6.27	2.76	2.10	15.63	2484	835678
d=3	14.75	8.49	6.26	3.17	2.34	15.21	2484	614333
d=4	15.03	8.76	6.26	3.28	2.41	15.11	2484	505975
d=5	15.90	9.64	6.26	3.43	2.45	16.62	2483	463496
d=6	16.43	10.16	6.26	3.56	2.5	17.51	2482	436945

Table 7.2: Results of implementation in $400kHz$ in $0.13\mu m$ technology

	Area Est. (kG)			Power Est. (μW)		Time Est. (ns)		
digit	Total	Comb.	Non-comb.	$T_{P_{dyn}}$	P_{cell}	Critical Path	Slack	cycle
d=1	13.08	6.8	6.27	10.49	8.82	15.15	2485	1499716
d=2	14.23	7.96	6.27	12.50	9.82	15.31	2485	835678
d=3	14.74	8.48	6.26	14.43	10.84	13.98	2486	614333
d=4	15.04	8.77	6.27	14.42	10.77	13.83	2486	505975
d=5	15.90	9.63	6.27	15.81	11.46	14.09	2486	463496
d=6	16.42	10.15	6.27	15.86	11.42	14.70	2485	436945

Table 7.3: Results of implementation in $1MHz$ in $0.13\mu m$ technology

	Area Est. (kG)			Power Est. (μW)		Time Est. (ns)		
digit	Total	Comb.	Non-comb.	$T_{P_{dyn}}$	P_{cell}	Critical Path	Slack	cycle
d=1	13.07	6.8	6.27	25.6	20.7	14.92	985	1499716
d=2	14.23	7.96	6.27	29.79	22.47	16.87	983	835678
d=3	14.75	8.49	6.26	31.83	23.43	16.72	983	614333
d=4	15.03	8.77	6.26	32.75	24.07	14.61	985	505975
d=5	15.90	9.63	6.26	34.36	24.59	17.53	982	463496
d=6	16.42	10.15	6.26	35.73	25.11	19.32	981	436945

Table 7.4: Results of implementation in $5MHz$ in $0.13\mu m$ technology

	Area Est. (kG)			Power Est. (μW)		Time Est. (ns)		
digit	Total	Comb.	Non-comb.	$T_{P_{dyn}}$	P_{cell}	Critical Path	Slack	cycle
d=1	13.06	6.78	6.28	108.99	91.19	15.57	184	1499716
d=2	14.20	7.93	6.27	148.44	112.04	16.83	183	835678
d=3	14.72	8.46	6.27	157.63	116.33	15.66	184	614333
d=4	15	8.74	6.26	162.76	119.97	14.52	185	505975
d=5	15.87	9.6	6.26	170.14	122.01	15.84	184	463496
d=6	16.38	10.12	6.26	177.43	124.99	18.25	182	436945

Table 7.5: Results of implementation in $20MHz$ in $0.13\mu m$ technology

	Area Est. (kG)			Power Est. (μW)		Time Est. (ns)		
digit	Total	Comb.	Non-comb.	$T_{P_{dyn}}$	P_{cell}	Critical Path	Slack	cycle
d=1	13.04	6.77	6.27	425.1	344.1	14.56	35	1499716
d=2	14.18	7.91	6.27	492.09	372.66	15.29	35	835678
d=3	14.72	8.45	6.26	526.73	388.46	15.33	35	614333
d=4	15	8.73	6.27	545.33	401.2	13.93	36	505975
d=5	15.85	9.59	6.26	572.78	410.54	15.54	34	463496
d=6	16.37	10.11	6.26	595.43	419.53	16.89	33	436945

Table 7.6: Results of implementation in $50MHz$ in $0.13\mu m$ technology

	Area Est. (kG)			Power Est. (μW)		Time Est. (ns)		
digit	Total	Comb.	Non-comb.	$T_{P_{dyn}}$	P_{cell}	Critical Path	Slack	cycle
d=1	13.037	6.76	6.276	1390	1100	14.51	5.49	1499716
d=2	14.18	7.91	6.268	1630	1210	15.40	5.6	835678
d=3	14.715	8.45	6.264	1930	1350	15.27	4.73	614333
d=4	14.992	8.725	6.267	1860	1330	13.97	6.03	505975
d=5	15.85	9.586	6.263	2170	1460	15.77	4.23	463496
d=6	16.365	10.10	6.263	2250	1490	17.64	2.36	436945

Table 7.7: Our example design for $d = 4$ and $400kHz$

	Area Estimation (kG)		
d=4	Total	Comb.	Non-comb.
TOP	15.04	8.77	6.27
FSM	2.74	2.47	0.27
MALU	3.1	3.1	0
CELL	0.54	0.54	0
Reg.F	7.06	2.09	4.97
Shifter	1.94	0.93	1.01
BUS	0.2	0.17	0.03
ROM	0.36	0.31	0.05

7.3 Trade-off

The trade-off between six different implementations with different digit sizes is given in Figure 7.2. It is obvious that between $d = 1$ and $d = 2$ the speed of the design is getting faster dramatically. Then, the design is getting faster proportional to increasing area consumption, and finally between $d = 5$ and $d = 6$ area consumption is increasing sharply with respect to time. Moreover, in Figure 7.3, power consumption of design ($d = 4$) is given with throughputs. The power consumption increases not only proportional to the throughput but also according to the digit size. Figure 7.4 shows different power consumptions in different digit sizes.

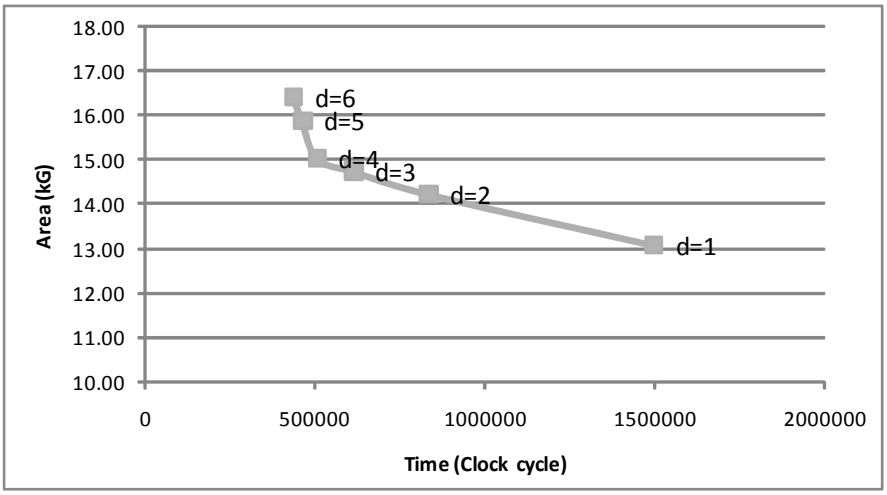

Figure 7.2: Area Consumption vs. Time

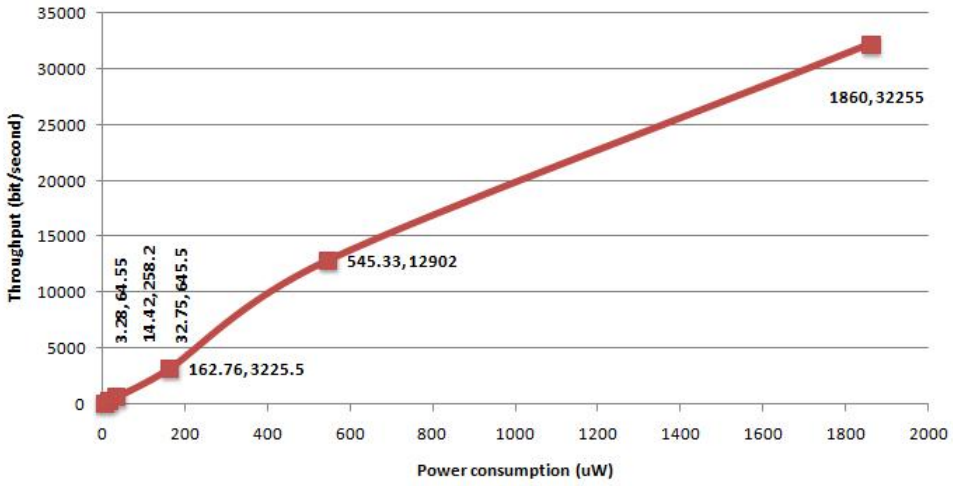

Figure 7.3: Throughput vs. Power Consumption in $d = 4$

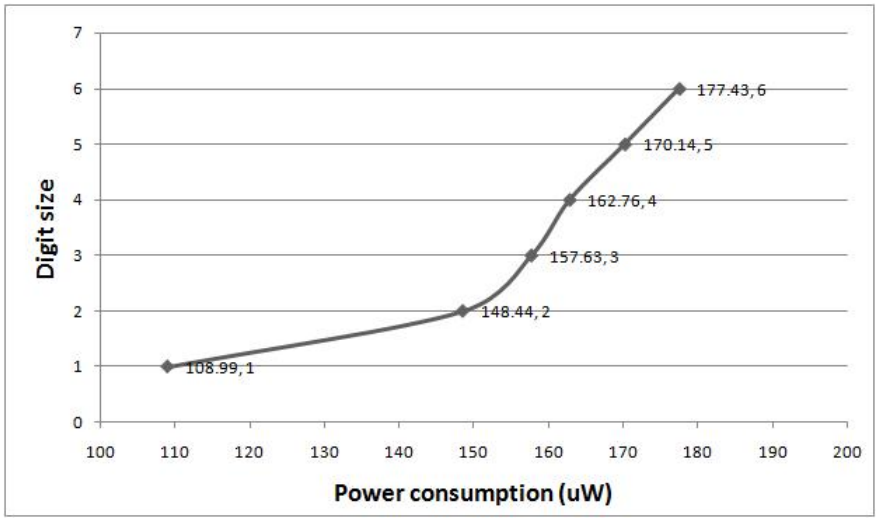

Figure 7.4: Power Consumption in $5MHz$ with different digit sizes

7.4 Clock-gating

Clock gating is one of the power-saving techniques used on many synchronous circuits. Clock gating support additional logic to a circuit to prune the clock tree. Thus, flip-flops do not change state in disabled parts of the design. Their switching power consumption goes to zero, and only leakage currents are incurred. In our design, after clock gating is applied to our register file, new results are reduced as %7 of area consumption and %22 of power consumption in average. The results are given in Table 7.8 and Table 7.9 for 400kHz and 5MHz respectively.

RTL clock gating works by identifying groups of flip-flops which share a common enable control signal. RTL clock gating uses this enable signal to control a clock gating circuit which is connected to the clock ports of all of the flip-flops with common enable term. These flip-flops consume zero dynamic power as long as this enable signal is false. An example of logic circuit illustrates the method of clock gating to understand easily in Figure 7.5.

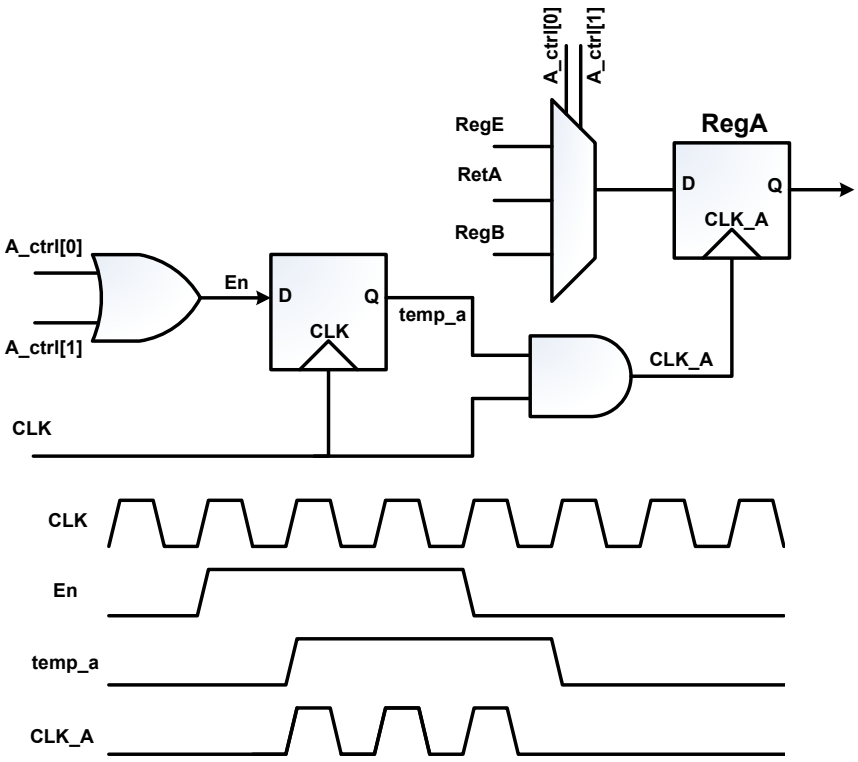

Figure 7.5: Process of Clock gating

Table 7.8: Results of implementation in $400kHz$ in $0.13\mu m$ technology after clock-gating

	Area Est. (kG)			Power Est. (μW)		Time Est. (ns)		
digit	Total	Comb.	Non-comb.	$T_{P_{dyn}}$	P_{cell}	Critical Path	Slack	cycle
d=1	12.11	6.308	5.804	8.12	6.18	11.54	2488.46	1499716
d=2	12.92	7.113	5.807	9.55	6.92	12.06	2487.94	835678
d=3	13.47	7.665	5.805	10.32	7.3	12.04	2487.96	614333
d=4	14.074	8.276	5.798	11.36	7.82	10.45	2489.55	505975
d=5	14.622	8.817	5.805	11.63	7.88	10.55	2489.45	463496
d=6	15.184	9.363	5.821	12.24	8.12	10.93	2489.07	436945

Table 7.9: Results of implementation in $5MHz$ in $0.13\mu m$ technology after clock-gating

	Area Est. (kG)			Power Est. (μW)		Time Est. (ns)		
digit	Total	Comb.	Non-comb.	$T_{P_{dyn}}$	P_{cell}	Critical Path	Slack	cycle
d=1	12.07	6.27	5.804	106.57	80.62	10.78	189.22	1499716
d=2	12.9	7.1	5.805	124.57	89.55	11.75	188.25	835678
d=3	13.43	7.62	5.804	135.76	95.43	10.92	189.08	614333
d=4	14.03	8.23	5.798	145	99.97	10.29	189.71	505975
d=5	14.58	8.77	5.806	148.7	100.96	10.38	189.62	463496
d=6	15.15	9.33	5.821	155.86	103.39	10.79	189.21	436945

8. CONCLUSION

In this dissertation, the first hardware implementation of binary Edwards curves is presented. The design steps of the implementation are given in details. The optimization of number of registers and clock cycles is stated. The design results for six different digit sizes are given. It can be seen that power consumption is related mostly to the frequency, so the first thing is to choose the clock frequency. And then the trade-off between area and time can be decided. The best digit size of design can be $d = 4$, since the working time is 108358 clock cycle less than $d = 3$ and area consumption is slightly bigger than $d = 3$. In our example design the best implementation results can be provided for $d = 4$.

As the future work some further improvements are possible. It could be possible to neglect Y coordinates of projective coordinates and to use only the X coordinate for projective addition algorithm. Moreover, common-Z coordinate can be applied to this implementation. Then, the complexity of calculations can be reduced and it will take less clock cycles to finish point multiplication. Furthermore, the immunity of the implementation must be checked against Simple Power Analysis (SPA) and Differential Power Analysis (DPA) attacks.

REFERENCES

[1] **Frösen, J.**, 1995, Practical Cryptosystems and their Strength, Tik-110.501 Seminar on Network Security.

[2] **Menezes, A.J.**, **van Oorschot, P.C. and Vanstone, S.A.**, 2001. Handbook of Applied Cryptography, CRC Press, *http://www.cacr.math.uwaterloo.ca/hac/*.

[3] **NIST**, November 2001, Announcing the ADVANCED ENCRYPTION STANDARD (AES), FIPS PUB 197.

[4] **Preneel, B.**, 2007. An Introduction to Modern Cryptology, **J. Bergstra and K. de Leeuw**, editors, The History of Information Security. A Comprehensive Handbook, Elsevier, pp. 565–592.

[5] **Barker, E.**, **Barker, W.**, **Burr, W.**, **Polk, W. and Smid, M.** NIST SP800-57: Recommendation for Key Management Part 1: General(Revised), Technical report.

[6] **Diffie, W. and Hellman, M.**, 1976. New directions in cryptography, IEEE Transactions on Information Theory.

[7] **Batina, L.**, December 2005. Arithmetic and Architectures for Secure Hardware Implementations of Public-Key Cryptography, Ph.D. thesis, Katholieke Universiteit Leuven, Belgium.

[8] **Koblitz, N.**, January 1987. Elliptic Curve Cryptosystems, *Mathematics of Computation*, **48(177)**, 203–209.

[9] **Miller, V.S.**, 1985. Use of Elliptic Curves in Cryptography, **H.C. Williams**, editor, CRYPTO, volume 218 of *Lecture Notes in Computer Science*, Springer, pp. 417–426.

[10] **Studholme, C.**, 2002, The Discrete Logarithm Problem, Research paper at University of Toronto.

[11] **Örs Yalçın, S.B.**, February 2005. Hardware Design of Elliptic Curve Cryptosystems and Side-Channel Attacks, Ph.D. thesis, Katholieke Universiteit Leuven, Belgium.

[12] **Hankerson, D.**, **Menezes, A. and Vanstone, S.**, 2004. Guide to elliptic curve cryptography, Springer, New York.

[13] **Saeki, M.**, February 1997. Elliptic Curve Cryptosystems, Ph.D. thesis, McGill University Montreal, Canada.

[14] **I. Blake, G. Seroussi, N.**, editor, 2000. Elliptic curves in Cryptography, Cambridge University Press.

[15] **Consortium, N.**, 2003, NESSIE security report, Technical report, *https://www.cosic.esat.kuleuven.be/nessie/deliverables/D20-v2.pdf*.

[16] **NIST**, November 2008, Digital Signature Standard (DSS), FIPS PUB 186-3.

[17] **Batina, L.**, March 2009, Elliptic Curve Cryptography, Lecture Handout.

[18] **Silverman, J.**, 1986. The Arithmetic of Elliptic Curves, volume 106, New York: Springer-Verlag.

[19] **Morain, F. and Olivos, J.**, 1990. Speeding Up The Computations On An Elliptic Curve Using Addition-Subtraction Chains, *Theoretical Informatics and Applications*, **24**, 531–543.

[20] **Bernstein, D.J. and Lange, T.**, 2007. Faster Addition and Doubling on Elliptic Curves, In Asiacrypt 2007 [10, pp. 29–50.

[21] **Edwards, H.M.**, 2007. A Normal Form for Elliptic Curves, *Bulletin of the American Mathematical Society*, **44(3)**, 393–422.

[22] **Bernstein, D.J.**, **Lange, T. and Farashahi, R.R.**, 2008, Binary Edwards Curves, Cryptology ePrint Archive, Report 2008/171, *http://eprint.iacr.org/*.

[23] **Kim, K.H. and Kim, S.I.**, 2007, A New Method for Speeding Up Arithmetic on Elliptic Curves over Binary Fields, Cryptology ePrint Archive, Report 2007/181, *http://eprint.iacr.org/*.

[24] **Montgomery, P.L.**, 1987. Speeding the Pollard and elliptic curve methods of factorization, *Mathematics of Computation*, **48**, 243–264.

[25] **GEZEL**, http://rijndael.ece.vt.edu/gezel2/index.php.

[26] **Synopsys, I.**, 2006, Design Compiler Tutorial Using Design Vision.

[27] **Oswald, E.**, 2002. Enhancing Simple Power-Analysis Attacks on Elliptic Curve Cryptosystems, **B.S.K. Jr.**, **Çetin Kaya Koç and C. Paar**, editors, CHES, volume 2523 of *Lecture Notes in Computer Science*, Springer, pp. 82–97.

[28] **Sakiyama, K.**, **Batina, L.**, **Mentens, N.**, **Preneel, B. and Verbauwhede, I.**, 2006. Small-footprint ALU for Public-Key Processors for Pervasive Security, Proc. Workshop RFID Security (RFIDSec '06).

[29] **Lee, Y.K.**, **Sakiyama, K.**, **Batina, L. and Verbauwhede, I.**, 2008. Elliptic-Curve-Based Security Processor for RFID, *IEEE Transactions on Computers*, **57(11)**, 1514–1527.

[30] **T. Itoh, S.T.**, 1988. Effective recursive algorithm for computing multiplicative inverses in GF(2^m), volume 24(6), Electronic Letters, pp. 334–335.

A. Control Sequence of Control Block

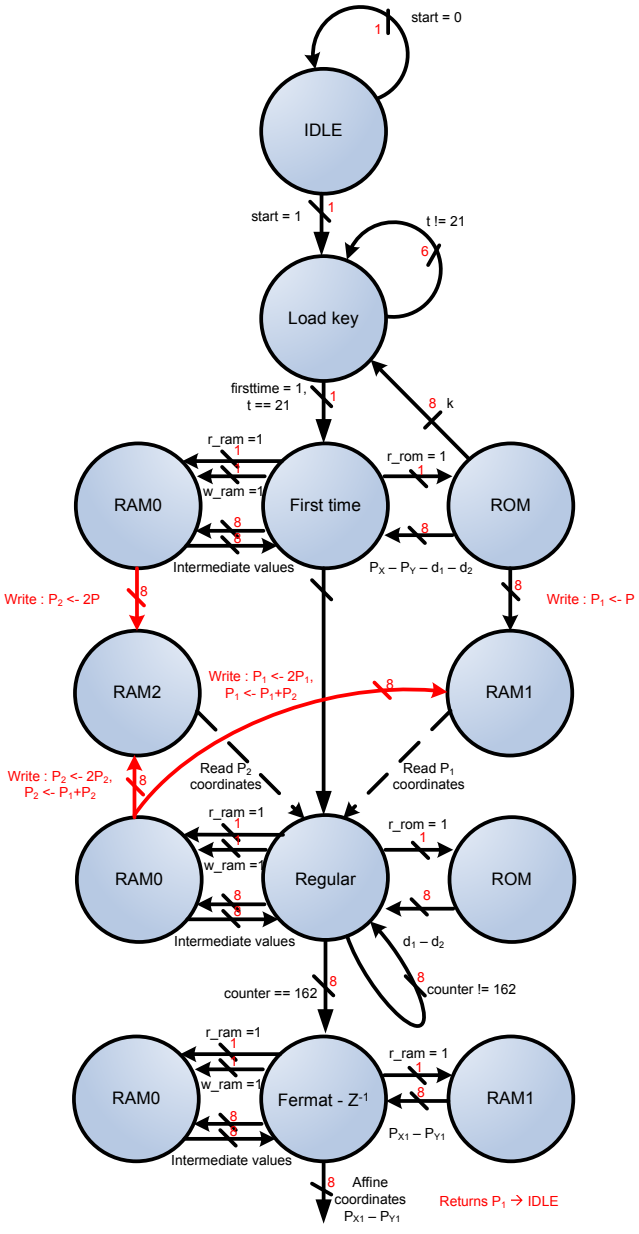

Figure A.1: The Finite State Machine of Control Block

B. Control Bits of Assign Operation

assign	read	online	ctrl state	msb_k	rrom	rram0	rram1	rram2	wram0	wram1	wram2	operation
0	0	0	0	0	1	1	1	1	1	1	1	regular FSM
1	0	0	0	1	1	1	1	1	1	1	1	regular FSM
2	0	0	1	0	1	1	1	1	1	1	1	regular FSM
3	0	0	1	1	1	1	1	1	1	1	1	regular FSM
4	0	1	0	0	0	1	0	0	0	0	1	regular FSM & carry P2 to RAM2
5	0	1	0	1	0	1	0	0	0	1	0	regular FSM & carry P1 to RAM1
6	0	1	1	0	0	1	0	0	0	1	0	regular FSM & carry P1 to RAM1
7	0	1	1	1	0	1	0	0	0	0	1	regular FSM & carry P2 to RAM2
8	1	0	0	0	1	1	1	1	1	1	1	regular FSM
9	1	0	0	1	1	1	1	1	1	1	1	regular FSM
10	1	0	1	0	0	0	1	0	0	0	0	regular FSM & read from P1 (P1 <-2P1)
11	1	0	1	1	0	0	0	1	0	0	0	regular FSM & read from P2 (P2 <-2P2)
12	1	1	0	0	1	1	1	1	1	1	1	regular FSM
13	1	1	0	1	1	1	1	1	1	1	1	regular FSM
14	1	1	1	0	0	0	1	0	0	1	0	regular FSM & read & write RAM1
15	1	1	1	1	0	0	0	1	0	0	1	regular FSM & read & write RAM2

Figure B.1: Assign Operation Control Map

C. Codes for Cell and MALU

```
dp Cell (in MSB_B : ns(1); in A, C : ns(163); out Ret_A : ns(163)) { // Definition of the input and output

    sig Shift_A : ns(163);  // Defining wires
    sig MSB_A : ns(1);

    sfg Mult {      Shift_A = A<<1;
                    MSB_A = A[162];
                Ret_A = ((MSB_B) ?  C^Shift_A : Shift_A) ^ ((MSB_A) ? 0xc9:0); }
}
hardwired h_Cell(Cell)  {Mult;}

// MALU_d4
dp Cell_00 : Cell // Using same Cell 4 times with different component name
dp Cell_01 : Cell
dp Cell_02 : Cell
dp Cell_03 : Cell

dp MALU (in Op, Mul_start, Mul_last : ns(1); in MSB_B : ns(4); in A, C: ns(163); out Ret_A : ns(163)) {      // Word Size = 4

    sig Re_A_00, Re_A_01, Re_A_02, Re_A_03, In_A_00, In_A_01, In_A_02, In_A_03 : ns(163);
    sig MSB_B_00, MSB_B_01, MSB_B_02, MSB_B_03 : ns(1);

use Cell_00(MSB_B_00, In_A_00, C, Re_A_00); use Cell_01(MSB_B_01, In_A_01, C, Re_A_01);          // Port map of Cells
use Cell_02(MSB_B_02, In_A_02, C, Re_A_02); use Cell_03(MSB_B_03, In_A_03, C, Re_A_03);

    sfg Malu {      Ret_A = (Mul_last) ? Re_A_02 : Re_A_03[1:162] # ((Op) ? Re_A_03[0] : (C[0]^A[0]));      // Word Size = 4
                    In_A_00 = (Mul_start)? 0x0 :A;      In_A_01 = Re_A_00;
                    In_A_02 = Re_A_01;                  In_A_03 = (Op) ? Re_A_02:A>>1;
                    MSB_B_00 = MSB_B[3];                MSB_B_01 = MSB_B[2];
                    MSB_B_02 = MSB_B[1];                MSB_B_03 = MSB_B[0](~Op);}
}
hardwired h_MALU(MALU) { Malu; }
```

Figure C.1: Codes for Cell and MALU

D. Comparison Between Normal Clocking and Clock-gating

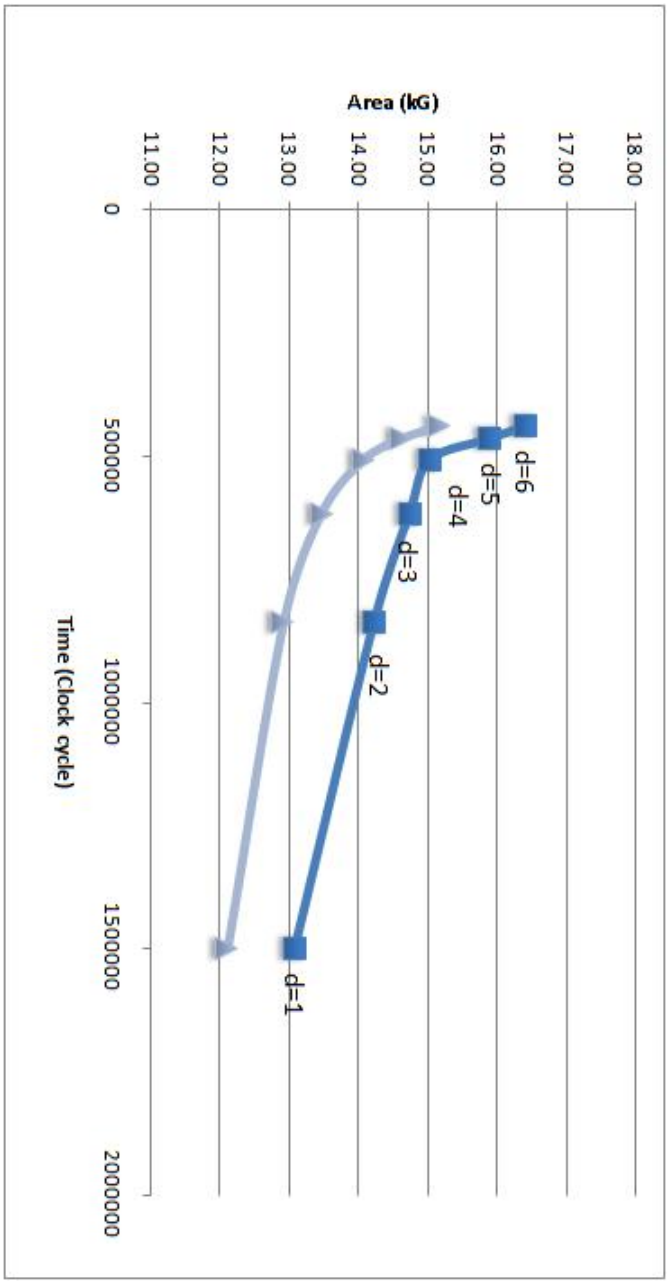

Figure D.1: Area Consumption vs. Time

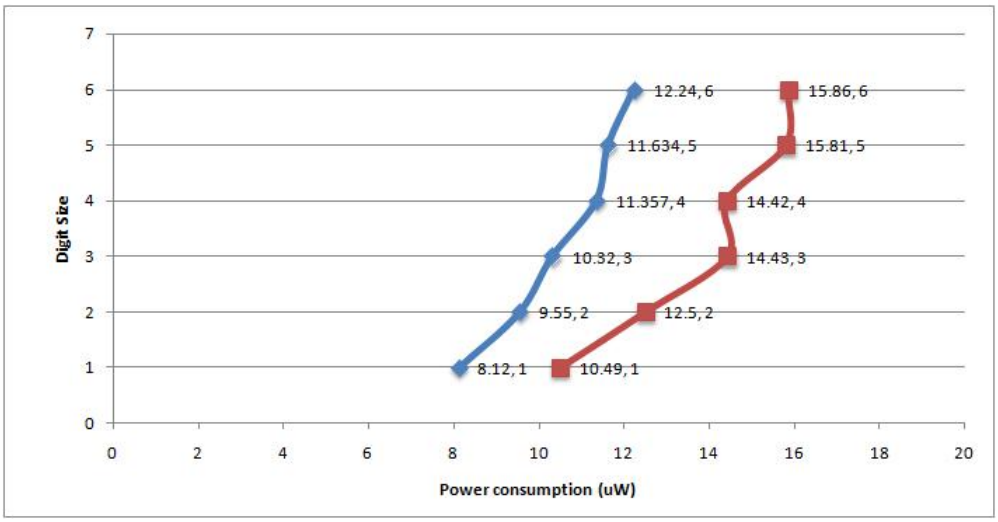

Figure D.2: Frequency vs. Power Consumption in $d = 4$

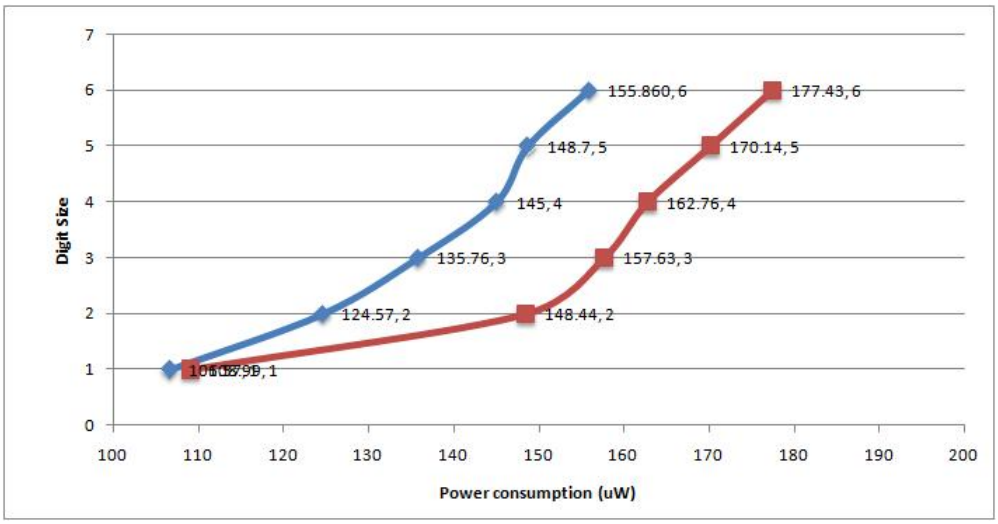

Figure D.3: Power Consumption in $5MHz$ with different digit sizes

E. Simulation Examples

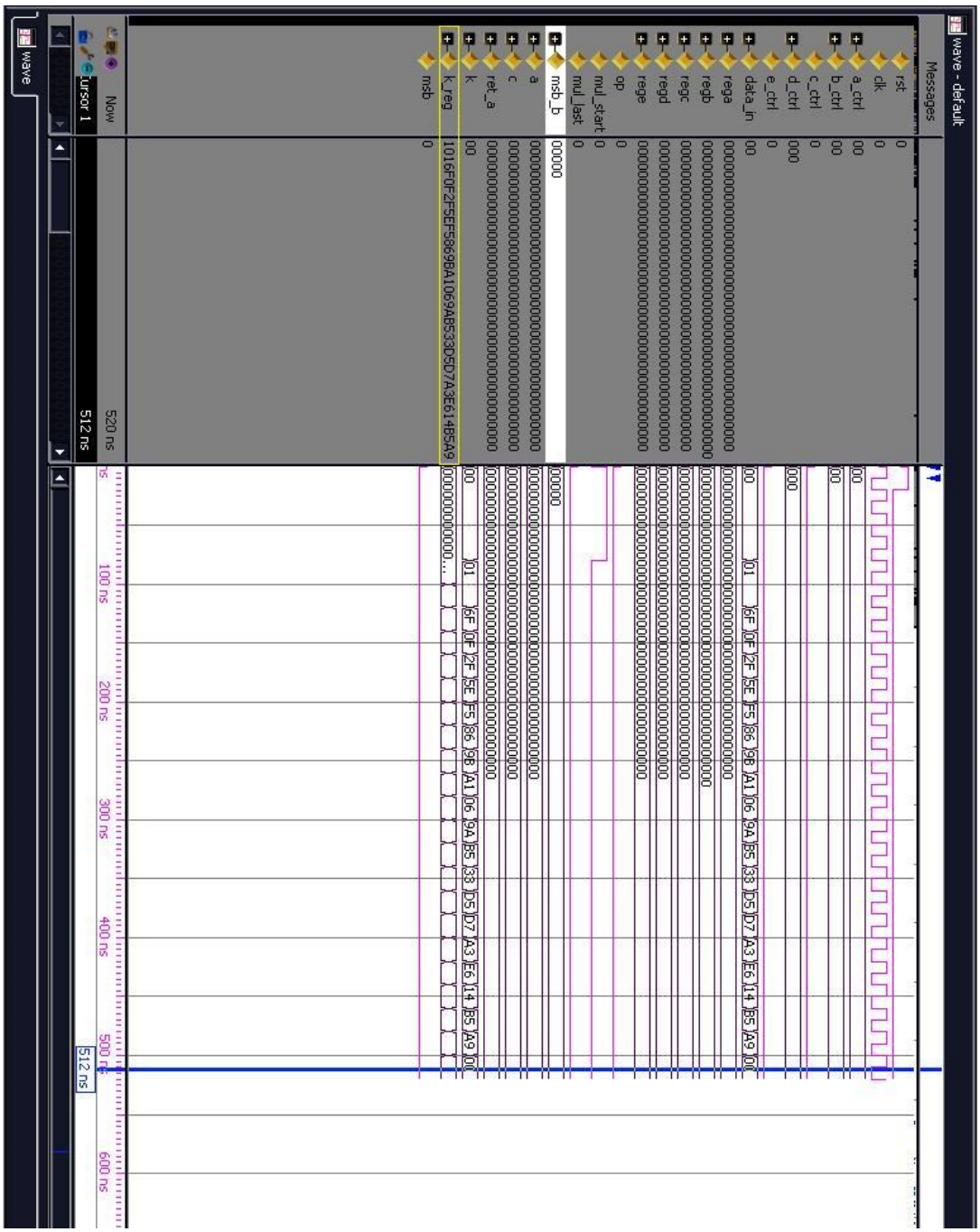

Figure E.1: Assigning Key Value in Modelsim

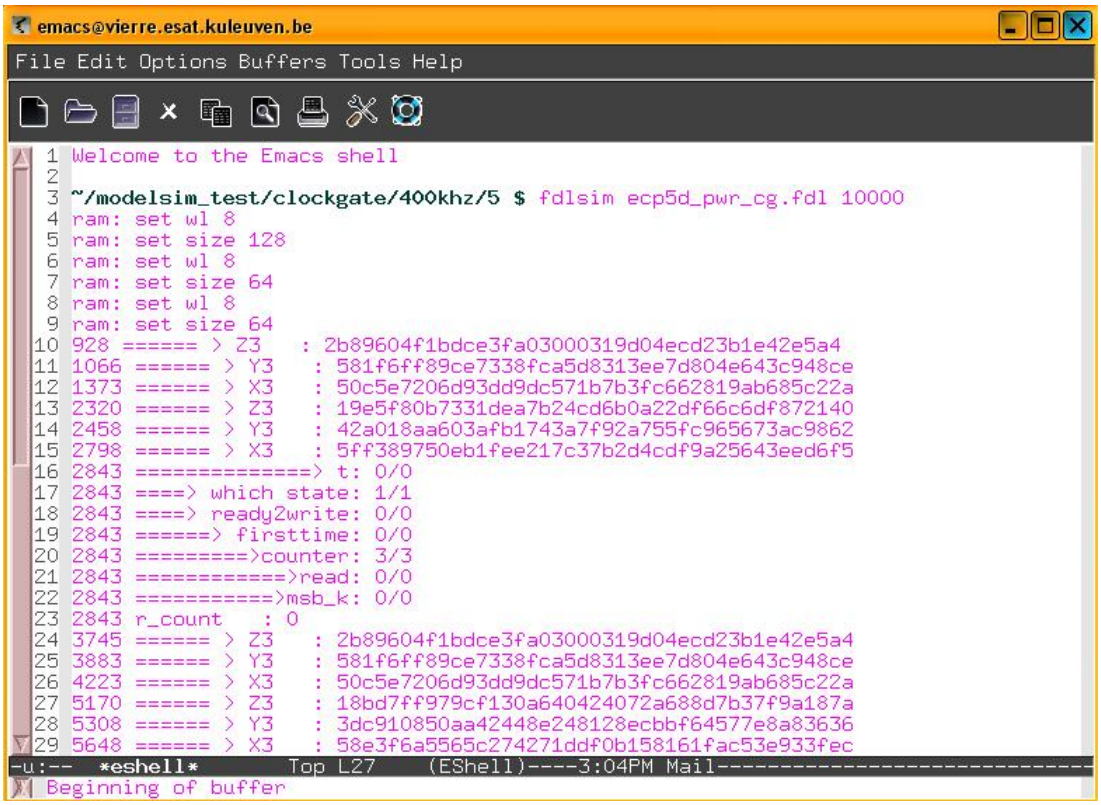

Figure E.2: Simulation in GEZEL for First Four Projective Addition

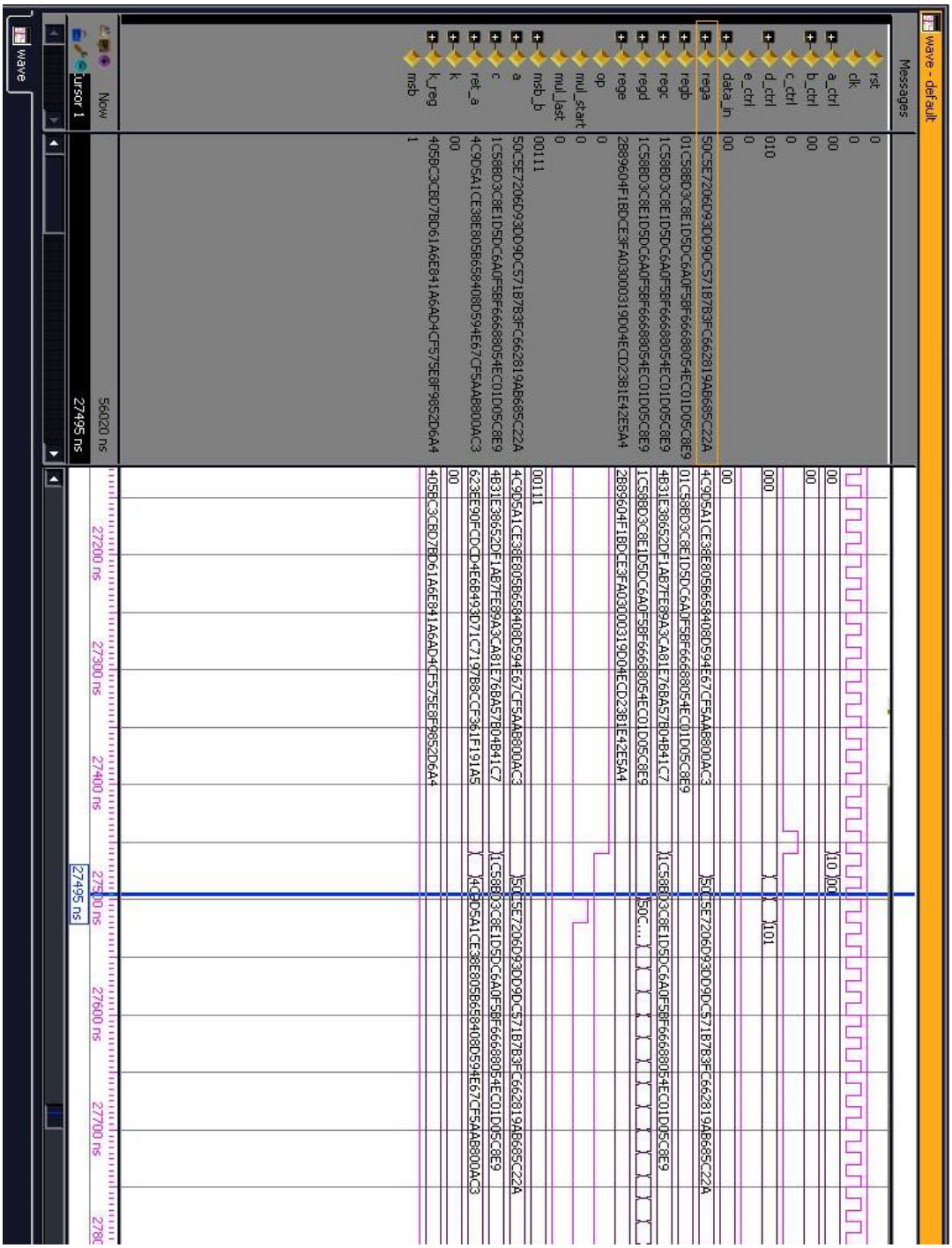

Figure E.3: Figure of Simulation in Modelsim for First X3 Value

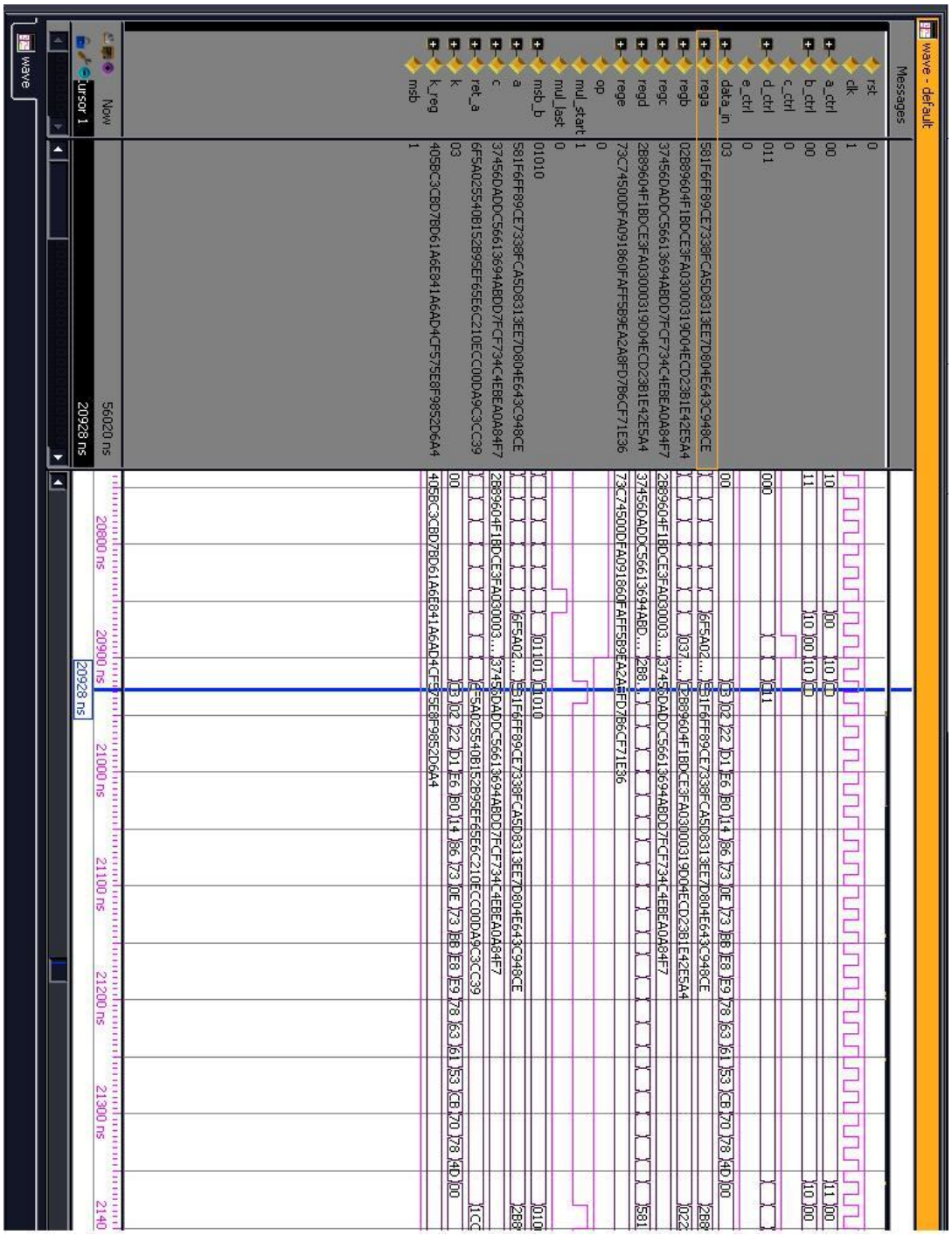

Figure E.4: Figure of Simulation in Modelsim for First Y3 Value

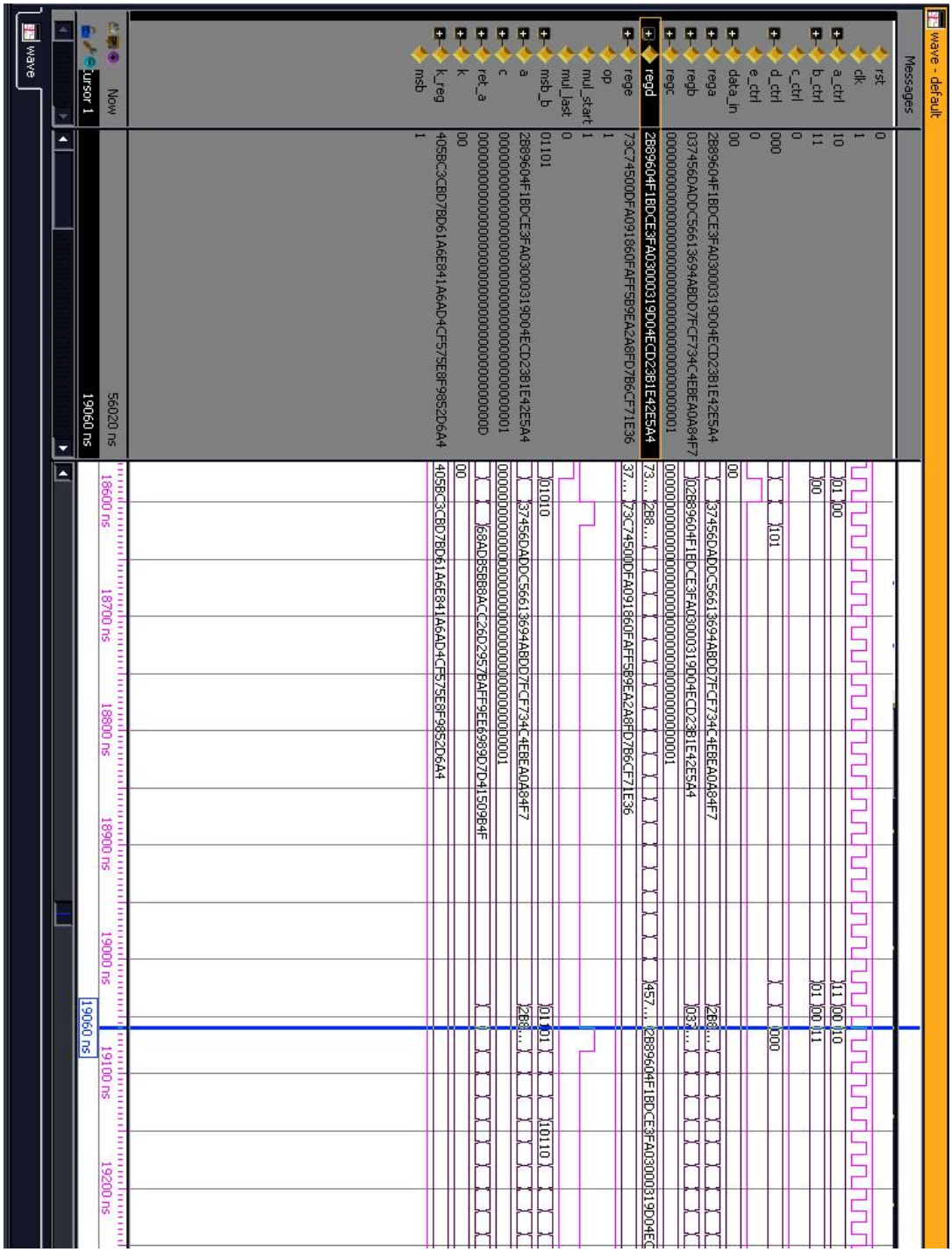

Figure E.5: Figure of Simulation in Modelsim for First Z3 Value

Figure E.6: Simulation in GEZEL for Final Points After Inversion

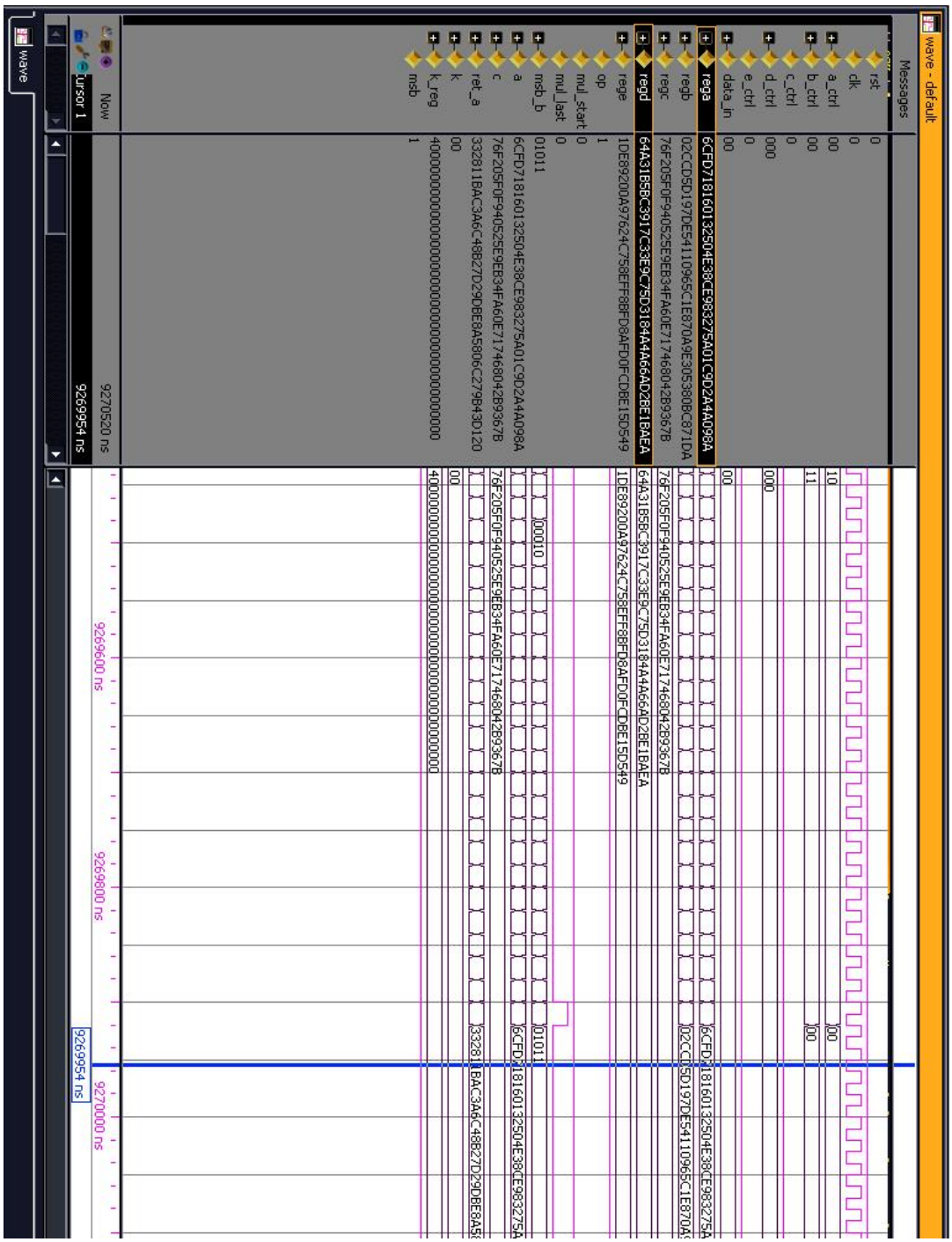

Figure E.7: Figure of Simulation in Modelsim for Final Points

CIRCULUM VITAE

Ünal Kocabaş was born in 1986 in Bursa, Turkey. He finished his primary, secondary and high school educations in Bursa. He received his bachelor degree in Electronics Engineering from Istanbul Technical University in 2007. In addition to bachelor degree, he studied and worked part-time for 9 months as an assistant of ASIC measurement department at the Mikroelekronik R&D Design Center in Istanbul, Turkey. Afterwards, he started his M.Sc. in Electronics Engineering in ITU in September 2007. In September 2008, he started his master thesis in the COSIC group (Computer Security and Industrial Cryptography) in the Department of Electrical Engineering (ESAT) of the Katholieke Universiteit Leuven, as an exchange student.