HARDWARE IMPLEMENTATION OF FAST ADDERS FOR MONTGOMERY MULTIPLIER IN ELLIPTIC CURVE CRYPTOGRAPHY

A Thesis Submitted in Partial Fulfillment of the Requirement for the Award of the Degree of

MASTER OF TECHNOLOGY

in VLSI Design

Submitted By

ADITYA SHANKAR

601662002

Under Supervision of

Mrs. Manu Bansal

Assistant Professor

THAPAR INSTITUTE
OF ENGINEERING & TECHNOLOGY
(Deemed to be University)

ELECTRONICS AND COMMUNICATION ENGINEERING DEPARTMENT

THAPAR INSTITUTE OF ENGINEERING & TECHNOLOGY (A DEEMED TO BE UNIVERSITY), PATIALA, PUNJAB JUNE, 2018

DECLARATION

I, Aditya Shankar hereby declare that the work presented in this thesis entitled "Hardware Implementation of Fast Adders for Montgomery Multiplier in Elliptic Curve Cryptography" in fulfillment of the requirement for the award of degree of Master of Technology (VLSI Design) submitted at Electronics and Communication Dept., Thapar Institute of Engineering & Technology (Deemed to be University), Patiala is an authentic record of work carried out under supervision of Mrs. Manu Bansal (Assistant Professor, ECED, Thapar Institute of Engineering and Technology from July, 2016 to July, 2018. The matter presented in this has not been submitted either in part or full to any other university or institute for the award of any other degree.

Aditya

Date: 09/08/17

Aditya Shankar
601662002

It is certified that the above statement made by the candidate is correct to the best of my knowledge and belief.

Manu Bansal

Date: 09/08/17

Mrs. Manu Bansal
Assistant Professor, ECED
Thapar Institute of Engineering & Technology
(A Deemed To Be University), Patiala, Punjab

ACKNOWLEDGEMENT

I take this opportunity to express my profound sense of gratitude and respect to all those who helped me through the duration of this dissertation. I acknowledge with gratitude and humility my indebtedness to **Mrs. Manu Bansal, Assistant Professor,** Electronics and Communication Engineering Department, Thapar Institute of Engineering and Technology, Patiala, under whose guidance I had the privilege to complete this dissertation. I wish to express my deep gratitude towards her for providing individual guidance and support throughout the dissertation work. I convey my sincere thanks to **Head of the Department, Dr. Alpana Aggarwal, ECED,** entire faculty and staff of Electronics and Communication Engineering Department for their encouragement and cooperation.

My greatest thanks is to all who wished me success especially my family. Above all I render my gratitude to the Almighty who bestowed ability and strength in me to complete this work.

Aditya Shankar

601662002

ABSTRACT

The Internet of Things network connects the number of devices and gains popularity in the number of applications for transmitting or sharing information. The malicious devices additions to the IoT network disturbs the whole network and steal the user's information. Hence, device authentication is a big concern. To resolve these issues, asymmetric cryptography algorithms are used. In the previous years, the RSA algorithm gained popularity and provide security in large key size which increases the computation time and memory requirement. To overcome these issues, NIST recommended Elliptic Curve Cryptography (ECC) algorithm for data authentication.

In this thesis, the mathematics of ECC algorithm is studied. Based on the studies, it is found that ECC is categorized into the prime and binary field. The prime field is more preferred as compared to the binary field because the requirement to propagate carries is removed and fast inversion algorithms exists. Further, to improve the performance of the ECC algorithm, many multipliers such as Montgomery, Karatsuba, NTT, Comba and their combinations are used. Out of these multipliers, Montgomery is best suitable for fast implementation for ECC due to the fact that it performs both multiplication and modular function simultaneously. Next, to improve the performance of Montgomery multiplier, various parallel prefix adders are studied and among them Kogge-Stone adder is selected because of least delay. The software implementation is done on GCC compiler on LEON-3 processor and hardware implementation is done on Xilinx Vivado for FPGA Basys3 board. The software performance analysis is done on the basis of latency, Cycles per Instructions and hardware performance analysis is done on the basis of slices, maximum frequency, and throughput. In hardware implementation, when comparing ECC with Montgomery multiplier and Kogge-Stone adder with the basic ECC, performance was improved by 15.9 %.

TABLE OF CONTENTS

LIST OF TABLES

S. NO. **NAME OF TABLE** **PAGE NO.**

LIST OF FIGURES

S. NO. **NAME OF FIGURE** **PAGE NO.**

LIST OF ABBREVIATIONS

ECC	Elliptic Curve Cryptography
RSA	Rivest Shamir Adlemman
ECDH	Elliptic Curve Diffie Hellman
CSKA	Carry Skip Adder
MM	Montgomery Multiplication
DLP	Discrete Logarithmic Problem
DoS	Denial of Service
PKC	Public Key Cryptography
VLSI	Very Large Scale Integration
NTT	Number Theoretic Transform
LUT	Look Up Table
FPGA	Field Programmable Gate Array
NIST	National Institute of Standards and Technology
ASIC	Application Specific Integrated Circuit

CHAPTER 1

INTRODUCTION

Internet of Things (IoT) is basically connection of several sensors, devices, etc. connected to each other through the internet. There are many applications of IoT such as Smart City, Smart Homes, E-Commerce, Agriculture, Energy and Resource management. But the popularity of E-commerce and Smart City is higher when compared to other applications. As the development in the field of computer network grows, an ever-increasing number of clients get to the remote server's administration in a dispersed computing environment. Electronic exchanges are gaining popularity as it is generally acknowledged in the Internet processing world. A wide range of organizations have moved into the field of business over internet rather than following the traditional way. Also, Smart City is one of the applications which are currently being researched and implemented upon. In modern era, every city is moving towards being a smart city. The idea of smart city is to collect data from various sensors and then analyze or process them to get information regarding basic city aspects like hospitals, power plants, traffic management, water supply network, etc. Smart Grids are also an important part of Smart cities. Smart grid is collection of smart meters, smart appliances, etc. It is like a digital control of electricity over its production, distribution, management, etc. As the number of nodes in IoT network increases, there are various threats during the transfer of the data from source to destination like eavesdropping, intended modification, impersonation, Denial of Service (DoS) etc. [1] To overcome these threats, various security measures needs to be implied and factors like data security and authenticity comes into picture, for the sensitive data. To provide security and authentication in the IoT network different cryptographic techniques are employed.

1.1 Cryptography

Cryptography is basically the method of transmitting information through a secure way. Basic as well as complex Mathematics are involved in cryptography. At transmitter side, the intelligent form of information is first converted into unintelligible format and then it is passed to the receiving end where is it converted back to intelligent form. As for the counter-part, cryptanalysis is the way of breaking the secure communication.

The basic goal of cryptography is exchanging of information or data between parties which cannot be understood by any unwanted party. Figure 1.1 describes the basic example of cryptography. Alice is the sender of the message which encrypts the message and then transfers it while Bob is the receiver who receives the encrypted text and decrypts it to retain the original message. Eve can be termed as intruder or adversary who can get the information from the unsecured channel.

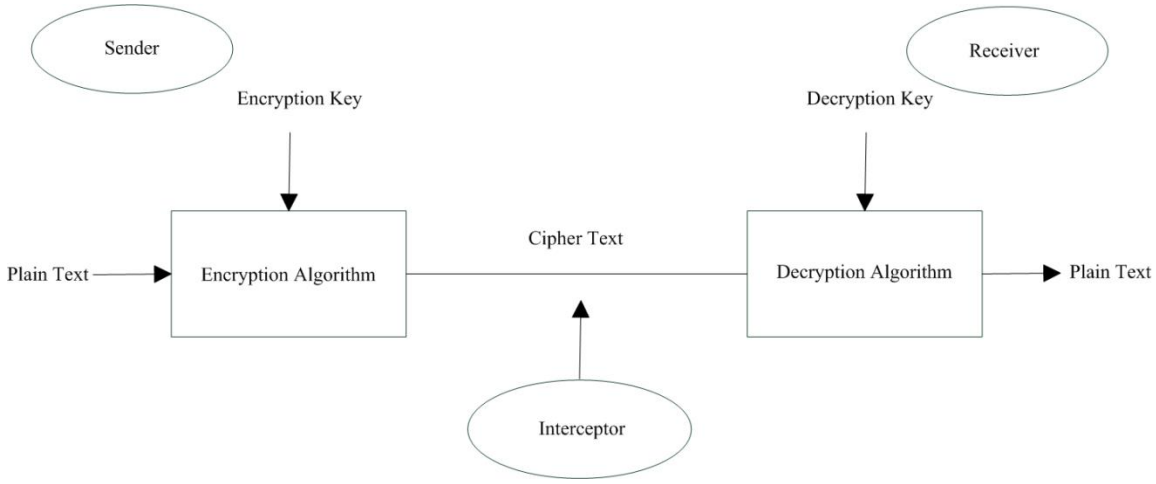

Figure 1.1 Basic Cryptography Flow

The strength of cryptography can be termed according the resources and time it takes to get back the plain text. There are several algorithms which does the encryption and decryption process in their own different manners. These algorithms use Key to encrypt and decrypt the data. Data security depends on the strength of the cryptographic algorithm along with the key confidentiality.

1.2 Characteristics of Cryptography

Cryptography basically works on the following four components. With a specific end goal to secure messages, there are numerical systems that give security administrations, for instance, confidentiality, integrity, authentication and non-repudiation. [2]

Confidentiality: It is basically encoding of the data so that only the desired party can access or understand the data.

Integrity: It is the property by which the data can't be altered by any third party. Only the sender or receiver has the right to change the data. Basically, the data is held and protected at both sender and receiver's end.

Non repudiation: It is a measure to ensure that if the sender has send the data to receiver then receiver can't deny that the data was not sent or not received.

Authentication: It is a measure to verify the identity of the sender or receiver i.e. if the data is received at the receiver, then receiver can verify that the data was actually sent from the intended sender.

1.2.1 Types of Cryptography

Algorithms in Cryptography are classified into basically three types namely, symmetric, asymmetric and hash algorithms. Hash algorithms are usually intended for increasing the permutations of the data. This section gives a brief idea about symmetric and asymmetric key cryptography.

- *Symmetric-Key Cryptography (Secret-Key Cryptography):* If a cryptographic algorithm uses the same key to encrypt as well as decrypt the data then it is termed as symmetric key algorithm which in turn is called symmetric key cryptography. The only disadvantage of this type of cryptography is that parties have to share their keys before the communication. Symmetric key cryptography is shown in Figure 1.2:

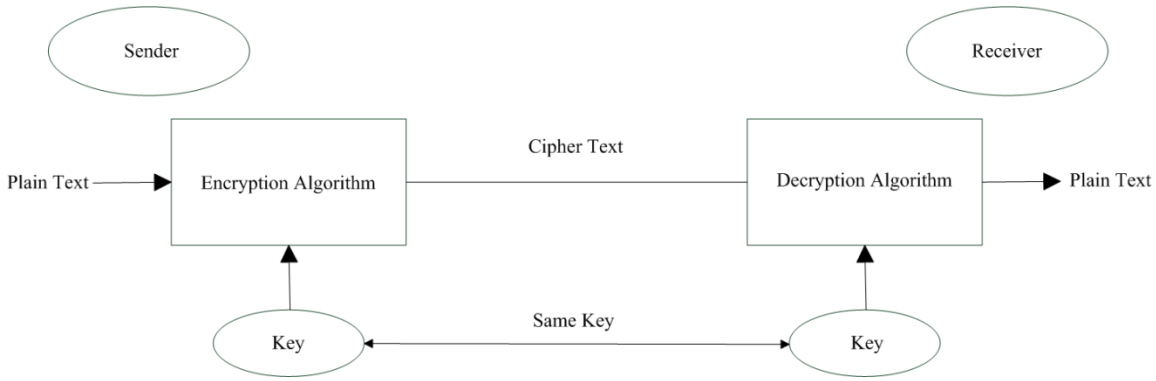

Figure 1.2 Symmetric Key Cryptography

There are two types of ciphers: Stream Cipher and Block Cipher. In the Stream Ciphers, operations are done on a single bit (byte) of data at a time and some form of feedback function is implemented in order to keep the key changing. Here same plain text will encrypt to create a different cipher text. On the other side, Block ciphers divides plaintext (input) into blocks of bits of same length and encrypt that each block using same key. Here same block of plain text will create same block of cipher text when same key will be used on them. Modern Block Ciphers are of iterative nature. They perform a function number of times. This function has linear and non-linear layers, which changes the data on every iteration using round keys generated from main secret key. This mode of encryption is very efficient while sending large amount of data but lacks when communication has to be made between multiple parties as the secret key need to be shared with every recipient.

There are many advantages of symmetric key cryptography like [3],

- The process of encryption is simple, fast and computational power is low.
- One is needed for both encryption and decryption and they are relatively short.
- Data throughput rates are high.

There are some disadvantages to this system like,

- *Key establishment and distribution*: Before starting the communication between two parties, both of them should agree on a key and share it via a secure mechanism.
- *Scalability*: Number of keys required as compared to the number of participants in the message exchange equals about the square of the number of participants.

- *Authentication*: Symmetric algorithms cannot be used for Digital Signatures.

- *Asymmetric-Key Cryptography*: Asymmetric key cryptography has two keys in its procedure. One is the public key which is for the encryption of the data and other is private key which is for the decryption of the data. The benefit of the algorithm is that the private key is never shared with any party. Hence it is also known as Public Key Cryptography. Asymmetric key cryptography is shown in figure 1.3.

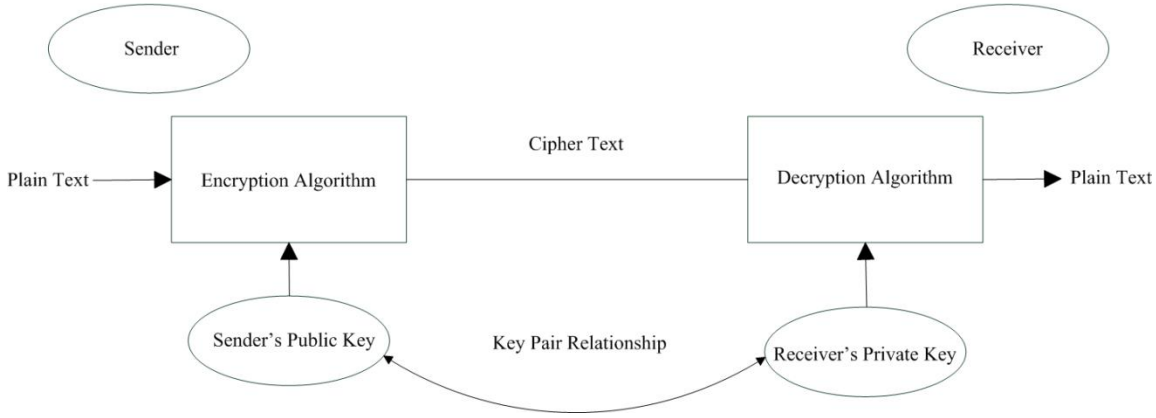

Figure 1.3 Asymmetric-Key Cryptography

In this encryption process different keys are used for encrypting and decrypting data. These keys are mathematically related but knowledge of one key will not lead to the other key easily. One key encrypts and other decrypts data and the order of the key used does not but both are needed to carry out the process. One key is called private key which is not shared with anyone and other is public key which is shared over the medium. It depends upon mathematical functions which are easy to calculate but their inverse function is comparatively hard to compute, example factorization and logarithms. In case of factorization, it is easy to calculate the product of two prime numbers but if only the number is given calculating its prime factors is relatively more difficult. This feature makes Asymmetric key cryptography more secure. It also provides to solution to problems such as key distribution and management and provision of non-repudiation [4].

Advantages of asymmetric key cryptography are:

- *Highly secure mechanism*: Their complex calculation makes it difficult to break them.
- *Secure Key distribution*: Sharing of keys over the channel becomes secure as using private key is kept a secret and without it the cipher text can never be obtained.
- *Authentication*: Asymmetric ciphers provide authentication using Digital signatures.

Disadvantages of public-key encryption:

- Data throughput rates of most popular public-key encryption methods are several orders slower than the best-known symmetric-key schemes.
- Key sizes are much bigger than those used in symmetric-key encryption. The size of the signatures is bigger than what tags can handle. It provides data origin authentication in symmetric-key techniques.
- There is no fully secure public-key scheme (as well as block ciphers). Most of the current public-key encryption schemes have their security strength. It is based on the presumed difficulty of a set of number-theoretic problems.

1.3 Different Types of Asymmetric Cryptography

1.3.1 Diffie Hellman

Diffie and Hellman proposed a type of public key cryptography in 1976. The Diffie Hellman keys are used to create single shared key between units as well as for encryption and decryption [5].

Let's assume that Alice want to transfer data to Bob. Both Alice and Bob agrees on a large prime number p and its primitive root g. Now Alice computes $A = g^a \ (mod\ p)$ and Bob computes $B = g^b \ (mod\ p)$. Now Alice and Bob exchange the values over the channel. After receiving the values, Alice computes $A^{'}=B^a \ (mod\ p)$ and Bob computes $B^{'}=A^b \ (mod\ p)$.

$$A^{'} \equiv B^a \equiv (g^b)^a \equiv g^{ab} \equiv (g^a)^b \equiv A^b \equiv B^{'} \ (\text{mod } p) \qquad (1)$$

This common value is their exchanged key.

The following diagram shows the basic flow of Diffie Hellman Key Exchange:

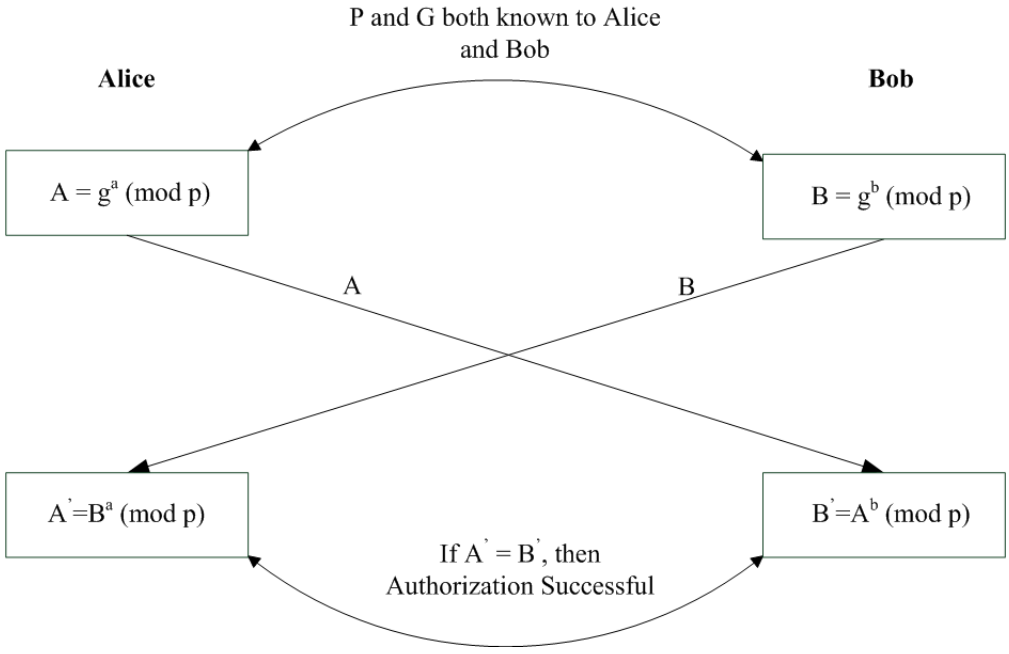

Figure 1.4 Diffie Hellman Key Exchange

The above key exchange can be understood from the following example:-

Diffie Hellman Key Exchange
Let, p=5 and g=3. This is a valid assumption since 3 is primitive root of 5. Let the secret key of Alice be, a=4 and secret key of Bob be, b=3. $A = g^a \pmod p = 3^4 \pmod 5 = 1$ $B = g^b \pmod p = 3^3 \pmod 5 = 2$ After the exchanging of values, Alice and Bob computes, $A' = B^a \pmod p = 2^4 \pmod 5 = 1$ $B' = A^b \pmod p = 1^3 \pmod 5 = 1$ So, $A' \equiv B^a \equiv (g^b)^a \equiv g^{ab} \equiv (g^a)^b \equiv A^b \equiv B' \pmod p = 1$ Hence, 1 is their exchanged key.

The basic is that it is based on Discrete Logarithm Problem. Hence, the mathematics is easier and also higher bits the primitive root has, more is the security. Disadvantages include the non-usability for digital signatures and highly prone to man-in-the-middle attack since there is no Authentication.

1.3.2 El-Gamal

Taher Elgamal proposed the El-Gamal encryption scheme in 1984. It is very similar to Diffie-Hellman key exchange and is based on discrete logarithm problem. The security of this cryptographic scheme depends on the difficulty of the discrete logarithm problem. It can be used for the pupose of generation of digital signatures as well as encryption [6].

Let's assume the situation that Alice wants to send information to Bob. Now, similar to Diffie-Hellman, there will be public parameter p (prime number) and g (primitive root of the prime number). Alice will compute $A = g^a \pmod p$, where a is Alice's private key. A is the public key of Alice. Bob will encrypt the data which is to send to Alice using Alice's public key. Let x be the message that Bob wants to communicate. Bob randomly chooses a number k modulo p, which is called an ephemeral key because it will be used only for the encryption of single message. Bob now computes two quantities, $y_1 = g^k \pmod p$ and $y_2 = xA^k \pmod p$. Bob transmit the values y_1 and y_2 to Alice. Now, Alice computes $z = y_1^a \pmod p$ and z^{-1}. Alice then multiplies z^{-1} with y_2 to get the value of x, which was the message sent from Bob. To prove this, expansion of the multiplication can be seen as follows:

$$
\begin{aligned}
z^{-1} \cdot y_2 &= (y_1^a)^{-1} \cdot y2 && \pmod p \\
&= (g^{ak})^{-1} \cdot (xA^k) && \pmod p \\
&= (g^{ak})^{-1} \cdot (x(g^a)^k) && \pmod p \\
&= x
\end{aligned}
$$

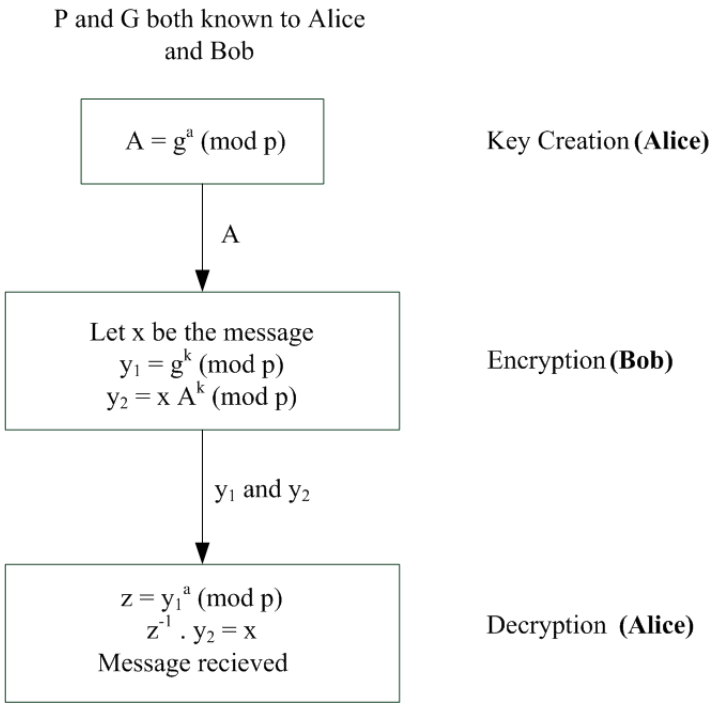

Figure 1.5 El-Gamal Key Exchange

The El-Gamal public key cryptosystem can be understood from the following example:

El-Gamal Key Exchange
Let the prime number, p be 5 and the primitive root, g be 8. Alice's secret key, a=3. So Alice computes: $A = g^a \pmod p = 8^3 \pmod 5 = 2$ Bob uses public key of Alice to encrypt his message. Let the message, x be 2. Here Bob generates an ephemeral key, k=6. So, Bob computes following two values: $y_1 = g^k \pmod p = 8^6 \pmod 5 = 4$ $y_2 = xA^k \pmod p = 2 \cdot 2^6 \pmod 5 = 3$ Bob sends these computed values to Alice. Alice decrypts it by the following operation: $z = y_1^a \pmod p = 4^3 \bmod 5 = 4$ $z^{-1} = 1 / 4 \pmod 5 = 4$ (Calculating Modular Multiplicative Inverse) $z^{-1} \cdot y_2 = 4 \times 3 \pmod 5 = 2$ where 2 is the message sent by Bob.

The main advantage of El-Gamal over Diffie-Hellman is that it is good for authentication. For encryption, it is not usually preferred. The disadvantage of El-Gamal is that, this scheme is more prone to Known-Plaintext attack.

1.3.3 RSA

RSA public key cryptosystem was proposed in 1978 by Ron Rivest, Adi Shamir and Leonard Adleman. It is based on factorization problem. The key used in RSA can be of 64-3072 bits. But this much bit length creates various issues in hardware implementation. So, if the target application is based over high constraints than using RSA wouldn't be that beneficial. Due to such length for the key, issues regarding computation time, power, bandwidth etc. RSA is used for encryption/decryption as well as for authentication purposes [7].

Let's assume that Alice wants to send data to Bob. First Bob selects two prime numbers for the generation of the public key, say p and q. Let $N = pq$, m be the message and e be the encryption exponent which is relatively prime to (p - 1) (q - 1). This N and e is transmitted to Alice as (N, e). Alice computes $c = m^e \pmod N$ and communicates it to Bob. To recover the message, Bob computes $m' = c \pmod N$.

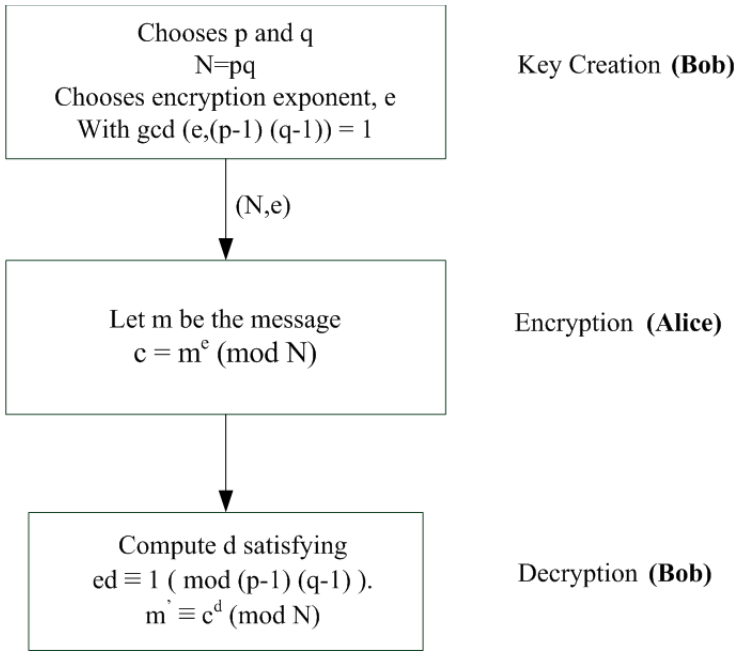

Figure 1.6 RSA Key Exchange

RSA cryptosystem can be easily understood with the following example:

RSA Key Exchange
Suppose that Alice chose p=5 and q=3. So, N= pq = 15. Now the encryption exponent, e is taken to be 11. The message to be transmitted is 6. Now, Alice computes $c = m^e \pmod N = 6^{11} \pmod{15} = 6$ c is communicated to Bob. Bob does the following calculation $e \cdot d \equiv 1 \pmod{(p-1)(q-1)}$ $11 \cdot d \equiv 1 \pmod 8$, so d = 11. Now Bob computes $m' = c^d \pmod N = 6^{11} \pmod{15} = 6$, which is the message sent by Alice.

The main advantage of RSA is that it is fast and easier to implement. The encryption is also simple and efficient. Though it does Authentication, but it has slow signing, decryption process and key generation is also slow when compared to more advanced asymmetric algorithm like ECC.

1.4 Organization of the Thesis

The main aim of this thesis is to understand and implement an asymmetric cryptographic algorithm on software and hardware and also to reduce its computational complexity. This thesis is divided into six chapters.

Chapter 1: Introduction

A brief idea about the need of authentication in various applications like wireless sensor networks, Smart Grid, IoT, etc. is described in this chapter. An introduction about cryptography and its types are also explained. Since this thesis focus on asymmetric encryption schemes, they are mathematically explained with their advantages and disadvantages along with some basic examples.

Chapter 2: Literature Survey

This chapter described the research and analysis of different authors in the area of asymmetric cryptographic algorithms regarding improving its performance.

Chapter 3: Problem Formation and Objectives

Based on the survey done, this chapter gives the issues or research gaps regarding the performance improvement and according to that objectives are defined.

Chapter 4: Overview of Algorithms used for Elliptic Curve Cryptography

Basic flow of Elliptic Curve Cryptography is described in this chapter. Working of Montgomery multiplier and comparison of various parallel prefix adders are also shown, which reduces the complexity and computation time in ECC.

Chapter 5: Results

Results of Software and Hardware implementation of ECC have been summarized using Leon3 processor, GCC 4.4.6 compiler and Xilinx Vivado 2016.4 respectively.

Chapter 6: Conclusion and Future Scope

The conclusion drawn from the literature survey is described in this chapter. Also based on the research gaps defined, some possible solutions and approaches are defined in the future scope section.

CHAPTER 2

LITERATURE SURVEY

Security aspects especially Authentication have become important part of daily life especially in business sector which leads to usage of asymmetric algorithms. Asymmetric algorithm like ECC is being implemented in many applications. To further improve the performance of ECC, various multipliers and adders have been studied.

Khalid Javeed, *et al*. [8], represented the problem of computation complexity in ECC and suggested a dedicated hardware for the computations in order to improve the performance. The hardware improves the performance by exploiting parallelism. Proposed hardware was implemented on various families of FPGA and results were compared in terms of computation time. For a 256-bit ECC implementation, proposed hardware resulted in 20,579 area in terms of slices, a computation time of 3.91 ms and a throughput of 65,473 bps. Also, a computation analysis of 3 different FPGA families was done. Virtex-6, Virtex-5 and Virtex-4 showed computation time of 2.01 ms, 2.62 ms and 3.91 ms, respectively, for 256-bit ECC scalar multiplication operation.

Reza Azarderakhsh, *et al.* [9], discussed about the important of Point Multiplication in ECC and its complexity. Author focused on binary Edward and hessian curves and proposed a high speed and efficient architecture to implement Point Multiplication. The approach was hybrid-double multiplier with the aim of reducing the latency and removing dependency of data. It was implemented on Virtex-4 for 163-bit ECC algorithm which resulted in 27,778 slices and computation time of 17.5 µs. The proposed architecture is 25% faster from the Author's previous implementation.

Darrel Hankerson, *et al.* [10], presented the idea of improving and verifying the ECC technique at software level. Authors suggested that the Koblitz curves were easy to implement and no modification for the multiplier was used. The implementation was done in C programming language on a Pentium II 400 MHz workstation. The software implementation resulted in 1176 µs and 2243 µs for 163-bit and 233-bit ECC implementation respectively, for random and Koblitz curves. Authors also implemented the authentication scheme using Elliptic Curve Digital Signature Algorithm (ECDSA). For ECDSA, Point Multiplication took 2702 µs and 5348 µs for 163-bit and 256-bit ECC implementation, respectively.

Nils Gura, *et al.* [11], compared three versions of Elliptic curves recommended by NIST, 163-bit, 192-bit and 224-bit with two RSA models of 1024-bits and 2048-bits. Authors also proposed a new algorithm to decrease the number to time the memory is accessed. The comparison was done on Atmel Atmega128 at 8 MHz and CC1010 at 14.74 MHz. ECC secp192r1 on Atmel Atmega128 resulted in a computation time of 1.24 seconds and 7.56 seconds on CC1010 while RSA 1024-bit implementation results in a computation time of 10.99 seconds on Atmel Atmega128 and 106.66 seconds on CC1010. The ECC implementations had a 25% increased performance than the RSA implementations.

M. Srinivasan, *et al.* [12], depicted the idea of an area efficient FPGA implementation of ECC in any finite field. Advantages of ECC was shown over traditional cryptographic schemes like RSA, AES, etc. with respect to power, area and resources. The ECC implementation of ECC on GF (2^{163}), GF (2^{233}), GF (2^{283}), GF (2^{409}) and GF (2^{571}) was done with bit serial multiplier on Virtex-4 FPGA board. The implementation with bit serial multiplier resulted in 33% speed up when compared to implementation with digit serial multiplier. For GF (2^{571}), the utilized slices on FPGA board were 10,296 and the device had a maximum frequency of 449.498 MHz.

Dindayal Mahto, *et al.* [13], introduced analysis of RSA (Rivest Shamir Adleman) and ECC (Elliptic Curve Cryptography). The security of the RSA cryptosystem depends on the Integer Factorization Problem (IFP) though the security of ECC depends on the elliptic curve discrete logarithm problem (ECDLP). The huge fascination towards ECC is that the best-known calculation for unraveling the ECDLP takes full exponential time while for comprehending IFP of. This analysis recommends that ECC takes less memory than RSA and is superior to RSA, particularly on devices which has constraint regarding their memory.

C. Rafferty, *et al.* [14], described the problem of multiplication of very large numbers in various applications like DSP, cryptography, etc. Author gives a thorough idea about different types of multipliers and also combines them to increase their performance. Multipliers like Comba, Karatsuba, Tom-Cook, Montgomery, and Number Theoretic Transforms (NTT) are analyzed on the basis of Latency, Slices and Maximum frequency. Author also described combination of multipliers like NTT-Comba-Karatsuba, NTT-Comba and their hardware complexity. The Montgomery Multiplier was the best for modular multiplication while NTT-Karatsuba-Schoolbook multiplier is better than others for very large number upto 16,384 bits.

Kee-Won Kim, *et al.* [15], focused on the need of high speed units for modular multiplication. The computation of modular arithmetic is very complex and time consuming. Authors proposed a combined algorithm to compute multiplication as well as squaring for GF (2^m) with low latency. This

algorithm uses Montgomery Multiplication algorithm just for conversion from one domain to another and is based on a bipartite structure. Due to its low latency, it can be used for various big data security operation.

Satyanarayana Vollala, *et al.* [16], proposed two modified forms of Montgomery Multiplication algorithm based on bit forwarding techniques. Authors target on the computational time and complexity of modular mathematics in applications. These algorithms are compared with existing ones based on throughput, power and energy. The two algorithms are named after the bits they forward in the computation, namely, 1-bit forwarding (BFW-1) and 2-bit forwarding (BFW-2). For 1024-bit numbers, the BFW-1 and BFW-2 showed an improvement of 11.02% and 15.13% in terms of throughput, 1.93% and 6.35% in terms of power and energy, respectively.

Kajal Agarwal, *et al.* [17], focused on the importance of multiplication operation and its need in various fields. Brief description and mathematics of the few generally used multipliers are described. There are different types of multipliers with their own advantages over other. Authors focuses on Array multiplier, Vedic multiplier, Booth's Multiplier, Wallace-Tree Multiplier and Karatsuba Multiplier. A review various publication involving these multipliers was done and a comparative analysis was shown. They were compared on parameters like Delay, Memory, Power and LUTs. Vedic Multiplier was suggested for various application especially DSP because of it low delay, low power, low memory and less LUTs when compared with other multipliers.

Khalid Javeed, *et al.* [18], proposed a 256 x 256 multiplier based on Montgomery Multiplication algorithm. Author mainly targets the computation complexity of asymmetric cryptography algorithms like RSA, ECC, etc. The addition of these 512-bits is optimized by using 64-bit chains which were inbuilt in the selected FPGA device (Virtex-6 Board). The comparison of proposed multiplier is done with Karatsuba multiplier and a splitting based multiplier on the basis of speed of the operation and DSP utilization on FPGA. Proposed multiplier works at 188 MHz frequency and uses 16 DSP Slices. Authors suggested the use of proposed multiplier in implementing Elliptic curve based cryptographic processors.

Aswathi Thomas, *et al.* [19], proposed an efficient multiplier algorithm based on Montgomery multiplier. This is then embedded to both RSA and ECC and a comparative analysis is done. Authors described about the delay in ECC due to heavy computations. Proposed algorithm, when implemented on RSA gives a 49.5 % decrement in area in terms of slices, 29.87 % reduction in delay in terms in minimum delay and 1.28 % reduction in terms of Power. When implanted on ECC, parameters are measured in two major processes in ECC namely, Point Addition and Point Doubling. For Point Addition, proposed algorithm reduces area by 38 % in terms of slices, 31.2 % reduction in delay and

1.21 % for power. For Point Doubling, proposed algorithm gives 32 % reduction in area and 38.45 % reduction in delay, in terms of minimum delay.

Shahzad Asif, *et al.* [20], suggested that the modular arithmetic should be performed in the Residue Number System (RNS). Proposed architecture has two variants. One is pipelines based and other is non-pipelined. Both architectures were implemented on both FPGA and ASIC for broader analysis. It was implemented on Vitex-6 Boards which resulted in cycle time of 14.20 ns and a throughput of 18028.2 Mbps for 2-stage pipelined architecture while for non-pipelined architecture, cycle time was 47.25 ns and throughput was 5120 Mbps. There was 3.6 % improvement in the proposed architecture when compared to existing ones.

Zoya Dyka, *et al.* [21], proposed an iterative based approach for Karatsuba multiplier method. This was done focusing on gaps in implementation of cryptosystems like RSA, ECC, etc. These include the computation cost, time and area. This approach had significant decrease in area (from 6.2 mm^2 to 2.1 mm^2). The approach showed effective results in terms of area and power while latency of the overall circuit was increased. Authors proposed their future work of the same type of implementation for Montgomery multiplication as it was considered better for cryptographic applications due to its ease in modular division and inversion properties.

William N. Chelton, *et al.* [22], described points of interest the plan of another fast pipelined application-specific instruction set processor (ASIP) for elliptic curve cryptography (ECC) utilizing field-programmable gate array (FPGA) innovation. Diverse levels of pipelining were connected to the information path to investigate the subsequent exhibitions also, locate an ideal pipeline depth. Three complex instructions were utilized to decrease the inactivity by diminishing the general number of directions, and another consolidated calculation was produced to perform point multiplying and point expansion utilizing the application particular guidelines. An implementation of NIST curve over GF(2^{163}) is shown, which accomplishes a point multiplication time of 33.05 s at 91 MHz on a Xilinx Virtex-E FPGA. Utilizing the more current Xilinx Virtex-4 innovation, a point multiplication time of 19.55 s was accomplished.

S. Daphini, *et al.* [23], described about the role of digital adder in any VLSI design and reviewed some work on parallel prefix adders like Brent-Kung, Kogge-Stone, Han-Carlson and Ladner Fischer. The working of adders is explained and they are compared on the basis of parameters like area, power and delay. It was concluded that Kogge-Stone adder was the fastest among all but has highest area (2669.9 μm^2) and power consumption (137.22 µW) while Brent Kung had the highest delay (825.9 ps) but least power consumption (85.08 µW).

Richard F. Hobson [24], described the importance of parallel prefix adders and proposes a 64-bit Multi Valence Adder (MVA) which helps in balancing fan-out and wire load. It greatly effects the energy efficiency when compared to structures of other parallel prefix adders like Kogge-Stone and Sklansky Adder. The proposed adder shows a 40 % increase in energy efficiency when compared with two versions of Kogge-Stone adder and 25 % reduction in delay logic when compared to a Sklansky based adder design.

M. Manasa, *et al.* [25], implemented a Carry Skip Ahead Adder (CSKA) structure that has a higher speed and lower energy utilization contrasted and the basic ones. The speed upgrade is accomplished by applying link and incrementation plans to enhance the effectiveness of the ordinary CSKA structure. Also, rather than using multiplexer logic, the proposed structure makes utilization of AND-OR-Invert (AOI) or potentially OR-AND-Invert (OAI) compound gates for the skip logic. The structure might be acknowledged with both settled stage size and variable stage measure styles. Also, a cross breed variable latency augmentation of the proposed structure, which brings down the power utilization without affecting the speed, was introduced.

CHAPTER 3

PROBLEM FORMATION

From the survey, it is found that asymmetric algorithms (Diffie Hellman [5], RSA [6], ECC [26]) has played important role for generating secure channel and for authentication purposes. The ECC algorithm gained popularity because of its small key size, less area or hardware and less computation time [26] as compared to RSA algorithm.

In the ECC algorithm, multiplication and modular functions are two major factors that leads to high computation time. In the literature, number of multipliers such as Montgomery, Karatsuba Multipliers are employed. The Montgomery multiplier has gain popularity when compared to the multipliers because it computes multiplication and modular function simultaneously.

Next, to improve the performance of Montgomery Multiplier, various adders are used, such as Carry Look-Ahead adder. These adders improves performance but increases hardware or power factor. Hence, in the review, study of various fast adders is done and further they are used in Montgomery multiplier for improving the performance. Based on this, following objectives are defined:

- Study and analyze various multipliers and adders for performance improvement in ECC algorithm.
- Hardware implementation of fast adders for Montgomery Multiplier which can be further used in ECC algorithm.
- Performance and comparative analysis of basic ECC with implementation of ECC with multiplier and adder.

CHAPTER 4

OVERVIEW OF ALGORITHMS USED FOR ELLIPTIC CURVE CRYPTOGRAPHY

Elliptic curves were used by Neal Koblitz and Victor Miller for the purpose of designing public key cryptosystems in 1985 [25]. Elliptic Curve Cryptography have become challenging and point of interest for many researchers due to its level of security with shorter key length when compared to other cryptographic algorithms. These short key length provide less hardware complexity, less power and increases overall performance.

4.1 Mathematics of Elliptic Curve Cryptography

Basic arithmetic of Elliptic Curve is required to understand the security aspect of Elliptic Curve Cryptography. A basic equation of an elliptic curve is $y^2 = x^3 + ax + b$ where the condition of non-singularity is $4a^3 + 27b^2 \neq 0$. From this equation, different elliptic curves can be obtained by changing the values of 'a' and 'b'. For any two points *'P'* and *'Q'* on the elliptic curve, there is a $kP = Q$, where *'k'* is a scalar. In ECC, multiplication of a scalar quantity with an existing point on elliptic curve yields into another point that lies on the curve. Hence, multiplication is a very basic operation in ECC. Based on this scalar multiplication in ECC, the strength of ECC is based on the difficulty of the Elliptic Curve Discrete logarithm problem. The ECDLP is basically the problem to determine an integer n such that $Q=n\,P$. The following are some basic examples of elliptic curves along with their graphs:

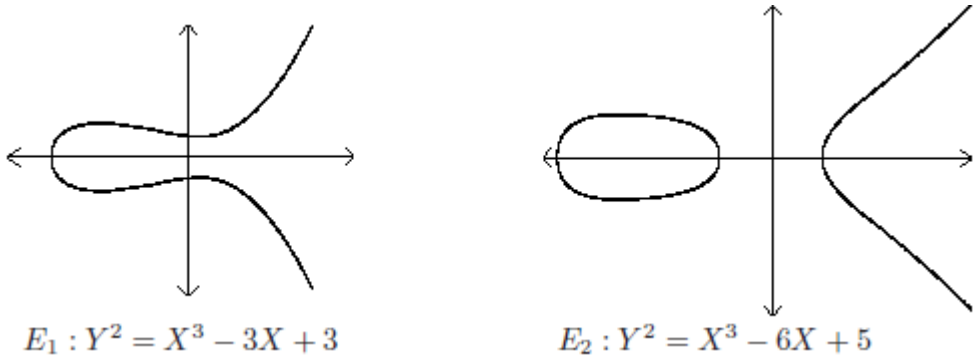

Figure 4.1 Types of Elliptic Curves based on Values of A and B [29]

4.1.1 Point Multiplication

Scalar point multiplication is the basic operation in Elliptic Curve Cryptosystem. It is an operation of the form $k.P$, where '*P'* is a point on the elliptic curve and *'k'* is a positive integer. This multiplication results into a Point *'Q'* which lies on the elliptic curve. There are two basic operations in Point Multiplication:

- Point addition (add two point to find another point)
- Point doubling (adding point p to itself to find another point)

For example to calculate $Q = kP$ if *'k'* is 31, then this computation can be broken down into:

$kP = 31P = (2(2(2(2P) + P) + P) + P) + P$

This is repetitive usage of Point Addition and Point Multiplication.

4.1.2 Point Addition

Point Addition in Elliptic Curve is addition of two points on elliptic curve to obtain another point on that curve. It is done by the following computation:

Consider an Elliptic Curve,

$$Y^2 = X^3 + AX + B \quad (1)$$

And let P1 and P2 be the points that lies on the curve. Now, if

$P_1 = 0$, then $P_1 + P_2 = P_2$.

$P_2 = 0$, then $P_1 + P_2 = P_1$.

$P_1 = (x_1 , y_1)$ and $P_2 = (x_2 , y_2)$.

If $x_1 = x_2$ and $y_1 = -y_2$, then $P_1 + P_2 = 0$.

Else

$$\lambda = \begin{cases} \frac{y_2 - y_1}{x_2 - x_1}, if\ P_1 \neq P_2 \\ \frac{3x_1^2 + A}{2y_1}, if\ P_1 = P_2 \end{cases} \quad (2)$$

then calculating the co-ordinates of the new point x3 and y3 as:

$x_3 = \lambda^2 - x_1 - x_2$ and (3)

$y_3 = \lambda (x_1 - x_3) - y_1$ (4)

Hence, $P_1 + P_2 = (x_3, y_3)$.

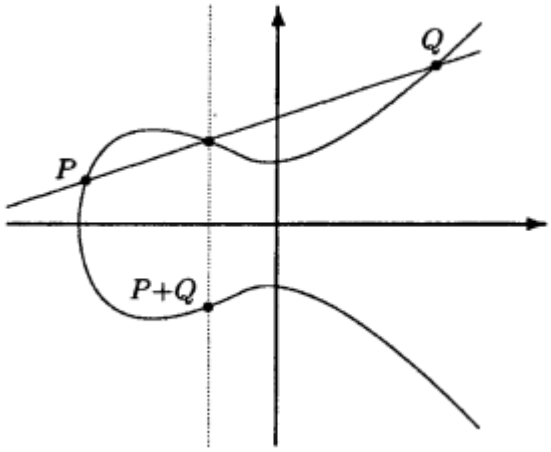

Figure 4.2 Point Addition [29]

Figure 4.2 shows addition of two points on elliptic curve graphically. If P_1 and P_2 are two points on the elliptic curve, then the line joining the two points will intersect the elliptic curve at some point,

say -Q. From –Q, if a straight line is drawn perpendicular to X-axis, then the intersection of the elliptic curve and line drawn will be the addition of points P_1 and P_2. So, $Q = P1 + P2$.

4.1.3 Point Doubling

Point Doubling is basically adding a Point to itself. Consider the Elliptic Curve, $Y^2 = X^3 + AX + B$ and $P(x_1, y_1)$ be the Point that lies on the elliptic curve. Double of this point can be computed using the following equations:

$$\lambda = \frac{3x_1^2 + A}{2y_1} \tag{5}$$

Let $Q(x_2, y_2) = P + P$

$$x_2 = \lambda^2 - 2x_1 \tag{6}$$

$$y_2 = \lambda (x_1 - x_2) - y_1 \tag{7}$$

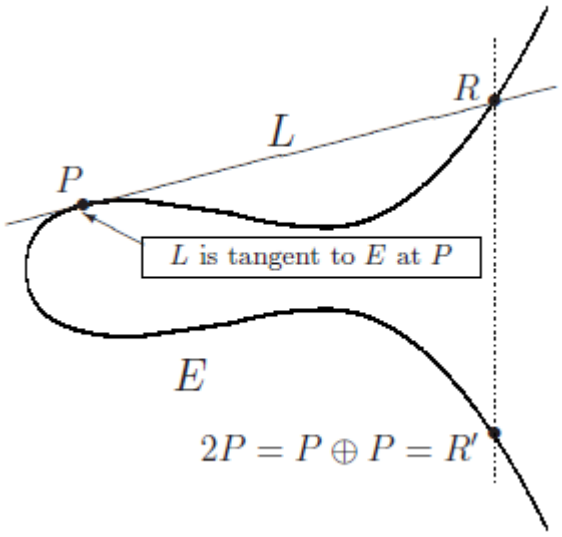

Figure 4.3 Point Doubling [29]

Figure 4.3 shows doubling of any point on elliptic curve, graphically. Say, P is a point that lies on elliptic curve. If a tangent is drawn at that point, then the tangent will intersect the elliptic curve at some point, say –Q. The perpendicular drawn from –Q to X-axis will intersect the elliptic curve at some point, which will be the double of point P. So, Q = P + P = 2P.

4.2 Elliptic curve over Finite Fields

A finite field is a set of finite number of elements. The number of units in the finite field is determined by the field order which is based on a prime number or powers of prime number. For cryptographic application, elliptic curve cryptosystem needs to be defined under a certain range. Basically ECC operated on finite fields for better accuracy and efficiency. There are three basic levels in ECC hierarchy. It consists of Cryptographic protocols, ECC operations which includes Point

addition and Point Doubling and Galois field operations. Finite fields are further classified into Prime and Binary Fields for Elliptic Curve Implementation represented by GF (P) and GF (2^P).

4.2.1 Elliptic Curve on Prime field

In order to achieve Elliptic curve cryptosystem within a finite field, the general equation of elliptic curve becomes, Y^2 (mod P) = X^3 + AX + B (mod P), where $4A^3 + 27B^2 \bmod p \neq 0$. When applying the modulus operation the range is restricted from 0 to P – 1, hence it can be termed as finite field for an elliptic curve.

The prime number *'P'* is taken to be very large in order to make the cryptosystem more secure. Now, to understand Point Addition and Point Multiplication, section 4.2.1.1 describes the flow of the processes.

4.2.1.1 Point Addition

Point Addition for finite field GF(P) is calculated according to the following equations. Here GF stands for Galois Field for any prime number 'P'.

Consider an Elliptic Curve,

$Y^2 \ (mod\ P) = X^3 + AX + B\ (mod\ P)$, where P=23, A=1 and B=1.

Let P_1 (4, 0) and P_2 (6, 4) be the points that lies on the curve.

$$\lambda = \begin{cases} \frac{y_2 - y_1}{x_2 - x_1}, if\ P_1 \neq P_2 \text{ (mod P)} \\ \frac{3x_1^2 + A}{2y_1}, if\ P_1 = P_2 \text{ (mod P)} \end{cases}$$

For this equation, $\lambda = \frac{4-0}{6-4} \text{ (mod 23)} = \frac{4}{2} \text{ (mod 23)} = 2$

Now, calculating the co-ordinates of the new point x3 and y3 as:

$x_3 = \lambda^2 - x_1 - x_2 \text{ (mod P)} = 2^2 - 4 - 6 \text{ (mod 23)} = -6 \text{ (mod 23)} = 17$, and

$y_3 = \lambda (x_1 - x_3) - y_1 \text{ (mod P)} = 2 (4 - 17) - 0 \text{ (mod 23)} = -26 \text{ (mod 23)} = 20$

Hence, $P_1 + P_2$ = (17, 20), which lies on the elliptic curve.

4.2.1.2 Point Doubling

Similar to Point Addition, Point doubling also gets in the range by using modular operation. Consider this example for understanding the Point doubling in Finite field.

Consider the Elliptic curve,

$Y^2 \ (mod\ P) = X^3 + AX + B\ (mod\ P)$, where P=23, A=1 and B=1.

Let P_1 (11, 3) be the point that lies on the elliptic curve.

$$\lambda = \begin{cases} \frac{y_2 - y_1}{x_2 - x_1}, if\ P_1 \neq P_2 \text{ (mod P)} \\ \frac{3x_1^2 + A}{2y_1}, if\ P_1 = P_2 \text{ (mod P)} \end{cases}$$

From this equation,

$$\lambda = \frac{3*11^2+1}{2*3} \pmod{23} = \frac{364}{6} \pmod{23} = 7$$

Now, calculating the co-ordinates of the new point x3 and y3 as:

$x_2 = \lambda^2 - 2x_1 \pmod{P} = 7^2 - 2*11 \pmod{23} = 49 - 22 \pmod{23} = 27 \bmod 23 = 4$

$y_2 = \lambda (x_1 - x_2) - y_1 \pmod{P} = 7 (11 - 4) -3 \pmod{23} = (7*7) - 3 \pmod{23} = 46 \bmod 23 = 0$

Hence, P + P = 2P = (4, 0), which lies on the elliptic curve.

4.2.2 Elliptic Curve on Binary field

The equation for implementation of Elliptic curve cryptography in binary field is $y^2 + xy = x^3 + ax^2 + b$, where $b \neq 0$. The field order for binary field is given by any prime or power of prime which is 'm' bits. So the degree of the polynomial can vary from 0 to m-1. The value of 'm' is taken very large for higher security of the cryptosystem.

4.2.2.1 Point Addition

Consider the elliptic curve, $y^2 + xy = x^3 + ax^2 + b$ (8)

Let P (x_1, y_1) and Q (x_2, y_2) be the two points on elliptic curve. Slope is calculated by the following formula:

$$\lambda = \frac{y_1 + y_2}{x_1 + x_2} \quad (9)$$

Now, calculation of co-ordinates of the resultant point is done as follows:

$x_3 = \lambda^2 + \lambda + x_2 + a$, and (10)

$y_3 = \lambda (x_1 + x_3) + x_3 + y_2$ (11)

Hence, Q = P + Q = (x_2, y_3).

4.2.2.2 Point Doubling

Consider the elliptic curve, $y^2 + xy = x^3 + ax^2 + b$.

Let P (x1, y1) be the point lying on Elliptic curve. Slope is calculated by the following formula:

$$\lambda = \frac{x_1 + y_1}{x_1} \quad (12)$$

Now, calculation of co-ordinates of the resultant point is done as follows:

$x_3 = \lambda^2 + \lambda + a$, and (13)

$y_3 = x_1^2 + (\lambda + 1) x_3$ (14)

Hence, Q = 2P = (x_3, y_3).

Above examples are very basic and can be easily solved. But in real applications of Cryptography, modular exponents can reach up-to 256 bits. There are various techniques or different multipliers

which performs both multiplication and modulus simultaneously. This thesis focus on Montgomery modular multiplier.

4.3 Montgomery Modular Multiplication

Exponentiation is an important aspect in cryptographic algorithms. One of the most commonly known modular exponential multiplier is Montgomery multiplier. The implementation of multiplier is done at integer level and bit level. This multiplier is very much efficient for implementing cryptographic techniques like RSA, ECC which requires high level of modular multiplication. The basic flow of this multiplier is to convert and integer into Montgomery domain and then back to integer domain. The basic benefit of doing this the calculating an extra parameter, which is always in the format of 2^n, which in turn reduces the division and multiplication to basic shifting operations [17]. For small numbers, Montgomery multiplier is not so efficient but for applications like RSA, ECC where exponents are big, it reduces the computation time and cost to a large extent when compared to other multiplication schemes.

The following is the basic algorithm of Montgomery multiplier for radix 2:

Algorithm 1 Montgomery Multiplication for Radix 2

Input : *A, B, P*

Output : $Q = A.B.2^{-n} \bmod P$

```
1:  Q = 0
2:  For i=0 to n-1
3:      Q = Q + A_i . B
4:      If Q is odd then Q = Q + P
5:      Q = Q / 2;
6:      If Q > P then Q = Q - P
```

Algorithm 1 gives a basic flow of Montgomery Multiplication. A and B are the two number which are to be multiplied, P is the modulus and n represents the number of bits of P. Output of this algorithm is $Q = A.B.2^{-n} \bmod P$. Here 2^{-n} is the quantity that helps in the conversion of number from integer domain to Montgomery domain. This algorithm can further be understood by taking an example.

Example 1

A = 7, B = 5 and P = 11

So, n = 4

Q = 0

Q = (0 + 5 + 11) / 2 = 8

Q = (8 + 5 + 11) / 2 = 12

Q = (12 + 5 + 11) / 2 = 14

Q = (14 + 0) / 2 = 7, which is the result in Montgomery Domain.

To change this to integer domain, the following equation can be calculated:

$$Q = MM(\bar{Q},1) \bmod P = \bar{Q} \cdot 1 \cdot r^{-1} \bmod P \quad (15)$$

Where $r = 2^n$ and $\bar{Q}$ is the result in Montgomery domain. MM stands for Montgomery Multiplication. Montgomery multiplication can also be implemented through a bit level implementation. The following is the algorithm for bit level implementation of Montgomery Multiplication. It take *a*, *b* and *p* in the form of inputs with *x* as their respective bits and gives the output *q(x).*

Algorithm 2 Montgomery Multiplication at Bit-Level

Input : $a(x),\ b(x),\ p(x)$

Output : $q(x) = a(x)\ b(x)x^{-e}\ mod\ p(x)$

1: $q(x) = 0$

2: $for\ \ i = 0\ \ to\ \ e - 1\ \ do$

3: $q(x) = q(x) + a_i b(x)$

4: $q(x) = q(x) + q_0 n(x)$

5: $q(x) = q(x)/x$

There are pre-computations in this multiplier. To further improve the performance different types of adders can be implemented. There are many types of adders like carry look ahead, carry save adder, etc. This thesis primarily focuses on parallel prefix adders due to their fast computation time.

4.4 Parallel Prefix Adders

Addition is one of the important function in any design. Adders are the basic units of any design. Usually, the maximum speed of operation is determined according to how fast the adders are in that design. One of the major issues in addition is long carry chain which depends on the width of the operand. Parallel Prefix Adders are used in recent implementation in VLSI due to their fast nature. PPA are based on the concept of Carry Look-Ahead adders. PPA are based on Propagate and Generate signals. There are many types of PPA namely, Brent Kung, Kogge Stone, Sklansky, Han Carlson, Knowles and Lander Fischer [26]. Every adder has its own approach but every adder follows a basic flow. These adders are fast due to their pre-computation stage. Figure 4.3 and Figure 4.4 shows the basic flow for PPA.

- Pre-processing stage is the first stage of PPA. In this stage, propagate and generate signals are evaluated.

The equations to compute propagate and generate signal are:

$$p_i = A_i \text{ xor } B_i \quad (16)$$

$$g_i = A_i \text{ and } B_i \quad (17)$$

where A and B are the inputs with *i* representing their bits.

- Carry generation is the second stage on PPA flow. As the name says, carry is generated using propagate and generate signals of the first stage.
- Last stage is the Post-processing stage where the sum bit is calculated using the carry bit and propagate signal.

$$C_i = G_i + P_i \,.\, C_{in} \quad (18)$$

$$S_i = p_i \text{ xor } C_{i-1} \quad (19)$$

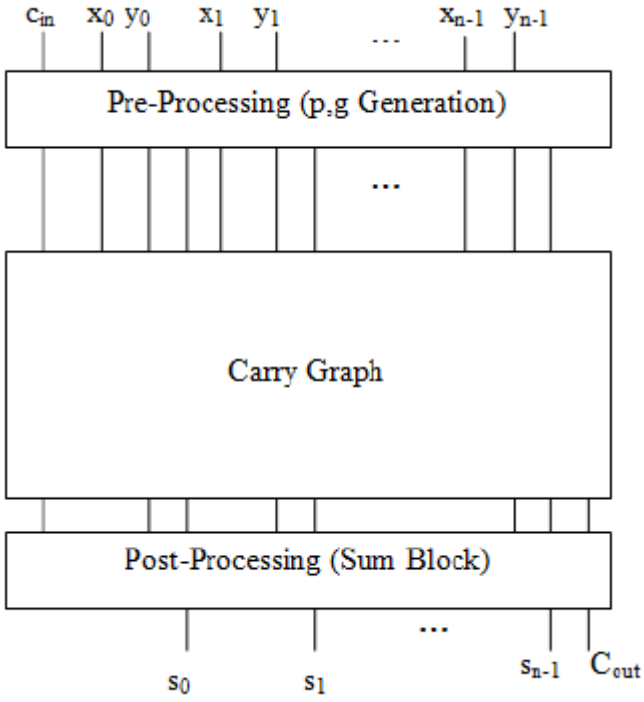

Figure 4.4 Flow Chart of Parallel Prefix Adders [27]

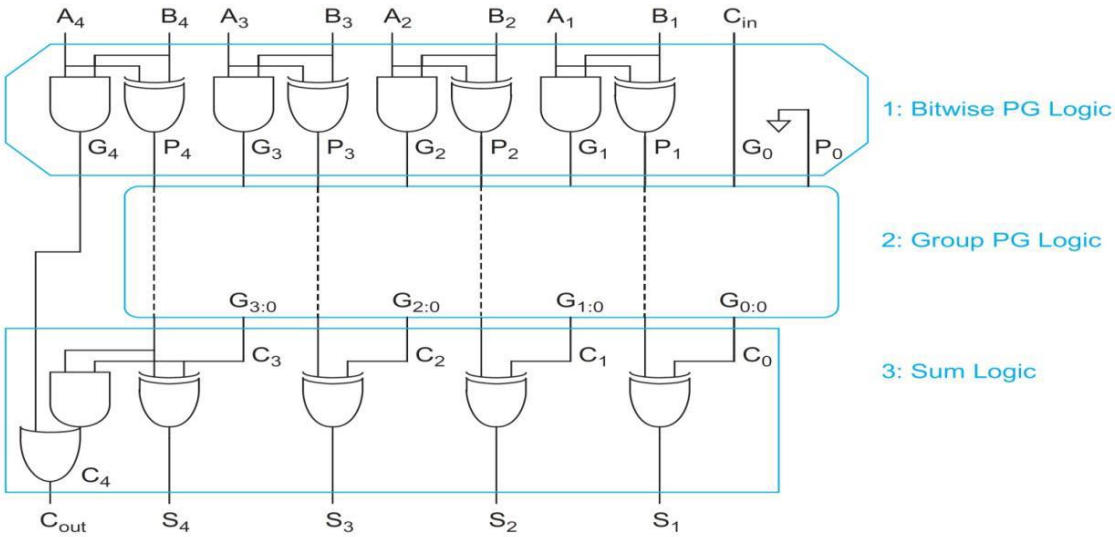

Figure 4.5 Enhanced View of Flow chart of Parallel Prefix Adders [27]

After the survey of [23], it was concluded that Brent Kung and Kogge Stone adders were better than other PPA in terms of computation time and area. Hence, this thesis focuses on implementation of Brent Kung adder and Kogge Stone adder.

4.4.1 Brent Kung

Figure 4.5 shows a 16-bit Brent Kung adder. It is a Parallel Prefix adder whose complexity is O(log2(n)). It takes less area in its implementation. When compared with other PPA, Brent Kung adder has low power, least area but highest delay. When compared to Kogge Stone adder, it has lesser wire congestion.

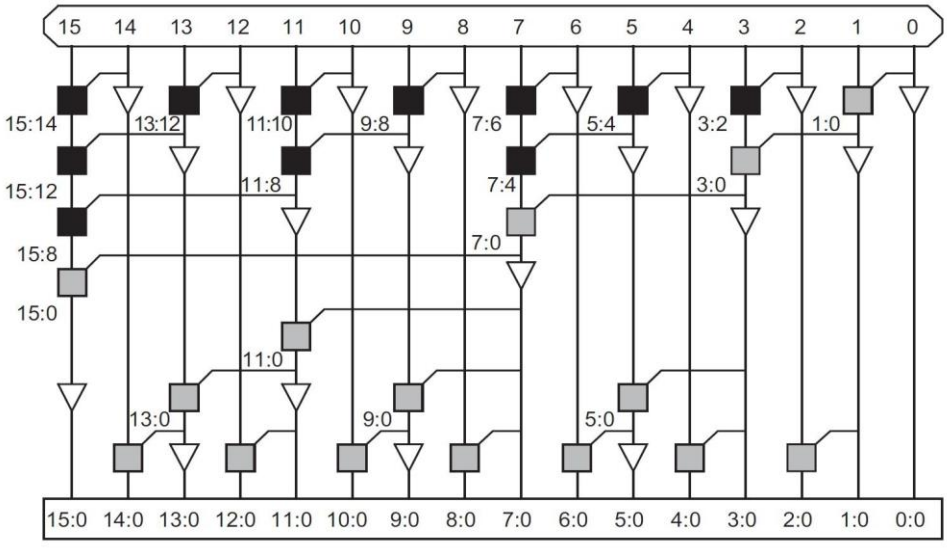

Figure 4.6 16-bit Brent Kung Adder [28]

4.4.2 Kogge Stone

Kogge Stone is a type of Parallel Prefix Adder with complexity of log2N. Wire congestion is very high in it, which in turn leads to higher area and power consumption but still Kogge stone has the least delay among all PPA. This is due the fan-out of 2 at each stage compared to fan out of 4 in other adders. Area is more so it contains more PG cells. Figure 4.5 shows the schematic of a 16-bit Kogge Stone adder.

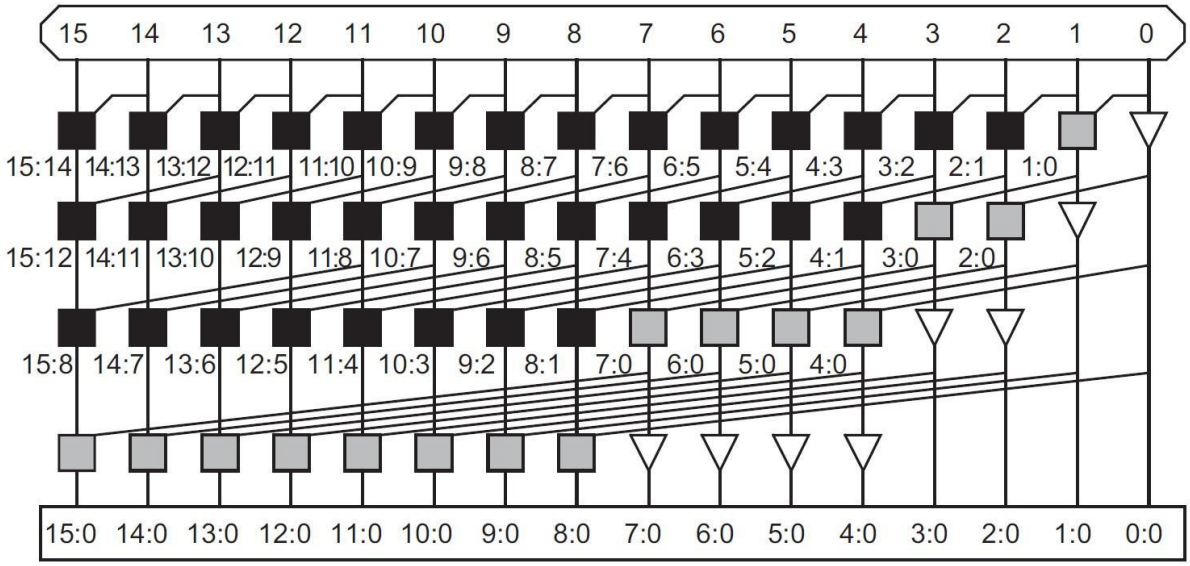

Figure 4.7 16-bit Kogge Stone Adder [28]

Kogge Stone Adder can be understood by the following example:

To calculate addition of two numbers, say A = $(1010)_2$ and B = $(0110)_2$, the schematic shown in figure 4.6 can be drawn.

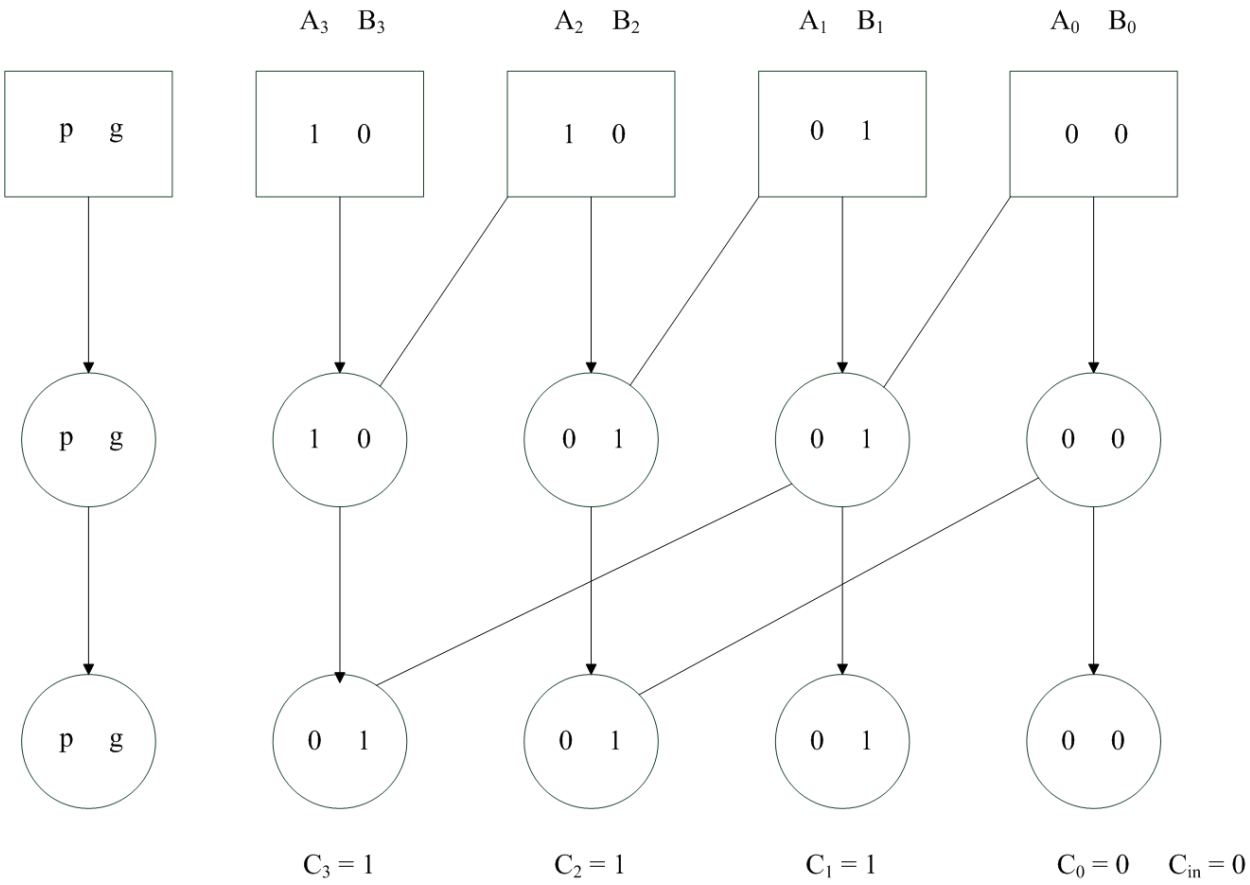

Figure 4.8 4-bit Kogge Stone Adder Example

Propagate and Generate signals are calculated by the formula:

$$P = A_i \oplus B_i \quad (20)$$

$$G = A_i \text{ and } B_i \quad (21)$$

The circles in the above schematic calculate the the value of P and G by the fomula:

$$P = P_i \text{ and } P_{i\ prev} \quad (22)$$

$$G = (P_i \text{ and } G_{i\ prev}) \text{ or } G_i \quad (23)$$

The value of Carry is equivalent to the latest generate signal, i.e. $C_i = G_i$.

The final sum is calculated by the formula, $S = P_i \oplus C_{i-1}$. (24)

For the above example, $S = (01100)_2 \oplus (11100)_2 = (10000)_2 = (16)_{10}$

CHAPTER 5

IMPLEMENTATION RESULTS

First, the software implementation of ECC is done and then hardware implementation of the components (Multiplier and Adder) is done. This module is then used in the implementation of ECC.

5.1 Software Implementation of ECC

Software Implementation of Elliptic Curve Cryptography was done on GCC 4.4.3 compiler with LEON3 processor at 50 MHz. The programming language used was 'C'.

The equation of the Elliptic Curve was taken as:

$y^2 \pmod P = x^3 + Ax + B \pmod P$, where A = 1, B = 1 and P =23

The first step in the process is Point Generation which is basically generating the points that lie on the elliptic curve. The Second step is Selection of Points from the generated points. Third step is Point Operation which is further subdivided into Point Addition and Point Multiplication. The following figures shows the software implementation of ECC:

```
system frequency: 50.000 MHz
serial port A on stdin/stdout
allocated 4096 KiB SRAM memory, in 1 bank
allocated 32 MiB SDRAM memory, in 1 bank
allocated 2048 KiB ROM memory
icache: 1 * 4 KiB, 16 bytes/line (4 KiB total)
dcache: 1 * 4 KiB, 16 bytes/line (4 KiB total)
tsim> load eccnew.exe
section: .text, addr: 0x40000000, size 53872 bytes
section: .data, addr: 0x4000d270, size 2760 bytes
read 372 symbols
tsim> run
starting at 0x40000000
                  0, 1
                  0, 22
                  1, 7
                  1, 16
                  3, 10
                  3, 13
                  4, 0
                  5, 4
                  5, 19
                  6, 4
                  6, 19
                  7, 11
                  7, 12
                  9, 7
                  9, 16
                  11, 3
                  11, 20
                  12, 4
                  12, 19
                  13, 7
                  13, 16
                  17, 3
                  17, 20
                  18, 3
                  18, 20
                  19, 5
                  19, 18
The Point Addition of P(3,10) and Q(9,7) is (17,20)
Program exited normally.
```

Figure 5.1 Point Generation along with Point Addition of two Points

```
system frequency: 50.000 MHz
serial port A on stdin/stdout
allocated 4096 KiB SRAM memory, in 1 bank
allocated 32 MiB SDRAM memory, in 1 bank
allocated 2048 KiB ROM memory
icache: 1 * 4 KiB, 16 bytes/line (4 KiB total)
dcache: 1 * 4 KiB, 16 bytes/line (4 KiB total)
tsim> load eccnew.exe
section: .text, addr: 0x40000000, size 53616 bytes
section: .data, addr: 0x4000d170, size 2760 bytes
read 372 symbols
tsim> run
starting at 0x40000000
                  0, 1
                  0, 22
                  1, 7
                  1, 16
                  3, 10
                  3, 13
                  4, 0
                  5, 4
                  5, 19
                  6, 4
                  6, 19
                  7, 11
                  7, 12
                  9, 7
                  9, 16
                  11, 3
                  11, 20
                  12, 4
                  12, 19
                  13, 7
                  13, 16
                  17, 3
                  17, 20
                  18, 3
                  18, 20
                  19, 5
                  19, 18
The Point Double 2P of P(3,10) is (7,12)
Program exited normally.
```

Figure 5.2 Point Generation along with Point Doubling of a Point

The performance parameters to measure the software implementation were number of clock cycles and Cycles per Instructions (CPI). The following table shows the basic results of the software implementation:

Table 5.1 Performance Analysis of Software Implementation of ECC

Parameter	Value
Clock Cycles	265989
CPI	1.74
Instructions	153847

5.2 Hardware Implementation of ECC

The hardware implementation of ECC was implemented using Verilog HDL. The tool used for Simulation, Synthesis and Implementation is Xilinx Vivado 2016.4. The implementation is done on XC7A35T-ICPG236C (Basys3) board. The elliptic curve used was same for both software and hardware implementations. According to the literature survey, the importance of adders and multipliers in performance improvement of ECC were studied and hence the hardware

implementation is subdivided into four modules. First is the implementation of a parallel prefix adder. The adder taken was Kogge Stone adder as it had less delay [28]. Second is the multiplier which is Montgomery Modular Multiplier. The main advantage of this multiplier is that the multiplication is carried out simultaneously with the modular arithmetic.

The implementation of 32-bit Kogge Stone adder is shown in Section 5.2.1. Section 2.2.2 shows the implementation of Montgomery Modular Multiplier. Next section shows the results when Kogge Stone adder is embedded into the multiplier to improve its performance. Section 5.2.4 shows the comparative analysis of ECC and implementation of ECC with Montgomery multiplier and Kogge-Stone adder. All implementation are compared by parameters namely Slices, Maximum Frequency and Throughput.

- *Slices*: Slices contain Look up Tables (LUTs) and Flip Flops. The number of LUTs and FFs are different in different families of FPGA. Slices are helpful as they are parameter for area usage in FPGA.
- *Maximum frequency*: The frequency over which circuit or design will not give correct/desired output. Formula used to calculate Maximum frequency is:

$$\text{Maximum Frequency} = \frac{1}{Required\ time - Worst\ Negative\ Slack} \tag{25}$$

- *Throughput*: It is defined as number of bits transferred in a given amount of time. Formula used to calculate throughput is:

$$\text{Throughput} = \frac{Block\ Size\ X\ Maximum\ frequency}{Latency} \tag{26}$$

5.2.1 Kogge-Stone Adder

A 32-bit Kogge stone adder is designed using Verilog HDL. The basic flow for the adder is explained in Section 4.4.2. After Simulating in Xilinx Vivado 2016.4, the following waveform is obtained.

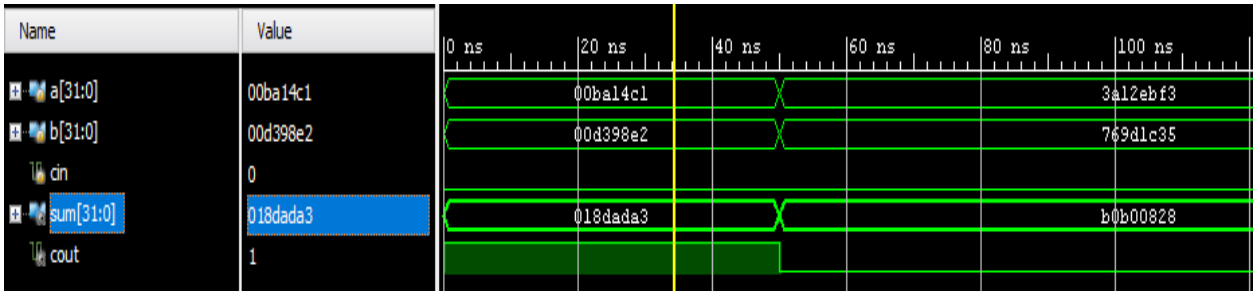

Figure 5.3 Waveform of 32-bit Kogge Stone Adder

In the above waveform, a and b are two 32 bit inputs. Sum is 32-bit output along with single bit cout as carry out. The following table shows the results of the adder.

Table 5.2 Example of Input and Output of a 32-bit Kogge Stone Adder

Input a[31:0]	Input b[31:0]	Output Sum [31:0]	Output Cout
0xBA14C1	0xD398E2	0x8DADA3	1
0x3A12EBF3	0x769D1C35	0xB0B00828	0

Table 5.3 Implementation Results of 32-bit Kogge-Stone Adder

Parameter	Value	Total	Utilization (%)
I/O Pins	5	106	4.71
Slices	251	8150	3.08
Maximum Frequency	189.32 MHz	-	-
Throughput	6058.24 Mbps	-	-

5.2.2 Montgomery Modular Multiplier

This multiplier was designed using Verilog HDL. The algorithm used is described in Section 4.3. After simulating in Xilinx Vivado 2016.4, the following waveform is obtained.

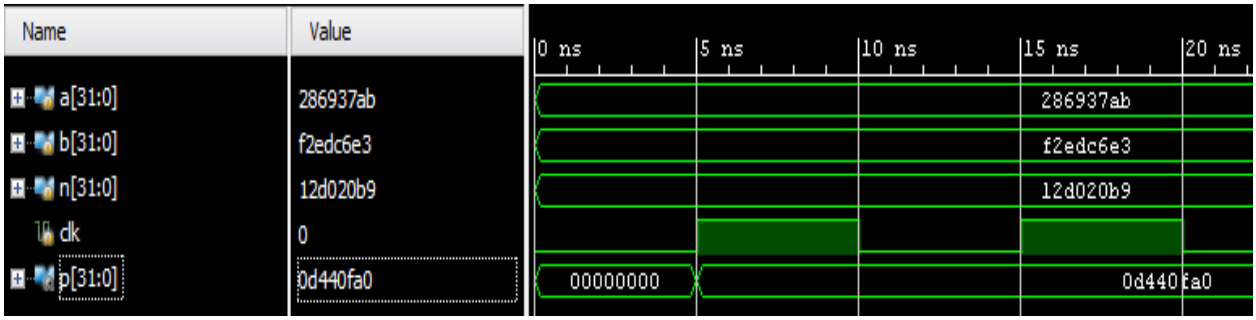

Figure 5.4 Waveform for 32-bit Montgomery Modular Multiplier

In the above waveform, a and b are the two 32-bit inputs, n is 32-bit the modulus and p is the 32-bit out in Montgomery domain.

Table 5.4 Example of Input and Output of a 32-bit Montgomery Modular Multiplier

Input a[31:0]	Input b[31:0]	Input n [31:0]	Output p [31:0]
0x286937ab	0xf2edc6e3	0x12d020b9	0x0d440fa0

Table 5.5 Implementation Results of 32-bit Montgomery Modular Multiplier

Parameter	Value	Total	Utilization (%)
I/O Pins	4	106	3.77
Slices	456	8150	5.60
Maximum Frequency	162.81 MHz	-	-
Throughput	5209.92 Mbps	-	-

5.2.3 Montgomery Multiplier with Kogge-Stone adder

The Montgomery multiplier was implemented with Kogge-Stone adder in order to improve the performance. The results were check by giving the same input vectors as given in simple implementation.

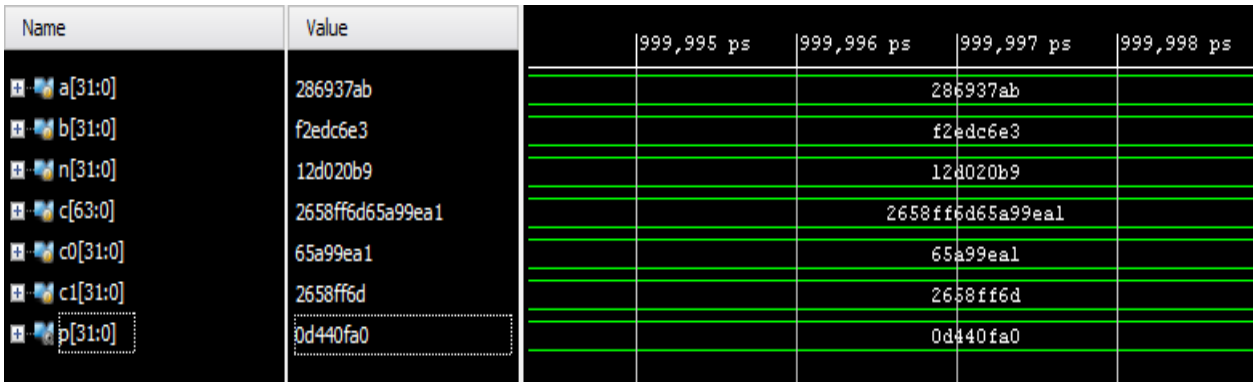

Figure 5.5 Waveform for 32-bit Montgomery Multiplier with Kogge-Stone Adder

In the above waveform, a and b are two 32-bits inputs along with a 32-bit modulus n. The output is 32-bit p. The internal signal are c, c0 and c1 which are basically pre-computed.

Table 5.6 Example of a 32-bit Montgomery Modular Multiplier with Kogge-Stone Adder

Input a[31:0]	Input b[31:0]	Input n [31:0]	Output p [31:0]
0x286937ab	0xf2edc6e3	0x12d020b9	0x0d440fa0

Table 5.7 Implementation Results of 32-bit Montgomery Modular Multiplier with Kogge-Stone adder

Parameter	Value	Total	Utilization (%)
I/O Pins	4	106	3.77
Slices	359	8150	4.40
Maximum Frequency	185.32 MHz	-	-
Throughput	5930.24 Mbps	-	-

5.2.4 Elliptic Curve Cryptography

The Elliptic Curve Cryptography was implemented using Verilog HDL. First the basic implementation of ECC was done. After doing the literature survey, the adder and multiplier were selected according to need for improvement in performance. This section gives a comparative analysis of the basic implementation on ECC and the implementation with Montgomery multiplier and Kogge-Stone adder.

The following are the waveforms of Point Addition and Point Doubling with selected point as P (3, 10) and Q (9, 7).

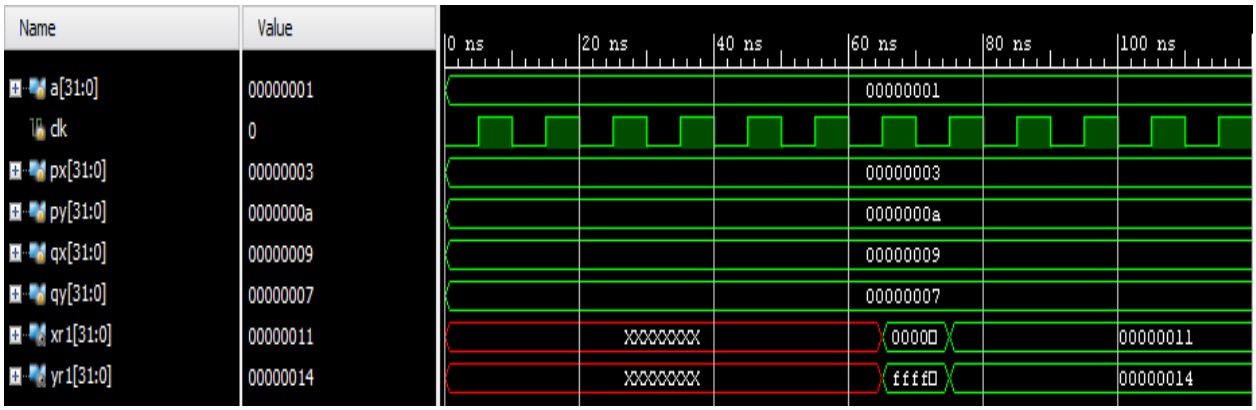

Figure 5.6 Waveform of Point Addition in ECC

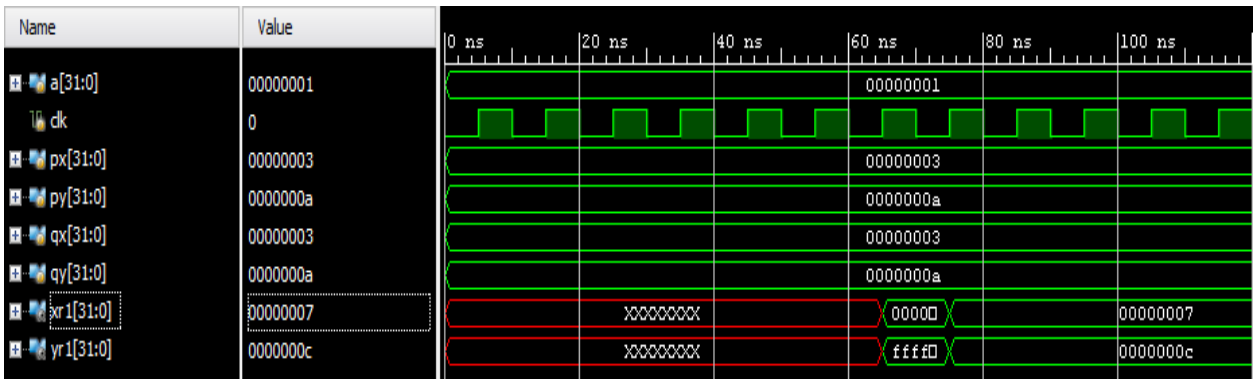

Figure 5.7 Waveform of Point Doubling in ECC

In the above waveforms, px and py are the X and Y co-ordinates of point P and qx and qy are the X and Y co-ordinates of point Q. The points P and Q are generated in the Point Generation method. The equation of the elliptic curve is:

$y^2 \pmod p = x^3 + Ax + B \pmod p$, where A = 1, B = 1 and p = 23.

Point Addition of P (3, 10) and Q (9, 7) resulted R (17, 20) and Point Doubling of P (3, 10) and Q (9, 7) resulted R (7, 12).

Table 5.8 Comparative of simple ECC (Design 1) and ECC with Multiplier and adder (Design 2)

Parameter	**Value**		**Total**	**Utilization (%)**	
	Design 1	**Design 2**		**Design 1**	**Design 2**
Slices	1883	2039	8150	23.10	25.01
Maximum Frequency MHz	148.65	172.33	-	-	
Throughput	679.542 Mbps	787.79 Mbps	-	-	

CHAPTER 6

CONCLUSION AND FUTURE SCOPE

6.1 Conclusion

In this work, initial study on IoT and its application is done. Since, the devices interconnected to each other are increased in modern designs, the security and authenticity of the information is a must. To provide these, asymmetric cryptographic algorithms such as Diffie-Hellman, El-Gamal and RSA are explained. Due to the large key size and high computation time, these algorithms are not suitable for resource constrained devices. Hence, ECC algorithm comes in picture which provides same level of security with small key size. The mathematics of elliptic curves in finite field for ECC algorithm was studied. Based on the literature survey, it was concluded that with the use of efficient multipliers, the performance of ECC algorithm can be improved. Now further, to enhance the performance of multiplier, an efficient adder can be used. Among all, Montgomery multiplier and Kogge-Stone adder were selected and implemented with ECC. Comparing basic ECC with the ECC implementation with Montgomery multiplier and Kogge-Stone adder, a performance improvement of 15.9 % is observed.

6.2 Future Scope

In this section, some future directions are defined in which further work can be done.

- In the asymmetric algorithms, adder chain is plays an important role, which further can be reduced or improved using optimization algorithm.
- Hardware implementation of ECC can be done at ASIC level.
- In Montgomery multiplier, bit forwarding algorithms can be implemented upon [15].
- At software level, many optimization techniques can be explored to improve the performance with respect to number of clock cycles.

REFERENCES

[1] Hossain, M. M., Fotouhi, M., & Hasan, R. (2015). *Towards an analysis of security issues, challenges, and open problems in the internet of things.* Paper presented at the Services (SERVICES), IEEE World Congress, pp. 21-28.

[2] Ferguson, N., Schneier, B., & Kohno, T. (2015). Introduction to cryptography. *Cryptography Engineering: Design Principles and Practical Applications*, pp. 23-39.

[3] Chandra, S., Bhattacharyya, S., Paira, S., & Alam, S. S. (2014). *A study and analysis on symmetric cryptography.* Paper presented at the Science Engineering and Management Research (ICSEMR), 2014 International Conference, pp. 1-8.

[4] Kovalenko, I. N., & Kochubinskii, A. I. (2003). Asymmetric cryptographic algorithms. Cybernetics and systems analysis, 39(4), pp. 549-554.

[5] Al-Aali, G., Boneau, B., & Landers, K. (2000). Diffie-hellman key exchange. *Proceedings of CSE 331, Data Structures Fall 2000, 67*.

[6] ElGamal, T. (1985). A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE transactions on information theory, 31*(4), pp. 469-472.

[7] Khatarkar, S., & Kamble, R. (2015). A Survey and Performance Analysis of Various RSA based Encryption Techniques. *International Journal of Computer Applications, 114*(7).

[8] Javeed, K., Wang, X., & Scott, M. (2017). High performance hardware support for elliptic curve cryptography over general prime field. *Microprocessors and Microsystems, 51*, pp. 331-342.

[9] Azarderakhsh, R., & Reyhani-Masoleh, A. (2015). Parallel and High-Speed Computations of Elliptic Curve Cryptography Using Hybrid-Double Multipliers. *IEEE Trans. Parallel Distrib. Syst., 26*(6), pp. 1668-1677.

[10] Hankerson, D., Hernandez, J. L., & Menezes, A. (2000). *Software implementation of elliptic curve cryptography over binary fields.* Paper presented at the International Workshop on Cryptographic Hardware and Embedded Systems, pp. 1-24.

[11] Gura, N., Patel, A., Wander, A., Eberle, H., & Shantz, S. C. (2004). *Comparing elliptic curve cryptography and RSA on 8-bit CPUs.* Paper presented at the International workshop on cryptographic hardware and embedded systems, pp. 119-132.

[12] Srinivasan, M., & Tamilselvan, G. (2017). VLSI Implementation of Low Power High Speed ECC Processor Using Versatile Bit Serial Multiplier. *Journal of Circuits, Systems and Computers, 26*(07), 1750114.

[13] Mahto, D., & Yadav, D. K. (2017). RSA and ECC: A Comparative Analysis. *International Journal of Applied Engineering Research, 12*(19), pp. 9053-9061.

[14] Rafferty, C., O'Neill, M., & Hanley, N. (2017). Evaluation of large integer multiplication methods on hardware. *IEEE Transactions on Computers, 66*(8), pp. 1369-1382.

[15] Kim, K.-W., Lee, H.-H., & Kim, S.-H. (2018). *Efficient Combined Algorithm for Multiplication and Squaring for Fast Exponentiation over Finite Fields $$ GF (2^{m}) $$.* Paper presented at the Proceedings of the 7th International Conference on Emerging Databases, pp. 50-57.

[16] Vollala, S., & Ramasubramanian, N. (2017). Energy efficient modular exponentiation for public-key cryptography based on bit forwarding techniques. *Information Processing Letters, 119*, pp. 25-38.

[17] Agrawal, K., Shah, M., & Asari, G. (2018). A Review Paper on Multiplier Algorithms for VLSI Technology.

[18] Javeed, K., & Wang, X. (2014). *Efficient montgomery multiplier for pairing and elliptic curve based cryptography.* Paper presented at the Communication Systems, Networks & Digital Signal Processing (CSNDSP), 2014 9th International Symposium, pp. 255-260.

[19] Thomas, A., & Manuel, E. M. (2016). Embedment of montgomery algorithm on elliptic curve cryptography over RSA public key cryptography. *Procedia Technology, 24*, pp. 911-917.

[20] Asif, S., & Kong, Y. (2017). Highly parallel modular multiplier for elliptic curve cryptography in residue number system. *Circuits, Systems, and Signal Processing, 36*(3), pp. 1027-1051.

[21] Dyka, Z., & Langendoerfer, P. (2005). *Area efficient hardware implementation of elliptic curve cryptography by iteratively applying Karatsuba's method.* Paper presented at the Design, Automation and Test in Europe, 2005. Proceedings, pp. 70-75.

[22] Chelton, W. N., & Benaissa, M. (2008). Fast elliptic curve cryptography on FPGA. *IEEE Trans. Very Large Scale Integr. Syst., 16*(2), pp. 198-205.

[23] Daphni, S., & Grace, K. V. (2017). *A review analysis of parallel prefix adders for better performnce in VLSI applications.* Paper presented at the Circuits and Systems (ICCS), 2017 IEEE International Conference, pp 103-106.

[24] Hobson, R. F. (2015). A framework for high-speed parallel-prefix adder performance evaluation and comparison. *International Journal of Circuit Theory and Applications, 43*(10), pp. 1474-1490.

[25] Manasa, M., Swapna, E., & KITS, S. (2017). VLSI Design of a High-Speed and Energy-Efficient Carry Skip Adder. *International Journal of Engineering Science, 13430*.

[26] Koblitz, N., Menezes, A., & Vanstone, S. (2000). The state of elliptic curve cryptography. *Designs, codes and cryptography, 19*(2-3), pp. 173-193.

[27] Seroussi, G. (1999). *Elliptic curve cryptography.* Paper presented at the Information Theory and Networking Workshop, 1999, p. 41.

[28] Vitoroulis, K., & Al-Khalili, A. J. (2007). *Performance of parallel prefix adders implemented with FPGA technology.* Paper presented at the Circuits and Systems, 2007. NEWCAS 2007. IEEE Northeast Workshop, pp. 498-501.

[29] Zamhari, N., Voon, P., Kipli, K., Chin, K. L., & Husin, M. H. (2012). *Comparison of parallel prefix adder (PPA).* Paper presented at the Proceedings of the World Congress on Engineering, pp. 4-6.

[30] Hoffstein J., Pipher J., Silverman Joseph H., *An Introduction to Mathematical Cryptography*, USA: Springer Science + Business Media, LLC, 2000.

601662002

by Aditya Shankar

Submission date: 12-Jul-2018 10:12PM (UTC+0530)
Submission ID: 982099844
File name: Thesis_Report.docx (9.09M)
Word count: 9546
Character count: 47416

601662002

ORIGINALITY REPORT

18%	13%	12%	10%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	dspace.fsktm.um.edu.my Internet Source	2%
2	www.dkrypt.com Internet Source	1%
3	Bahadori, Milad, Mehdi Kamal, Ali Afzali-Kusha, and Massoud Pedram. "High-Speed and Energy-Efficient Carry Skip Adder Operating Under a Wide Range of Supply Voltage Levels", IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 2015. Publication	1%
4	www.ripublication.com Internet Source	1%
5	ijsetr.org Internet Source	<1%
6	Submitted to Graz University of Technology Student Paper	<1%
7	Submitted to VIT University Student Paper	<1%

8	article.sapub.org Internet Source	<1%
9	Submitted to Visvesvaraya Technological University Student Paper	<1%
10	Submitted to Arab Open University Student Paper	<1%
11	ieeexplore.ieee.org Internet Source	<1%
12	Submitted to Sheffield Hallam University Student Paper	<1%
13	Jeffrey Hoffstein. "Discrete Logarithms and Diffie Hellman", Undergraduate Texts in Mathematics, 2008 Publication	<1%
14	www.ijser.org Internet Source	<1%
15	security.ece.orst.edu Internet Source	<1%
16	Submitted to University of York Student Paper	<1%
17	Heiser, Jay, Steve Stanek, Sasan Hamidi, Ben Rothke, Paul Lambert, Ralph Spencer Poore, James Tiller, Ronald Gove, and Mark Edmead. "Methods of Attacking and Defending	<1%