2008

Hardware implementation of elliptic curve Diffie-Hellman key agreement scheme in GF(p)

Zerene Sangma

Hardware Implementation of Elliptic Curve Diffie-Hellman Key Agreement Scheme in GF(p)

by

Zerene Sangma

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of
Master of Science in Computer Engineering

Supervised by

Dr Kenneth Hsu
Department of Computer Engineering
Kate Gleason College of Engineering
Rochester Institute of Technology
Rochester, New York
October 2008

Approved By:

Dr Kenneth Hsu
Primary Adviser

Secondary Adviser
Dr Dhireesha Kudithipudi

Secondary Adviser
Dr Marcin Lukowiak

Thesis Release Permission Form

Rochester Institute of Technology

Kate Gleason College of Engineering

Title: Hardware Implementation of Elliptic Curve Diffie-Hellman Key Agreement Scheme in GF(p)

I, Zerene Sangma, hereby grant permission to the Wallace Memorial Library reporduce my thesis in whole or part.

Zerene Sangma

Date

Dedication

I wish to dedicate my project work to my parents and sister for their love and support, for always believing in me and encouraging me.

Acknowledgments

At the outset, I would like to acknowledge all those who have made this work a success.

I thank God for His faithfulness and unconditional love. With great pleasure, I would like to extend my sincere thanks and gratitude to my primary advisor Dr. Hsu for his patience, continued support and guidance throughout the thesis from the beginning till the end.

I extend my heartfelt thanks and gratitude to my committee members Dr Dhireesha Kudithipudi and Dr Marcin Lukowiak who gave their valuable time and guidance in making this thesis a success.

To my colleague and friend Jonas Sibomana, for his time and effort in helping me with the design and debugging process. I am grateful to him for treating this thesis as his own and his encouragement and support when I felt like I was going nowhere.

I would also like to thank Dr. A. Savakis for being my mentor and guide. I thank Pamela Steinkirchner and Kathryn Stefanik and all the staff at CE department for their continued support and efficient management. They have indeed made every student's life easier.

I would also like to express my heartfelt thanks to my friends Jinhee and Angeline for their prayer and support. Last but not the least, my family for giving their all to support my dreams and ambitions.

Abstract

With the advent of technology there are many applications that require secure communication. Elliptic Curve Public-key Cryptosystems are increasingly becoming popular due to their small key size and efficient algorithm. Elliptic curves are widely used in various key exchange techniques including Diffie-Hellman Key Agreement scheme.

Modular multiplication and modular division are one of the basic operations in elliptic curve cryptography. Much effort has been made in developing efficient modular multiplication designs, however few works has been proposed for the modular division. Nevertheless, these operations are needed in various cryptographic systems. This thesis examines various scalable implementations of elliptic curve scalar multiplication employing multiplicative inverse or field division in GF(p) focussing mainly on modular divison architectures.

Next, this thesis presents a new architecture for modular division based on the variant of Extended Binary GCD algorithm. The main contribution at system level architecture to the modular division unit is use of counters in place of shift registers that are basis of the algorithm and modifying the algorithm to introduce a modular correction unit for the output logic. This results in 62% increase in speed with respect to a prototype design.

Finally, using the modular division architecture an Elliptic Curve ALU in GF(p) was implemented which can be used as the core arithmetic unit of an elliptic curve processor. The resulting architecture was targeted to Xilinx Vertex2v6000-bf957 FPGA device and can be implemented for different elliptic curves for almost all practical values of field p. The frequency of the ALU is 58.8 MHz for 128-bits utilizing 20% of the device at 27712 gates which is 30% faster than a prototype implementation with a 2% increase in area utilization. The ALU was tested to perform Diffie-Hellman Key Agreement Scheme and is

suitable for other public-key cryptographic algorithms.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Security is very important in today's communication driven era. There is an increasing demand for secured data just as there is an increasing demand for information interchange and electronic services. Encryption is a process of converting a message by means of an algorithm into some different characters such that it cannot be understood by any one else other than the intended user who has the means to convert the characters into the original message. The algorithms used in encryption are called cryptographic algorithm and systems that implement such algorithm are called cryptosystems.

There are two kinds of cryptosystems that implement cryptographic algorithms to encrypt and decrypt data. These are public-key and private-key cryptosystems. The basic difference between the two systems is that in public-key cryptosystem also known as asymmetric cryptosystem a public-key is used to encrypt the data and a private key is used to decrypt the data. The public-key, as the name implies is widely distributed while the private key is kept secret. In private-key cryptosystem, also known as symmetric cryptosystem, one secret key is used to encrypt and decrypt the data. In such a system, both the users involved in encrypting and decrypting the data must know the secret key beforehand.

Due to the nature of the algorithms, public-key cryptosystems are much slower than private-key cryptosystems. So in practice, vast amount of data is encrypted using private-key cryptosystems. Therefore, one needs to consider means for the exchange and establishment of keys between two parties for symmetric encryption algorithm. There are two types of key establishment schemes: Key Agreement scheme and Key Distribution scheme. In

key distribution scheme, one party decides on a key and sends the information to the other party. In key agreement scheme, both the parties interact with each other, exchange information and generate a key based on the information. Key agreement schemes are best done using public-key cryptosystem. Once the key is established the users can use more secure and faster ways for encryption of bulk data.

1.1 Motivation

In 1976, Whitfield Diffie and Martin Hellman first published the concept of public-key cryptography[1]. Since then popularity of public-key cryptography has grown rapidly. In 1977, Rivest, Shamir and Adleman invented the well known RSA public-key cryptosystem[2]. Cryptography focuses on finite fields. It was found that finite fields of the form $\mathbb{Z}_P$ where P is a prime number and of the order 10^{40}[39]can be used to implement public-key cryptographic algorithms. The Diffie-Hellman Key Agreement Scheme was the first and the best known example of public-key cryptography[3].

In 1985, Neal Koblitz[5] suggested the use of elliptic curves in public-key cryptography. It was proposed that a group of points on an elliptic curve over a finite field can be used for encrypting data and can provide more security than the fields of the form $\mathbb{Z}_P$. Since then elliptic curves have played a significant role in public-key cryptography. Elliptic curve public-key cryptosystems can provide the same level of security as RSA cryptosystem with much smaller key size. For example, a 160-bit elliptic curve cryptosystem is as secure as 1024-bit RSA cryptosystem[2]. The use of smaller keys and computationally more efficient algorithm than traditional cryptographic algorithms are the main reason behind the increasing popularity of elliptic curve cryptography (ECC). There are two types of finite fields that are popular for elliptic curve cryptography - $GF(p)$ and $GF(2^m)$. Hardware implementation of elliptic curve cryptography over $GF(2^m)$ are more popular than $GF(p)$ due to their carry free addition[6]. However, in case of Field Programmable Gate Arrays, carry chain adders are optimized so arithmetic over $GF(p)$ are less complex and more

suitable for FPGA implementation[21].

Arithmetic over elliptic curves requires modular division, modular multiplication, and modular addition/subtraction operations. The modular division is the most critical operation as it is computationally extensive and expensive. Most of the implementation in this case is done by representing the points on the curve using projective coordinates that eliminates inversion/division. However, a final division is still required to convert the projective coordinates into affine coordinates. In other cases, modular division is replaced by modular inversion followed by modular multiplication. Montgomery Multiplication algorithm is one of the most efficient algorithm for modular multiplication[7]. On the other hand, the Extended Binary GCD Algorithm is the most efficient method to perform modular division[8]. Till date, there have been very few hardware implementations of elliptic curve cryptosystem employing a modular division unit[9], however, hardware implementations of modular division architecture suitable for ECC have been documented in [10][11][12][13].

Elliptic curve cryptography finds its wide use in various public-key cryptography algorithms in particular those involving discrete logarithms. To name a few:

- Elliptic Curve ElGamal Cryptosystem
- Elliptic Curve Diffie-Hellman Key Exchange
- ElGamal Digital Signatures

1.2 Previous Work

There has been extensive research in the field of elliptic curve cryptography. Elliptic curves over $GF(2^m)$ have gained much popularity because arithmetic in $GF(2^m)$ provides carry free addition[14][15][15][16][17]. However, there have been few $GF(p)$ arithmetic processors reported till date as well. Most of the architectures over $GF(p)$ are suitable for field programmable gate array technology and use projective coordinates to perform elliptic curve point multiplication and focus on modular multiplication neglecting modular

inversion/division. This section briefly highlights the work that has been done related to this thesis.

The first documented elliptic curve processor over $GF(p)$ was presented by Orlando and Paar in [18, 19]. The elliptic curve processor is a scalable architecture in terms of area and speed suitable for field programmable gate arrays(FPGAs). The processor uses a new type of high-radix Montgomery Multiplier that relies on precomputation of frequently used values and on the use of multiple processing engines. Projective coordinates were used to represent the points on curve. It was estimated that 192-bit point multiplication would take 3 ms based on the assumption that that the multiplier would have 100% throughput . The expected latency was not considered. The clock frequency was observed at 40 MHz.

In 2003, Ors et all[20] presented a hardware implementation of Elliptic Curve (EC) processor over $GF(p)$. Montgomery multiplier in a systolic array architecture was used for modular multiplication. It was stated by the authors that the resulting architecture used less memory than the one proposed by Orlando and Paar and had minimum clock frequency of 10.952 ns, about 91.308 MHz.

Another implementation of ECC ALU was presented by Alan Daly et all[21]. The ALU is capable of performing all modular operations for elliptic curve cryptography(ECC) including a modular division. The modular division was performed by a modular inverse followed by Montgomery modular multiplication. The architecture presented can be implemented for ECC using affine and projective coordinates. The resulting ALU was able to operate at little over 50 MHz.

Alan Daly et all in[9] also introduced an elliptic curve processor over $GF(p)$ on reconfigurable logic based on a modular division algorithm by Shantz[22]. The resulting architecture was targeted to Xilinx Virtex XCV200e-6bg560 and achieved a maximum frequency of 45 MHz for 64-bit architecture. This architecture is the only one reported in literature till date that employs a modular divider to perform elliptic curve scalar multiplication.

There are several other hardware implementations of ECC over $GF(p)$ reported in

literature[23][24][25][26][27]. Most of these architectures are based on Montgomery Multiplier and support one specific elliptic curve. For example, in [27] a novel GF(p) elliptic curve cryptography processor was proposed. The proposed architecture used projective coordinates hence, modular inversion was avoided.

The motivation behind the proposed thesis is to gain understanding between the various area and speed trade-off that can be implemented to present a powerful ALU capable of performing elliptic curve arithmetic. Since there are various techniques to perform elliptic curve scalar multiplication which is the main operation in elliptic curve arithmetic, this thesis aims to study implementation of modular division module for operations on elliptic curve over GF(p).

1.3 Objectives

The objective of the thesis is to implement an Elliptic Curve Cryptography (ECC) ALU in finite field suitable for Diffie-Hellman KAS. Affine coordinates will be used to represent the points on the elliptic curve. The underlying operation of scalar multiplication on elliptic curve requires a modular division. In most of the architectures till date, the modular division is replaced by modular inverse followed by modular multiplication. This thesis aims to implement an ECC ALU based on the variant of Extended Binary GCD modular divider as proposed in[11]. The goal of the thesis is to introduce a new divider architecture and implement it using FPGA. In[11], the divider architecture was implemented using radix-2 signed digit representation, in this thesis we proposed to use standard binary representation and thus avoid the special adders that will be required to perform radix-2 addition thereby reducing the area. Using a new hardware design approach by reducing the number of components and using optimized adders that are in-built in FPGAs, further optimizations can be achieved. The proposed modular divider architecture will be implemented in an ECC ALU over GF(p). The goal is to perform a comparative study with respect to some of the implementations in GF(p) that are using different approaches for scalar multiplication.

Using a modular divider may lead to reduced area in comparison to the designs that implement modular inverse and modular multiplier, however, the speed of former designs may be lower than the latter. With careful hardware design a balance may be achieved in terms of area and speed.

To the best of our knowledge there is no implementation of ECC ALU utilizing modular division based on the said algorithm except in this work.

Field Programmable Gate Arrays (FPGAs) are arrays of logic units that can be programmed by the user to operate as special purpose functional units. They can evaluate certain types of tasks at far higher speeds than those achievable on general-purpose processors. Tasks with limited data dependencies and tasks with significant scope for parallel execution are those on which the most significant performance advantage can be extracted. Considering this easy prototyping feature and speed grade provided by the FPGA, the implementation was targeted on a vertex II pro FPGA.

1.4 Thesis Outline

The organization of thesis is as follows:

Chapter 2 introduces the basic arithmetic background needed to understand the concept followed in this thesis. It starts with the finite field arithmetic background and introduction to $GF(p)$. A brief introduction to Modular Inverses, Extended Euclidean algorithm, and Binary GCD algorithm are also included in this chapter. The chapter then proceeds with an introduction to elliptic curves and elliptic curves defined over a finite field. It briefly dwells on the elliptic curve discrete logarithm problem that explains why elliptic curves are becoming increasingly popular in the field of public-key cryptography. The chapter ends with introduction to the elliptic curve point multiplication operation and the coordinate representation used in this thesis.

The goal of Chapter 3 is to introduce the Key Agreement Scheme and the algorithm involved in the scheme. This chapter is important as it introduces the first and the well

known public-key cryptographic algorithm which is the Diffie-Hellman Key Agreement Scheme(DHKAS). The architecture presented in this thesis is suitable for elliptic curve Diffie-Hellman KAS. The algorithm for elliptic curve DHKAS is presented at the end of chapter 3.

Chapter 4 highlights the supporting work for this thesis.

Chapter 5 introduces the design methodology used to design the architecture presented in this thesis. It introduces the ECC $GF(p)$ architecture. It describes the functionality of the main units namely the controller and the arithmetic unit. The chapter presents the algorithm implemented in this thesis. It ends with a brief explanation of the double-add algorithm used in the implementation of elliptic curve scalar multiplication.

Chapter 6 explains the hardware implementation of the various logic units of the architecture starting from the explanation of the finite state machine to the output generated. It describes the logic units used for performing various finite field arithmetic, describes the new architecture for the modular divider, and provides the functional simulation results of each logic unit in the architecture.

Chapter 7 is dedicated to providing the simulation results of the entire architecture, provides a comparative study with respect to some of the prototype implementations of an elliptic curve processor for point multiplication for curves defined over a $GF(p)$ finite field.

Chapter 8 summarizes the conclusion of this work and also provides recommendation for future work.

Appendix A lists all the simulation results using modelsim simulator.

Appendix B includes the synthesis reports. Post simulation reports are presented in Appendix C and Appendix D lists all the VHDL files.

Chapter 2

Background

In Cryptographic systems, messages and keys are represented as numerical values and in order to encrypt the message various mathematical operations are applied so that the resulting encoded message is difficult to decode except by the intended recipient. In order to build, analyze or study strong cryptosystems, it is important to know the mathematical tools and concepts related to the field. This introductory chapter presents the various mathematical concepts related to finite field, elliptic curves, discrete logarithm algorithm and diffie-hellman problem that are important to the research work.

2.1 Finite Field Arithmetic

Finite Fields were first introduced by Evariste Galois in 1830 in his proof of the unsolvability of the general quintic equation. Finite Fields have found their applications in computer science, information theory and are used in many public-key cryptosystems including Elliptic Curve Cryptography(ECC). This section highlights some of the properties of finite fields that are fundamental for efficient hardware implementations of finite field arithmetic with respect to cryptography.

2.1.1 Finite Field Background

The following are few fundamental definitions associated with finite fields. Additional information on this subject can be found in[29].

Abelian group:

An *Abelian or commutative group* consists of a set G together with a binary operation(*) satisfying the following axioms:

- *Closure:* a group is closed if $g, h, f \in G$ such that $g * h = f$;
- *Associativity:* for all $f, g, h \in G$, $f * (g * h) = (f * g) * h$ defines the associative law*;*
- *Identity:* there exists an identity element $e \in G$ such that for all $g \in G$, $g * e = g = e * g$ holds;
- *Inverses:* for each element $g \in G$ there is an element $m \in G$ such that $g * m = e = m * g$, where e is an identity element;
- *Commutativity:* for all $g, h \in G$, the commutative law$gh = hg$ holds.

A *Ring* is a set ***R*** with two binary operations addition[+] and multiplication [*], with distinct elements satisfying the following:

- *Additive associativity:* R is an Abelian group with respect to addition. For all $f, g, h \in R$, $f + (g + h) = (f + g) + h$;
- *Commutativity:* for all $r, s \in R$, $r * s = s * r$;
- *Distributivity:* for all $r, s, t \in R$, $r(s + t) = rs + st$;
- *Identity:* There exists an element 1 in R such that for $r \in R$, $r * 1 = r$;

A field is a ring such that the elements F form an Abelian group under the operation addition[+] with 0 as the identity element. The rest of the elements of F other than the ones that

form the additive associativity form an Abelian group under the operation multiplication[*] with 1 as the identity element. The distributive law holds for the two binary operations such that for all $a, b, c \in F$, $a(b + c) = (a * b) + (a * c)$. If the number of elements are finite, the field is called a Finite Field. The field with p^n elements is called GF(p^n) for "Galois Field" in honor of Evariste Galois, a French mathematician who did early work on Fields.

For every power p^n of a prime, there exists exactly one finite field with p^n elements, and these are the only finite fields.

2.1.2 Order of Field

For finite prime field GF(p) of order p, all elements lie in the same residue class modulo p. This implies that $a = b$ in GF(p) is the same as $a \equiv b(modp)$ for all $a, b \in p$. It is to be noted, however, that a ring of residue class modulo 4 is not a field even though $2 \star 2 \equiv 0(mod4)$ holds, since 2 does not have a inverse, the ring of residues modulo 4 is distinct from the finite field with four elements. Finite fields are therefore denoted GF(p^n), instead of GF(q), where $q = p^n$.

For finite field GF(p^n) where $n > 1$ the integers mod p^n do not form a field. This can be explained as the congruence $px \equiv 1(modp^n)$ do not have a solution, so it is not divisible by p even though $p\,! \equiv 0(modp^n)$. Therefore, more complicated constructions are used to produce fields with p^n elements. However, in this research work our focus is on arithmetic in GF(p^n) with $n = 1$.

2.1.3 Arithmetic in GF(p)

This section presents a brief introduction to finite field arithmetic, additional information can be found in[30] . Finite field arithmetic is important and relevant because the work described in this research employs operations performed in Finite Field. Finite fields are used in a variety of cryptographic algorithms.

Finite field arithmetic is different from standard integer arithmetic as it has limited

number of elements and all operations performed in the finite field result in an element within that field. Therefore, all operations performed in finite field are similar to additions, subtractions, multiplications of integers except they are followed by modulo p. For example, consider a field GF(5). In order to perform addition of $3, 4 \in 5$ we need to perform $(3 + 4) mod\ 5$ which is reduced to $2(mod 5)$. This requires two step operation:

- calculate $4 + 3 = 7$;
- since $7 > 5$, subtract $7 - 5 = 2$;

Therefore, the result is $2(mod 5)$, and this operation is called Modular Addition. The same principle is used for Modular Subtractions.

Modular multiplication is the most critical operation in finite field arithmetic after modular division. The results of the operation $a * b = ab$ usually results in $ab \in [0, (p-1)^2]$. The reduction of such large product requires dividing by p such that $q = (ab/p)$ and $r = ab - qp \in (0, p)$. Here, q is the quotient and r is the remainder of the division. Many algorithms have been introduced and studied that accounts for faster Modular Multiplication techniques as modular multiplication finds it's repeated use in RSA cryptography, elliptic curve cryptography and other cryptography algorithms. Since the operation is required to be performed many times due to the nature of the algorithms and it's very expensive and requires lot of resources, a extensive study has been done in this field to improve the computation capacity of systems performing such operations. One of the most recommended algorithm for Modular Multiplication, till date is the Montgomery Multiplication. Montgomery multiplication interleaves steps of multiplication and reduction. Some of the work in this field can be found in[23][31]. An Elliptic Curve Processor employing such Multiplier is presented in [32, 33].

Modular Division is the most critical operation in the finite field arithmetic. Division in finite field requires calculating the quotient, multiplication and addition modulo p. Modular division is used intensively in many cryptographic algorithms and have a high computational complexity. Division is generally performed by multiplying the numerator with

inverse modulo p which is calculated using Extended Euclidean Algorithm.

The work introduced in this research, implements a powerful Modular Division Unit based on Extended Binary GCD Algorithm, therefore subsequent sections explains briefly the Extended Euclidean Algorithm and the Extended Binary GCD Algorithm that are key tool in performing modular division.

2.1.4 Extended Euclidean Algorithm and Modular Inverses

Given a number n and modulus m, modular inverse of m exists if $gcd(n, m) = 1$, in other words if n and m are cooprime. Extended Euclidean Algorithm(EEA) is an extended version of Euclidean Algorithm used for finding the greatest common divisor(GCD) of two number a and b as well as the integers x and y in Bezout's identity

$$ax + by = gcd(a, b) = 1$$

The Extended Euclidean Algorithm is presented in Algorithm 1[34]. The EEA algorithm provides output in the following form,

$$1 = gcd(a, b) = ax + by;$$

Arranging the terms we get:

$$a * x + b * y = 1$$

or $b^{-1} = y(mod\ a)$; This is confirmed by checking $b * y = 1(mod\ a)$;

The division step in Algorithm1 to compute the remainder is computationally expensive and alternatives to it are being studied and documented. Most of the efficient methods of implementing remainder calculation on hardware are done by digit or bit serial methods. In general, this is achieved by observing the dividend and then performing a single step shift operation. The Extended Binary GCD Algorithm provides the alternative to the Extended Euclidean Algorithm by completely avoiding any division steps.

Algorithm 1 Extended Euclidean Algorithm

Input: Integer or polynomials a and b, two vectors $(1, 0)$and $(0, 1)$
Output: g, x, y such that $ax + by = g$ where $g = gcd(a, b)$
Step1: $a_0 \leftarrow a$; $b_0 \leftarrow b$; $y_0 \leftarrow 0$; $y \leftarrow 1$; $x_0 \leftarrow 1$; $x \leftarrow 0$;
Step2:
if $a_0 > b_0$
$q = [a_0/b_0]$;
$g = a_0 - qb_0$;
$temp = (x_0, y_0) - q * (x, y)$;
$a_0 \leftarrow g$;
$(x_0, y_0) \leftarrow temp$;
else
$q = [b_0/a_0]$;
$g = b_0 - qa_0$;
$temp = (x, y) - q * (x_0, y_0)$;
$b_0 \leftarrow g$;
$(x, y) \leftarrow temp$;
end if;
$g = a_0 - qb_0$;
$temp = (x_0, y_0) - q * (x, y)$;
Step3: repeat step2 until $g = 0$;
Step4: $g = b_0$;
Step5: Return (g, x, y)

2.1.5 Extended Binary GCD Algorithm

Extended Binary GCD Algorithm is an efficient way to perform Modular Division. As the name implies, it's the extended version of Binary GCD algorithm. The Binary GCD algorithm calculates GCD of two non-negative integers and has an advantage over the classic Euclidean algorithm as it replaces the divisions and multiplications with shift that are more efficient and cheaper when using binary representation on digital systems especially on embedded systems. A basic variant of Binary GCD algorithm is presented in Algorithm 2. The following identities are applied to compute GCD of two numbers:

1. $gcd(0, b) = b$; and $gcd(a, 0) = a$;
2. If a is even and b is even, then $gcd(a, b) = 2 * gcd(a/2, b/2)$;
3. If a is even and b is odd, then $gcd(a, b) = gcd(a/2, b)$; Same is true if a is odd and b is even.
4. If a and b are odd, then g$cd(a, b) = gcd((a - b/2), b)$ if $a > b$. If $b > a$ then $gcd(a, b) = gcd((b - a/2), a)$;

The Extended Binary GCD Algorithm extends algorithm 1 to compute GCD of two numbers and adds more such simple shift and comparison steps to compute division of two integers modulo p. The division step is eliminated at the expense of comparison and few

Algorithm 2 Binary GCD Algorithm(Stein's Algorithm)

Input: a and b are positive integers
Output: $gcd(a, b)$
Step1: while $a \neq 0$ do
while $a_0 = 0$ do
$a = a/2$; – shift right
if$(a < b)$ then
$b = n$;
$b = a$;
$a = b$; //swap
$a = a - b$;
Step2: return b;

Algorithm 3 Extended Binary GCD Algorithm

Input: x, y, p; n is the number of bits
Output: z, such that $z = (x/y) mod p$;
Step1: $a \leftarrow y$; $b \leftarrow m$; $u \leftarrow x$; $v \leftarrow 0$; $\rho \leftarrow n$; $\delta \leftarrow 0$;
Step2: while $\rho > 0$do
while $a$is even do
$a = a/2$; $u = u/2\ mod m$; $\rho = \rho - 1$; $\delta = \delta - 1$;
end while;
if $\rho < 1$ then
$temp = b$;
$b = a$;
$a = temp$;
$temp = u$;
$v = u$;
$v = temp$; //swap
$\delta = -\delta$;
end if;
if $(a + b) \equiv 0 (mod\ 4)$ then $q = 1$ else $q = -1$;
$a = (a + q * b)/4$;
$u = (u + q * v)/4 (mod\ m)$;
$\rho = \rho - 1$;
$\delta = \delta - 1$;
end while;
Step3:if $b = 1$ then $z = v$ else $z = m - v$;

shifts performing division and multiplication by 2 or multiple of 2. The Extended Binary GCD Algorithm is rewritten in Algorithm 3 and performs the calculation of $x/y\ mod\ p$ [11].

The following identities of binary GCD are applied:

1. Identity 1: If a is even and b are even, then $gcd(a, b) = 2 * gcd(a/2, b/2)$;
2. Identity 2: If a and b are odd, check if $a + b$ is divisible by 4. If so, then $gcd(a, b) = gcd((a+b)/4, b)$; If not, then $a-b$ is divisible by 4. Therefore, $gcd(a, b) = gcd((a - b/4), b)$;

In the following algorithm, ρ indicates the minimum of upper bounds of $|a|$ and $|b|$; δ is used to check difference between the upper bounds of a and b and if it is less than 1, then

the two variable are swapped and the next operation is performed on the greater of the two variables. There are four variables a, b, u and v. In algorithm 3, [a,b] represent the GCD sequence while [u,v] represent the modular quotient or the result sequence. Additional information on the Extended Binary GCD Algorithm is found in[8]. Algorithm 3 is a powerful tool to compute Modular Division as it makes use of simple shifts and a comparison to perform the required operation. The only problem that seem to dominate the algorithm is the comparison required at each step of iteration. There have been many ideas and implementations documented till date to speed up the operation. One of the most efficient implementation done in this field is by Takagi et al. in[11] wherein the comparison is avoided by replacing it with the inspection of least significant bit(LSB) from shift registers and variables. The algorithm will be discussed in detail in Chapter 5.

2.2 Elliptic Curves

Elliptic Curves were first introduced into cryptography by Miller and Koblitz in the year 1985[5]. Later on, it was shown that elliptic curves can be used to factor integers which was introduced by Lenstra[37]. Elliptic curves modulo a prime p are of significant importance in public-key cryptography and have gained popularity in public-key cryptography due to use of smaller key size and computationally efficient algorithms. For example, an elliptic curve cryptography system with 160-bit key can provide the same level of security as RSA with 1024-bit key length. In this section, a brief introduction to elliptic curve arithmetic is provided which is relevant to the work presented in this research. Additional information can be found on elliptic curves and their applications in cryptography in[38].

2.2.1 Elliptic Curve Arithmetic over real numbers

Definition2.1: An elliptic curve is defined as the set E of solutions $(x, y) \in \mathbb{R} \times \mathbb{R}$ **to the equation** $y^2 = x^3 + ax + b$ **together with the point at infinity** $\mathcal{O}$ **where** $a, b \in \mathbb{R}$ **are constants such that** $4a^3 + 27b^2 \neq 0$**[34].**

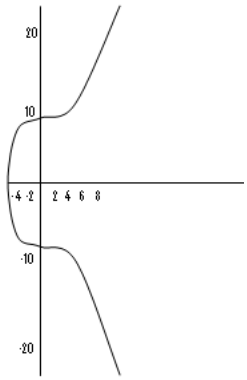


Figure 2.1: Elliptic Curve E: $y^2 = x^3 + 73$

Figure 2.1 depicts an elliptic curve $y^2 = x^3 + 73$ where $a = 0$ and $b = 73$. For different values of a, b a different elliptic curve is generated. It is proved that the condition $4a^3 + 27b^2 \neq 0$ is necessary and sufficient to prove that the equation $y^2 = x^3 + ax + b$ has three distinct roots. Such an elliptic curve with distinct roots is called a non-singular elliptic curve and forms an Abelian group with respect to a binary operation. If the operation is Addition, then the point at infinity $\mathcal{O}$ is the additive identity element such that $P+\mathcal{O} = \mathcal{O}+ P = P$, for all $P \in E$.

Elliptic Curve Arithmetic usually denotes the addition operation and involves obtaining a point on the curve as a result of addition of other points on the curve. For example, given two points P_1 and P_2 on a curve, we can obtain the third point P_3 on E using the folowing (Figure 2.2):

Let the two points on an elliptic curve be P_1 and P_2:

- Draw a straight line L across the curve such that the line intersects the curve on the points $P_1$and P_2. If $P_1 = P_2$; then draw a line tangent to the curve at P_1.

- As seen in the figure above, the line intersects the curve on a third point say Q . The y coordinate of Q is changed from y to $-y$ to get the point P_3. The law of addition on E is defined as $P_1 + P_2 = P_3$

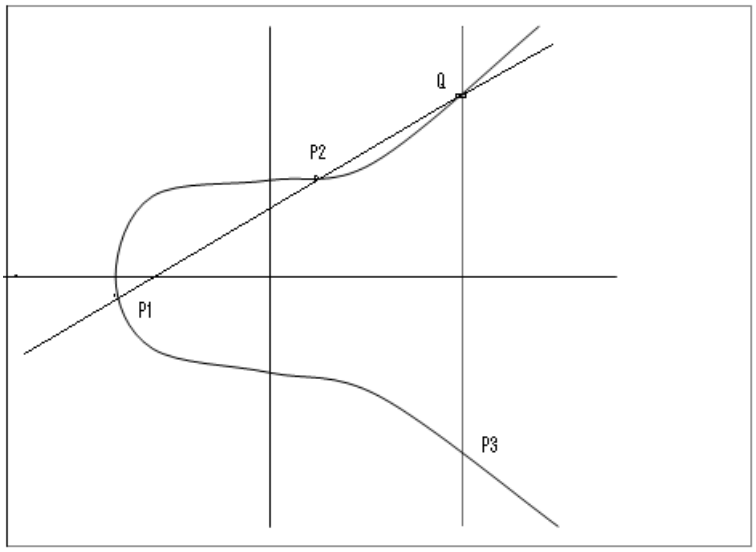


Figure 2.2: Geometric illustration for addition of points on an elliptic curve

For computation purpose, the geometrical aspects are replaced by formulas which are as follows:

Consider the points $P_{1,}P_2 \in E$; $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$. $P_1 + P_2$ can be calculated in three cases:

- $x_1 \neq x_2$. This is the case depicted in figure 2.2. First of all, calculate slope of the line L:

$$slope = m = (y_2 - y_1)/(x_2 - x_1) \text{ (2.3)}$$

The point $P_3 = (x_3, y_3)$ such that $P_1 + P_2 = P_3$ is then calculated as

$$x_3 = m^2 - x_1 - x_2 \text{ (2.4)}$$

$$y_3 = m(x_1 - x_3) - y_1 \text{ (2.5)}$$

If the slope is infinite, then $p_3 = \infty$.

- $x_1 = x_2$ and $y_1 = -y_2$. For this case, it is defined that $(x, y) + (x, -y) = \mathcal{O}$; hence, the two coordinates are inverses with respect to addition operation of elliptic curves.

- $x_1 = x_2$ and $y_1 = y_2$. This case is termed as the point double operation since $P_1 = P_2$; therefore $P_3 = 2P$. The slope is calculated as:

$$slope = m = (3(x_1^2) + a)/2y_1 \text{ (2.6)}$$

The coordinates (x_3, y_3) is calculated in the same way as in equations (2.4) and (2.5). The addition operation of elliptic curve exhibits the following properties:

- *Associative law:* $(P + Q) + R = P + (Q + R)$ where P, Q, R are points on E.
- *Commutative law:* $P + Q = Q + P$ where P, Q, R are points on E.

In other words, points on an elliptic curve E form an Abelian group with respect to the addition operation and $\mathcal{O}$ is the identity element of the group.

2.2.2 Elliptic Curve Arithmetic in Finite Field

Elliptic curves arithmetic in finite field is performed modulo a Prime p and can be defined exactly as when defined over the reals.

Definition2.2: An Elliptic Curve E is defined as the set of solutions $(x, y) \in \mathbb{Z}_\rho \times \mathbb{Z}_\rho$ **that satisfy the equation** $y^2 \equiv x^3 + ax + b\ (modp)$ **along with the point at infinity** $\mathcal{O}$**.** $a, b \in \mathbb{Z}_\rho$ **are constants such that** $4a^3 + 27b^2 \neq 0(modp)$ **and** $p > 3$**[34]**

The addition operation on elliptic curve E is defined in the same way as in previous section except all operations are performed modulo prime p. For example; addition of points $P_1(x_1, y_1)$ and $P_2(x_2, y_2)$ where $x_1 = x_2$ and $y_1 = y_2$. The slope is calculated as:

$$slope = m = (3(x_1^2) + a)/2y_1\ (mod\ p) \text{ (2.7)}$$

The coordinates (x_3, y_3) is calculated as:

$$x_3 = m^2 - x_1 - x_2 (mod\ p) \text{ (2.8)}$$

$$\mathrm{y}_3 = m(x_1 - x_3) - y_1 (mod\ p) \text{ (2.9)}$$

2.2.3 Points on the Elliptic Curve Mod p

Consider an elliptic curve E: $y^2 \equiv x^3 + ax + b(mod\ p)$ where $p \geq 5$. The number of points on E is roughly estimated as equal to P. In order to determine the points on E, one needs to look at each possible $x \in p$ and then attempt to solve the equation for y. To be accurate, the number of points on an elliptic curve defined over a finite field are bounded by the expression in equation 2.10 also known as Hasse's Theorem:

$$|N - p - 1| < 2\sqrt{p} \text{ (2.10)}$$

Here the symbol N is the number of points on the elliptic curve. Understanding the number of points on elliptic curves is important as it helps in understanding the nature of the group one is working on, which can be an important element in solving the discrete logarithm problem. This is briefly touched in the next section.

2.2.4 Properties of Elliptic Curve

The success of elliptic curves in cryptographic techniques is due to the fact that the Discrete Logarithm problem for elliptic curves is believed to be computationally infeasible. The classical discrete logarithm problem is stated as: Let p be a prime, given two numbers α, β find k such that $\alpha \equiv \beta^k(mod\ p)$. The difficulty in computing discrete logarithm lies in the representation of the group one is working on.

For example, let G be a cyclic group of order n and $\alpha, \beta \in G$. If G is represented as an additive group of $\mathbb{Z}_n$, then computing discrete logarithms in G is equivalent to solving the linear equation $ax \equiv b(mod\ n)$ where a, b are integers such that $a, b \in G$; this can be easily done by using the extended Euclidean algorithm. If G is represented as a subgroup of a multiplicative group of a finite field or as a multiplicative group of elements from $\mathbb{Z}_p$, where p is prime, then the problem can be intractable.

The elliptic curve version of the problem is stated as : Given two points $P$and Q on an elliptic curve such that $P = kQ(= Q + Q + Q + ... + Q)$; it is computationally infeasible

to calculate k if the parameters are chosen carefully. The discrete logarithm problem is difficult because elliptic curves exhibit the following:

- Since the points on the curves are represented using coordinates and the addition formulas are trivial, it becomes difficult to determine the relationships between individual elements on the curve and to determine the structure of the group the element belongs to.

- By simply changing the parameters (a, b of $y^2 \equiv x^3 + ax + b \ (mod\ p)$) of an elliptic curve, different elliptic curves are generated that provide large number of finite Abelian groups. The discrete logarithm problem is more difficult for finite Abelian groups.

It has been observed that the elliptic curves discrete logarithm problem can easily be solved for curves defined over $\mathbb{Z}_\rho$(where P is prime) that have exactly p number of points on them. One can avoid such curves. Till date there is no good attack on the discrete logarithm problem for elliptic curves. There is an analog of Pohlig-Hellman attack that works on elliptic curves[2]. However, the attack can be rendered ineffective by carefully choosing E and the point P. Therefore, one needs to carefully consider the computational as well as the security aspects while choosing a curve.

For example, in order to simplify operations on elliptic curve like point doubling and point addition the constant a can be chosen as equal to 0. This eliminates one addition in equation (2.6). The parameters a and b are also important in determining the size of the group.

2.2.5 Point Multiplication on Elliptic Curve

In order to compute a multiple kP of an elliptic curve point P where k is a positive integer, double and add algorithm(Algorithm 4) is used. In comparison to the arithmetic of integers, a squaring operation on a integer $\alpha \rightarrow \alpha^2$ is replaced by the doubling operation $P \rightarrow P^2$ on

Algorithm 4 Double-And-Add(or Subtract) Algorithm

Input: Point $P(x, y)$ on elliptic curve of order n; multiplier $k > 0$ and it's binary representation $(k_{l-1}, ..., k_0)$
Output: Point $Q(x_3, y_3) = kP$ on elliptic curve as a result of point double or point add or both
Step1: $Q \leftarrow$P
for $i \leftarrow l - 2$ downto 0 do
$Q \leftarrow 2Q$;
if $k_i = 1$ then
$Q \leftarrow Q + P$;
end if;
return(Q);

elliptic curve and multiplication of two integers $\alpha \times \beta$ is replaced by point addition $P_1 + P_2$ on elliptic curves.

The Double and Add algorithm is illustrated with an example below :

Consider an elliptic curve E: $y^2 \equiv x^3 + x + 7206 (mod\ 7211)$. Let a point on the curve on be $P_1 = (3, 5)$. In order to compute $6P$ where $k = 6$ the following double-add algorithm is performed: $6P = 2(2P + P)$; there are total of 2 point doubles and 1 point addition operation. Therefore, the operations include first performing a point double $(2P)$, followed by a point addition $(2P + P)$ and then a final point double to provide the result of point multiplication when the multiplier is $k = 6$.

2.3 Coordinate Representation

Elliptic curve points can be represented using multiple coordinate system. The Elliptic Curve point multiplication discussed in the previous section uses affine coordinates, however, it is best suited using projective coordinates. Point multiplications are computed by performing repeated point additions and point doubles, and if the value of the multiplier k is large which is usually the case, it results in a number of iterative point double and point addition operations. As operation of point multiplication requires inversion, repeated inversion becomes computationally expensive and inefficient.

The advantage in using projective coordinates is that it eliminates inversion modulo p in the operation of point addition and point double. The following explains projective coordinate representation of elliptic curve arithmetic:

In case of projective coordinates, the point $(x, y) \in E$ in the affine coordinates is mapped to the point $(x, y, z) \in E, z = 1$. The projective coordinates are converted to affine coordinates by dividing by Z which results in $(x/z; y/z; 1)$. Using the projective coordinates, the addition of the two points on elliptic curve $P_1 = (x_1, y_1, z_1)$ and $P_2 = (x_2, y_2, z_2)$ is shown as;

- for $P_1 \neq P_2$:

$$x_3 = AD;$$

$$y_3 = CD + A^2(Bx_1 + Ay_1);$$

$$z_3 = A^3 z_1 z_2; \text{ (2.10)}$$

where $A = x_2 z_1 + x_1 z_2$, $B = y_2 z_1 + y_1 z_2$, $C = A + B$ and
$D = A^2(A + az_1 z_2) + z_1 z_2 BC;$

- for $P_1 = P_2$:

$$x_3 = AB;$$

$$y_3 = x_1^4 A + B(x_1^2 + y_1 z_1 + A);$$

$$z_3 = A^3; \text{ (2.11)}$$

where $A = x_1 z_1$, $B = bz_1^4 + x_1^4;$

However, using projective coordinates results in increasing the number of modulo multiplications required per point scalar multiplication (16 per point addition, 10 per point doubling). Additionally, the projective coordinates need to be converted back to affine

coordinates for the final result; this still requires an inversion over $GF(p)$. In Affine coordinates, for Elliptic Curve point addition we need to perform 2 multiplications, 1 modular division and 6 modular addition and subtraction. For point doubling, 3 multiplications, 1 modular division and 7 additions/subtractions need to be performed. In our architecture, we have defined elliptic curves using affine coordinates to take advantage of the lesser number of modulo multiplications and concentrated on implementing a powerful modular divider.

Chapter 3

Key Agreement Scheme

3.1 Introduction

With the advent of technology there are many applications that require secure communication. Most of the cryptosystems in the market are based on private-key cryptography due to the fact that public-key cryptographic algorithms are much slower than the private-key algorithms. Thus, one needs to consider means for the exchange and establishment of keys for private-key cryptosystem. In Key-Agreement schemes(KAS), users can exchange and establish keys by means of an interactive protocol. Key-Agreement schemes are public-key cryptography based and are also used to initiate a conversation between two introduced users. The following section briefly explains the Diffie-Hellman Key Agreement Scheme which is one of the first and the best known KAS. As mentioned earlier, elliptic curves have gained wide popularity in cryptographic techniques due to the fact that systems employing elliptic curves are capable of providing security at the same level as RSA cryptosystems with much smaller key lengths. This makes ECC suitable for hardware implementation in embedded systems where area and power constraints exist. The subsequent section describes the application of elliptic curves in Diffie-Hellman Key Agreement Scheme which is the basis of the research work presented in this thesis.

3.2 Diffie Hellman Key Agreement Scheme

Diffie Hellman KAS was the first public-key cryptography scheme published in 1976. The Diffie Hellman KAS protocol is presented in Protocol 3.1. Let there be two users A and B who wish to initiate a key exchange process, suppose that (G,.) is a group closed under the multiplication operation and $\alpha \in G$ is an element of the order n, both the parameters $(G, .)$ and a are published in public domain. The protocol below presents the Diffie-Hellman KAS[34].

Protocol3.1 Diffie-Hellman Key Agreement Scheme

1. User A chooses a random number a_a such that $0 < a_a < n - 1$ and computes $b_a = \alpha^{a_a}$ and sends this value to user B;

2. User B chooses a random number b_b such that $0 < b_b < n - 1$ and computes $a_b = \alpha^{b_b}$ and sends this value to user A;

3. A receives a_b and computes $K = (a_b)^{a_a}$;

4. B receives b_a and computes $K = (b_a)^{b_b}$;

At the end of the session, the two users have calculated the same key:

$$K = \alpha^{b_b a_a} = \alpha^{a_a b_b} = CDH(\alpha, b_a, a_b)$$

Protocol 3.1 is also illustrated in Figure 3.1. The Computational Diffie Hellman(CDH) problem is based on the assumption that the discrete logarithm problem is intractable within a cyclic group, that is, a passive adversary is not able to calculate or compute any information about the key K. In case of an active adversary, if there is an intruder in the middle of the two users while a key process is being initiated then the intruder can easily establish a key with both the users by intercepting messages between user A and B(Figure 3.2). This is known as the Intruder-in-the-middle attack and it renders the Diffie-Hellman key agreement scheme to be vulnerable. In such case, one needs to make sure that the key exchange process is initiated only between the intended user and not a intruder in the middle. This is done by defining authenticated agreement schemes wherein the users first authenticate themselves and then initiate the process after validating their identification or authority.

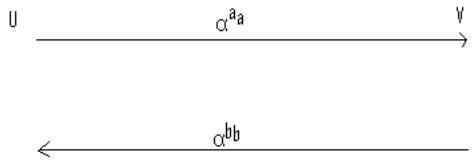


Figure 3.1: Diffie-Hellman Key Exchange Protocol

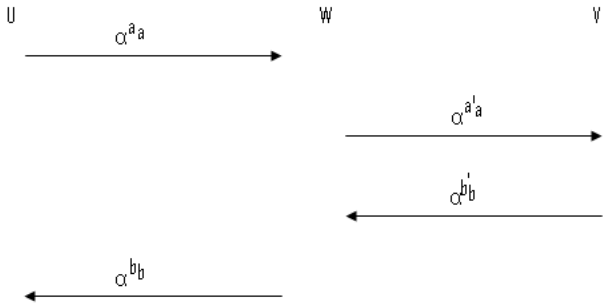


Figure 3.2: Intruder-in-the-middle Attack

3.3 Elliptic Curve Diffie Hellman Key Agreement Scheme

In order to take advantage of the discrete logarithm problem on elliptic curves which is computationally infeasible, the Diffie-Hellman Key Agreement Scheme can be implemented using elliptic curves[3][35]. The elliptic curve(EC) based Diffie-Hellman key protocol is illustrated in Figure 3.3.

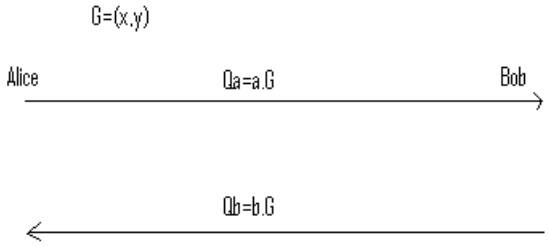


Figure 3.3: EC-based Diffie-Hellman Key Exchange KAS

The basic steps in the algorithm includes:

1. Alice and Bob want to exchange a key. They carefully chose an elliptic curve E and a public base point G on the elliptic curve. $G(x, y)$ is published.

2. Alice chooses a random integer say a and Bob chooses a random integer b. The random integers are kept private.

3. Alice computes a new point on the elliptic curve by performing scalar multiplication $Q_a = a.G$ and publish it or sends it to Bob. Bob computes $Q_b = b.G$ and publishes it.

4. Due to the discrete logarithm problem of elliptic curve, it is computationally infeasible to compute a or b even though Q_a and Q_b are public.

5. At the user end, Alice takes Q_b, and computes a new point on elliptic curve $K = aQ_b$. Similarly, Bob takes Q_a and computes a new point on the elliptic curve, $k = bQ_a$.

At the end of the session, Alice and Bob have computed the same point multiplication on elliptic curve modulo a prime p. A passive adversary will not be able to compute the shared K as the secret keys a and b are not known. Once the session key is established, both the parties Alice and Bob can take advantage of fast private-key encryption algorithms available.

Chapter 4

Supporting Work

Elliptic Curves were first introduced in cryptography in 1985 and since then there have been significant research in this field toward the implementation of elliptic curves in various cryptographic techniques and algorithms. Ample research have been done to improve the computations required to perform point multiplication using different coordinates representation and algorithms.

Two main articles have been studied in this research [18] and [9] that have performed an FPGA implementation of elliptic curve ALU in GF(p). Both the implementations are Montgomery multiplication based and use projective coordinates to represent the point on the curve.

4.1 Modular Division

The elliptic curve arithmetic in finite field for the computation of point multiplication requires modular multiplication, division, addition and subtraction. Modular division is known to be the most computationally intensive operation due to the fact that in performing point multiplication on elliptic curves a number of point double and point addition operations are performed and each operation requires a modular division. Modular division has not received a lot of attention as it can be replaced by a modular inverse followed by a multiplication. Nevertheless, there have been some significant research in this area. In this regard, two different design implementations were investigated and the final selected

design was optimized for speed. The optimization was carried at all levels of the design. Only pipelined multipliers, adders and divider are used in the entire implementation to compensate to the processing time issue.

Chapter 5

Design Methodology

In this section, an Elliptic Curve Arithmetic Logic Unit in GF(p) using field programmable gate-array technology is presented. The architecture uses affine coordinates to represent the points on the curve over $GF(p)$. The architecture presented is suitable for elliptic curve Diffie-Hellman KAS. The modular divider used in this implementation is presented in subsequent sections. A brief introduction on the arithmetic background for modular division has been previously presented in Chapter 2 section 2.1.5.

Figure 5.1 shows the architecture presented. It consists of two main components – The main Controller and the Arithmetic Unit. Classic binary representation has been used for all arithmetic operations. The Main Controller unit controls the Arithmetic Unit, determines the computation of kP and interacts with the host system. The Arithmetic Unit is responsible for performing Elliptic Curve point addition and double, this includes computing various finite field arithmetic operations like modular addition, multiplication, modular division and the non-modular addition, subtraction, multiplication and division computations. The Arithmetic Unit comprises of pipelined multiplier, pipelined divider, one or more adders, subtractor and constitute the modular division, point double, point addition and point multiplication modules. The size of the field is variable and can be reprogrammed when implemented using FPGA.

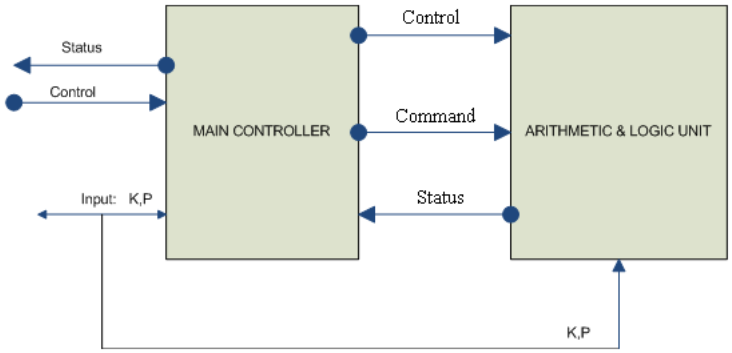


Figure 5.1: Elliptic Curve ALU Architecture

5.1 Main Controller

The Main Controller controls the entire point multiplication operation. It is responsible for decoding the multiplier k and determining how many iterations of point double and add needs to be performed, sending control signal to arithmetic unit to perform point double or point addition, and carrying out the double-add algorithm (Algorithm 4).

The main controller is also responsible for coordinating interaction with the host system and synchronizing the input and output system. Following is a typical sequence of steps performed when a user inputs data to generate coordinates of a point on elliptic curve as a result of point multiplication.

Firstly, an elliptic curve is chosen and a large prime number m is chosen for operations modulo prime m. The user then loads the coordinates (x, y) of the point P on the elliptic curve defined over $GF(m)$, and the multiplier $'k'$ as input to the Main Controller and initializes the controller. The main controller initializes and sends signal to Arithmetic Unit to start processing. The first operation is the point double. For the next operations, the controller is responsible for decoding the multiplier k and to determine if a point double or a point addition needs to be performed.

The main controller is also responsible for loading the arithmetic unit with the correct inputs and storing results from previous operation of point double or point add. The

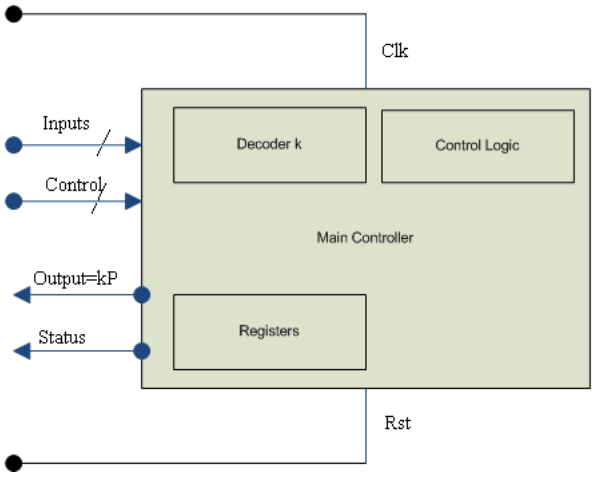


Figure 5.2: Main Controller Unit

main controller outputs the coordinates of kP(Figure 5.2). As seen in the figure, the Main Controller unit comprises of the control logic, decoder for k and registers.

5.2 Arithmetic Unit

The Arithmetic Unit is the main processing unit of the architecture. All the arithmetic in the Galois Field is performed modulo the field prime p. The performance of the unit determines the performance of the entire design. The unit is responsible for carrying out operations like modular division, modular addition and normal addition, subtraction, division and multiplication. Figure 5.3 shows functional diagram of Arithmetic Unit presented in our work. In addition to the arithmetic operation, the Arithmetic Unit also stores temporary values and precomputed values in registers.

A typical sequence of operation when the Arithmetic Unit is initiated by the Main Controller includes - Reading the inputs which are coordinates of two points on Elliptic Curve, determining the slope between the two points, calculating the coordinates as a result of point addition or point double operation on Elliptic Curve, and sending the results to the Main Controller for Output. Two main algorithms namely Extended Binary GCD algorithm for Modular Division and Double-and-add(or subtract) algorithm for elliptic curve point

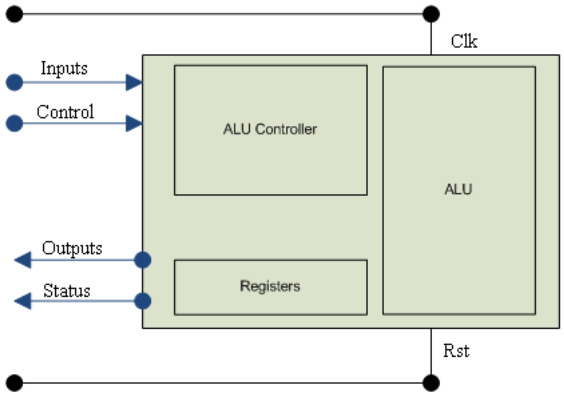


Figure 5.3: Block Diagram of Arithmetic Unit

multiplication forms the basis of these operations.

5.2.1 Modular Division

The heart of the architecture is the modular divider unit. We have implemented the modified version of Extended Binary GCD Algorithm for the modular division operation. The algorithm was proposed by Takagi et al in[11]. Two different design implementations were investigated for this purpose and the algorithm by Takagi was selected due to it's efficiency. Takagi replaced the comparison of registers with that of comparison of the least significant bit(LSB). The selected design was implemented using a different hardware design approach.

The modular algorithm for hardware implementation is rewritten in algorithm 5. For an n-bit modulus, the implementation performs each iteration in one clock cycle. Takagi has implemented the algorithm using radix-2 signed digit representation.

The presented algorithm is slightly different from the original algorithm. We have modified the output logic and added a modular correction unit since the output step in the original algorithm was relevant to radix-2 signed digit representation. The design implementation has been done using classical binary representation to save on the conversion unit and SD2 adders. The snippet of VHDL code for algorithm 5 is presented in Figure 5.5.

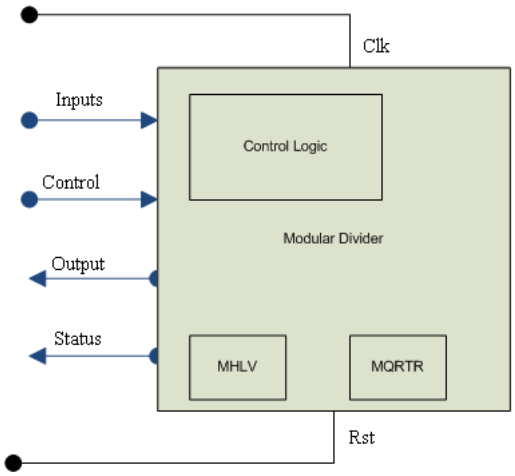


Figure 5.4: Block Diagram of Modular Divider

The results of the implementations are compared to the proposed implementation in [10] and have been presented in Chapter 7. The block diagram of the Modular Division Unit is shown in Figure 5.4.

We have implemented the control variables p and d as in the original algorithm. However, counters are used to represent the position of the bit for p and d rather than using shift registers. This directly impacted the area utilization as a 4-bit counter was used to represent a 2^{130}bit variables p and d. p is introduced to check the minimum of (α, β) where α and β are values 2^{α} and 2^{β} representing the upper bounds of $|A|$ and $|B|$. The while loop in the beginning of the algorithm controls the number of iterations to be performed and the termination condition $\rho = 0$ assures that $A = 0$. d is used to compare between A and B and is represented as $d = \alpha - \beta$. As α and β represent the upper bounds of $|A$ and $|B|$, variable d has just one bit of value 1 while the rest are $0s$. $'s'$ is a flag representing the sign of value $\alpha - \beta$. When this flag is set, the registers A and B are swapped and operations are performed on A.

The MHLV and MQRTR components in the algorithm are components that perform operation on the result sequence modulo prime m. It was previously discussed in Chapter

Algorithm 5 Hardware algorithm for Modular Division performing $Z = X/Y$ mod m)

```
Input   : x,y ∈ [1, M-1] and M        (n=number of bits)
Output: z such that z=x/y (mod M)

Step 1 : A:=Y ; B:=M ; U:=X ; V:=0 ; P=2^n ; M=M ; D=1 ; S=1
Step 2 : While P?1
                if A/4=0 then
                        A=A/4;  U=MQRTR(U,M);
                        If S=0 then
                                If D=4 then S=1;
                                If D>2 then D=D/4;
                                Else P=P/2; S=1;
                                End if;
                        Else /*S=1*/
                                D=4*D;
                                If P>2 then P=P/4;
                                Else P=P/2;
                        End if;
                Elsif A/2=0 then
                        A=A/2; U=MHLV(U,M);
                        If S=0 then
                                If D=2 then S=1;
                                        D=D/2;
                         Else /*S=1*/ D=2*D;
                                P=P/2;
                        End if;
                Else
                        if (A+B)/4=0  then q=1
                        Else q=-1;
                        if S=0 or D=1 then
                                A=A+qB; U=MQRTR(U+qV,M);
                                If S=1 then P=P/2; D=2*D;
                                Else /*S=0*/
                                        If D=2 then S=1; D=D/2;
                                End if;
                        Else /*S=1 and D>1*/
                                B=A; V=U;
                                A=A+qB; U=MQRTR(U+qV,M);
                                If D=2 then S=0;
                                        D=D/2;
                                End if;
                        End if;
                End if;
        End while;
Step 3: If X is negative, Z=M-V; If B=3 (mod 4) then V=-V;
Step 4: Z=V;
```

Algorithm 6 Hardware algorithm for Modular Division performing $Z = X/Y$ mod m)

Inputs : m: $0 < m < 2^{n-1}$Prime Number and $GCD(m, Y = 1)$; X, Y: $-m < X, Y < m$ $(Y \neq 0)$

Outputs: Z , $0 < Z < m$

Step1 : $A := Y, B := m, U := X, V := 0, m := m, P := 2^{n+1}, D := 1, s := 1$;

Step2: while $p_0 \neq 1$ do

if $[a_1 a_0] = 0$then $/ * A \equiv 0(mod 4) * /$

$A := A >> 2$; $U := MQRTR(U, M)$

if $s = 0$ then if $d_2 = 1$then $s := 1$;

if $d_1 = 0$then $D := D >> 2$;

else $P := P >> 1$; $s := 1$;end if

else $/ * s = 1 * /$

$D := D << 2$;

if $p_1 = 0$then $P := P >> 2$ else $P := P >> 1$;

end if;

elsif $[a_1 a_0] = 0$ then $/ * A \equiv 2(mod\ 4) * /$

$A := A >> 1$; $U := MHLV(U, M)$;

if $s = 0$ then if $d_1 = 1$ then $s := 1$;

$D := D >> 1$;

else $/ * s = 1 * /D := D << 1$;

$P := P >> 1$; end if;

else $/ * A \equiv 1(mod 4)$or $A \equiv 3(mod 4) * /$

if $([a_1 a_0] + [b_1 b_0] mod 4 = 0$ then $q := 1$else $q := -1$;

if $s = 0$ or $d_0 = 1$ then

$A := (A + qB) >> 2$;

$U := MQRTR(U + qV, M)$;

if $s = 1$ then

$P := P >> 1$; $D := D << 1$;

else $/ * s = 0 * /$

if $d_1 = 1$ then $s := 1$;

$D := D >> 1$;

end if;

else $/ * s = 1 and\ D > 1 * /$

$\{A := (A + qB) >> 2, B := A\}$;

$\{U := MQRTR(U + qV, M), V := U\}$

if $d_1 = 0$then $s := 0$;

$D := D >> 1$;

end if

end if

endwhile

Step3: if $([b_1 b_0] = 3$or $[b_{N-1}] = -1)$then $V := -V$;

Step4: $Z := V$;

```
1  ======================================================================
2  case sstate is                                        --state machine
3     when idle=>
4        if a(1 downto 0)     = "00" then        -- there are 3 states based
5           sstate<=SA;
6        elsif a(1 downto 0) = "10" then         -- on each condition and update state
7           sstate<=SB;                          -- where registers are updated
8        else
9           sstate<=SC;
10       end if;
11
12       When SA=> if done = '1' then sstate<=update;
13          end if;
14
15       When SB=> if done = '1' then sstate <=update;
16          end if;
17
18       When SC=> if done = '1' then sstate<=update;
19          end if;
20 ======================================================================
21 elsif select2='1' then
22    A1 <= std_logic_vector(shift_right(signed(A),1));        -- divide A by 2 with sign bit intact
23    B1 <= B;
24    V1 <= V;
25    if u(0)='0' then
26       u1<= std_logic_vector(shift_right(signed(U),1));      -- divide U by 2 if U is even
27    else
28       u1<= std_logic_vector(shift_right(signed(U+M),1));    -- perform U+M and divide by 2 is U is o
29    end if;
30       if S='0' then                                        --check sign of result
31          D1 <= D-1;                                        -- decrement counter
32          P1 <= P;
33          if D=2 then
34             S1 <= '1';                                     --set flag s
35 ============================================================
```

Figure 5.5: partial VHDL code for Modular Divider based on algorithm 5

2 that a and b are the sequence used to calculate $GCD(Y, m)$ while u and v are the sequence used to calculate the modular quotient. The algorithm first checks if a is divisible by 4. If it is divisible by 4, then it performs $a/4$ and $u/4(mod\ m)$else it checks if it is divisible by 2 and perform $a/2$ and $u/2mod\ m$.

$u/4mod\ m$ is taken care by the $MQRTR(u, m)$ component. It performs the following:

- If $m \equiv 1(mod4)$ then it checks if $u(mod\ 4)$ is $0, 1, 2, 3$ and performs $u/4$, $u - m/4$, $u + 2m/4$, $u + m/4$ respectively.
- If $m \equiv 3(mod4)$ then it performs $u/4$, $u + m/4$, $u + 2m/4$, $u - m/4$, if $u(mod4)$ is $0, 3, 2, 1$ respectively.

In case of the halving operation, the MHLV performs $u/2$ if u is even and $(u+m)/2$ if u is odd. For the case when A is not even, the algorithm checks $A + B$ is divisible by 4 based on identity 2 in Chapter 2 section 2.1.5. The algorithm implementation is accelerated by checking the last two bits of variables involved for divisibility by 4.

5.2.2 Double-And-Add Algorithm

The double-and-algorithm of elliptic curve scalar multiplication is embedded in the Main Controller(MC) and the ALU. The MC is responsible for the control logic while the ALU performs the computations.

Chapter 6

Hardware Implementation

Development of the designs was carried out using primarily Xilinx ISE 10.1 and ModelSim SE 6.3. Performance path-finding and timing measurements were taken from Xilinx ISE post and route simulations and the area utilization were deduced from post synthesis results. Simulations were run through ModelSim SE first for each component used from adder level to the entire design. Bottom-Up approach was made to design each unit in the architecture.

The following sections give implementation details of each block separately.

6.1 Controller

The main controller delegates the point multiplication operation. It is a finite state machine and includes control signals for different operations. It takes in as the input the parameters of the elliptic curve namely a and modulus m, the coordinates of point on the curve (x, y) and the multiplier k. The output of the controller are the coordinates of the new point (x_3, y_3) as a result of point multiplication and the control signal $done$ to signal the host system that the computation successfully completed. The control signals in the controller are used to communicate with the arithmetic unit, and to re-initialize the arithmetic unit with new inputs when the results from the previous computations are ready. The controller also communicates with the logic unit responsible for decoding the multiplier k. Table 1 shows an example of elliptic curve scalar multiplication with the following parameters:

x_1	y_1	$Slope_1$	x_2	y_2	$Slope_2$	x_3	y_3
7120	2230	2489	1054	3421	5781	2079	5707

Table 6.1: Scalar Multiplication by algorithm4

- Input: E: $y^2 = x^3 + x + 7222(mod\ m)$; $a = 1, b = 7222, m = 7211$; $G(x, y)$:$x = 7120, y = 2230$; Multiplier $p = 4$;
- Output: $4G$= $G(x_3, y_3)$;
- $4G = 2(2G)$;

Detail calculation for the above case is shown below:

- First $2P$ is calculated $x = 7120$, $y = 2230$. The equations are taken from chapter 2 section 2.2.1.

$$slope = m = (3(7120^2) + 1)/2 * 2230(mod 7211) = 152083201/4460(mod\ 7211) = 2489(mod 7211);$$

$$x_3 = 2489^2 - 7120 - 7120(mod 7211) = 1054(mod 7211);$$

$$y_3 = 2489(7120 - 1054) - 2230(mod 7211) = 3421(mod 7211);$$

- $4P = 2(2P)$: $x = 1054$, $y = 3421$;

$$slope = m = (3(1054^2) + 1)/2 * 3421(mod 7211) = 3332749/6842(mod 7211) = 5781(mod 7211);$$

$$x_3 = 5781^2 - 1054 - 1054 = 2079(mod 7211);$$

$$y_3 = 5781(1054 - 2079) - 3421 = 5707(mod 7211);$$

Functional Simulation of the 32-bit controller with the input parameters in Table 6.1 is shown in Figure 6.1.

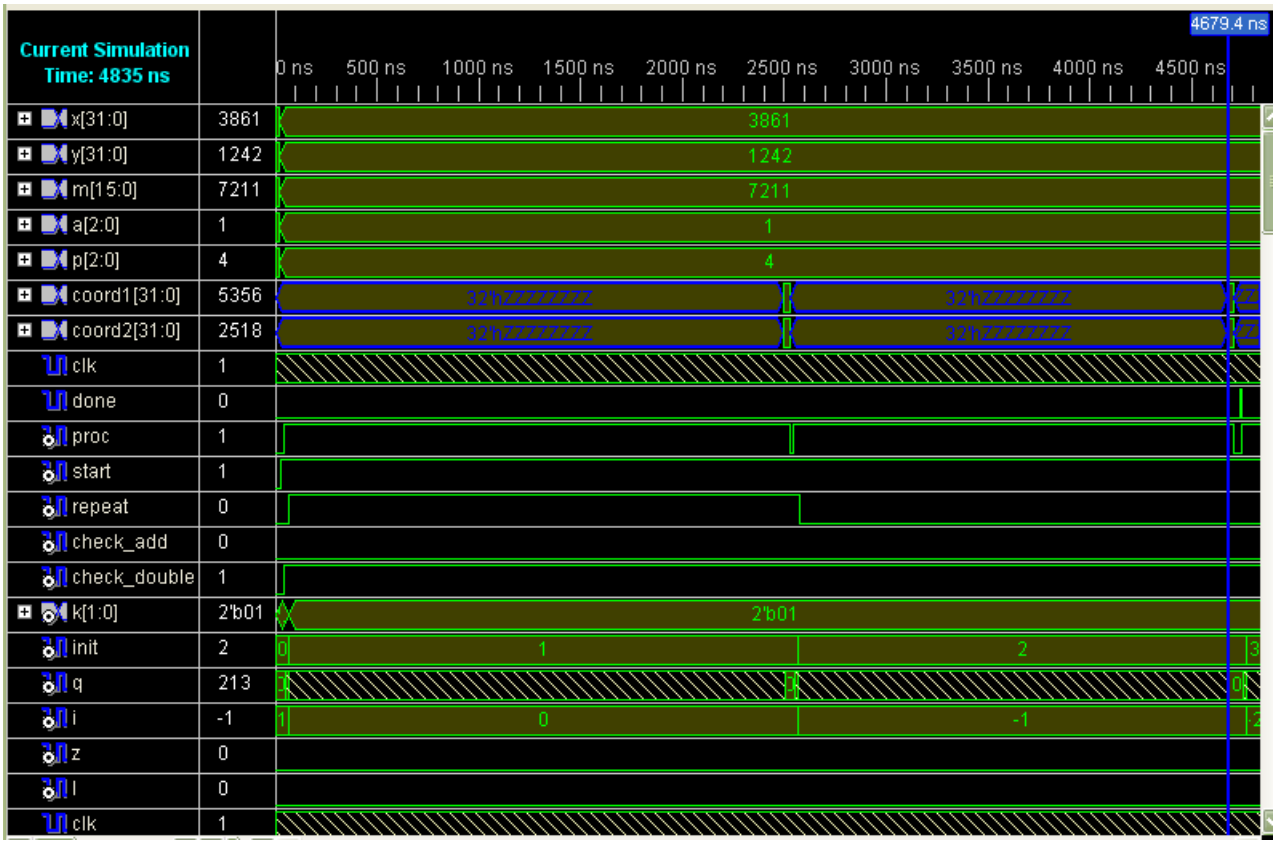


Figure 6.1: Simulation results of the Main Controller

The figure above is the simulation result for the main controller performing Elliptic Curve Scalar multiplication for the parameters:: E:$y^2 \equiv x^3 + ax + b (mod\ p)$; $a = 1$, $b = 7206$, $p = 7211$, $x = 7120$, $y = 2230$, $k = 4$::$kG = 4(7120, 2230) = (coord1, coord2 = (2079, 5707)$

6.2 Arithmetic Logic Unit(ALU)

In this section, the implementation details for each logic unit performing modular or non-modular arithmetic are discussed. The architectures described below comprise the $GF(p)$ ALU presented in this thesis.

The $GF(p)$ ALU is a finite state machine with in-built ALU Controller. The controller is responsible for driving the state machine, sending control signals to the arithmetic unit and updating the status of the logic unit to the main controller. The ALU is divided into three main units: Modular Divider, Processing Unit and Non-modular ALU. Due to computational load on the modular divider, it was not included in the Processing Unit along with other modular arithmetic units.

6.2.1 Modular Divider

The modular divider is the heart of the ALU capable of performing $(X/Z)\ (modM)$ for n bits where M is a large prime number in $GF(M)$. The modular divider is responsible for calculating the slope between the two points on elliptic curve modulo a prime p. The algorithm used in implementing the divider was discussed in previous chapter. This section explains the implementation details and the technique used to optimize the architecture for speed. The basic difference between the implementation in[11] is the use of normal binary representation to perform all the operations as opposed to SD2 representation in[11]. This results in lesser area by preventing use of SD2 adders. The divider consists of 7 registers as input/output selections namely A, B, D, P, U, V, M initialized to $Y, M, 1, 2^n, X, 0, M$ respectively and a control logic possible for driving the machine between various states.

The divider was divided into three main components based on the selection if the content of register A is: divisible by 4, divisible by 2 or none. The block diagram of the divider is shown in Figure 6.2. The control logic is also responsible for interacting with each component and updating the temporary registers with results from each component at the end of each iteration. Each iteration in the algorithm takes about 3-5 clock cycles depending

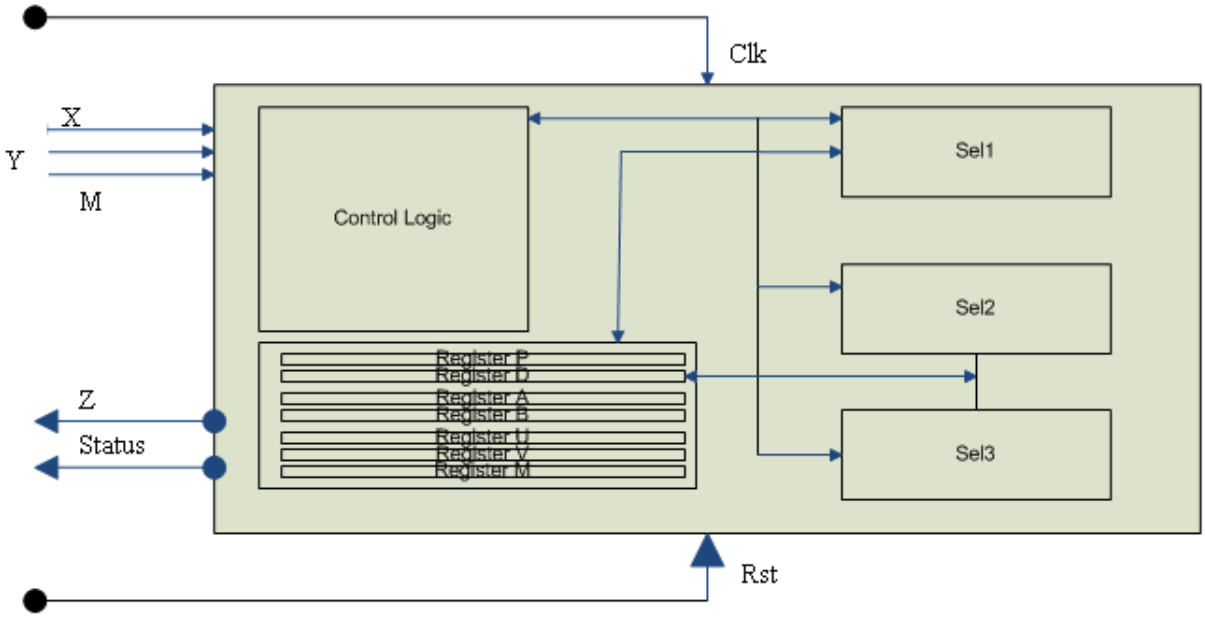


Figure 6.2: Block Diagram of Modular Divider

on the component operating on the data. Further optimizations were achieved by using the optimized adders on FPGA.

Table 6.2 shows an example of the modular divider for $X = 123, Y = 111, M = 11$.

The resulting functional simulation is shown in Figure 6.3.

6.2.2 Processing Unit

The processing unit is responsible for performing all the arithmetic operations. It comprises of a control logic, modular adder/subtractor, modular correction unit, pipelined multiplier and pipelined divider besides adders, subtractors and comparators.

6.2.2.1 Modular Adder/Subtractor

The modular add/subtractor integrates the operation of addition/subtraction and outputs the result modulo a prime M. This particular unit finds it's use at the time of calculating the coordinates of a point addition or point double operation. The modular adder logic usually

M= $[00001011]_2$, X= $[01111011]_2$, Y= $[01101111]_2$

Sel	A	B	P	D	S	U	V
	01101111(111)	00001011(11)	1000000000	0000000001	1	01111011(123)	00000000(0)
Sel3	00011001(25)	00001011(11)	0100000000	0000000010	1	00011100(28)	00000000(0)
Sel3	00001001(9)	00011001(25)	0100000000	0000000001	1	00000111(7)	00011100(28)
Sel3	11111100(-4)	00011001(25)	0010000000	0000000010	1	11111000(-8)	00011100(28)
Sel1	11111111(-1)	00011001(25)	0000100000	0000001000	1	11111110(-2)	00011100(28)
Sel3	00000110(6)	11111111(-1)	0000100000	0000000100	0	00001100(12)	11111110(-2)
Sel2	00000011(3)	11111111(-1)	0000100000	0000000010	0	00000110(6)	11111110(-2)
Sel3	00000001(1)	11111111(-1)	0000100000	0000000001	1	00000010(2)	11111110(-2)
Sel3	00000000(0)	11111111(-1)	0000010000	0000000010	1	00000000(0)	11111110(-2)
Sel1	00000000(0)	11111111(-1)	0000000100	0000001000	1	00000000(0)	11111110(-2)
Sel1	00000000(0)	11111111(-1)	0000000001	0000010000	1	00000000(0)	11111110(-2)
-V							00000010(2)

Table 6.2: Z=123/111(mod 11)

The table above presents step by step illustration of the algorithm in [11] for $x = 123$, $y = 111$, $p = 11$ resulting in $z = x/y(mod\ p) = 2$

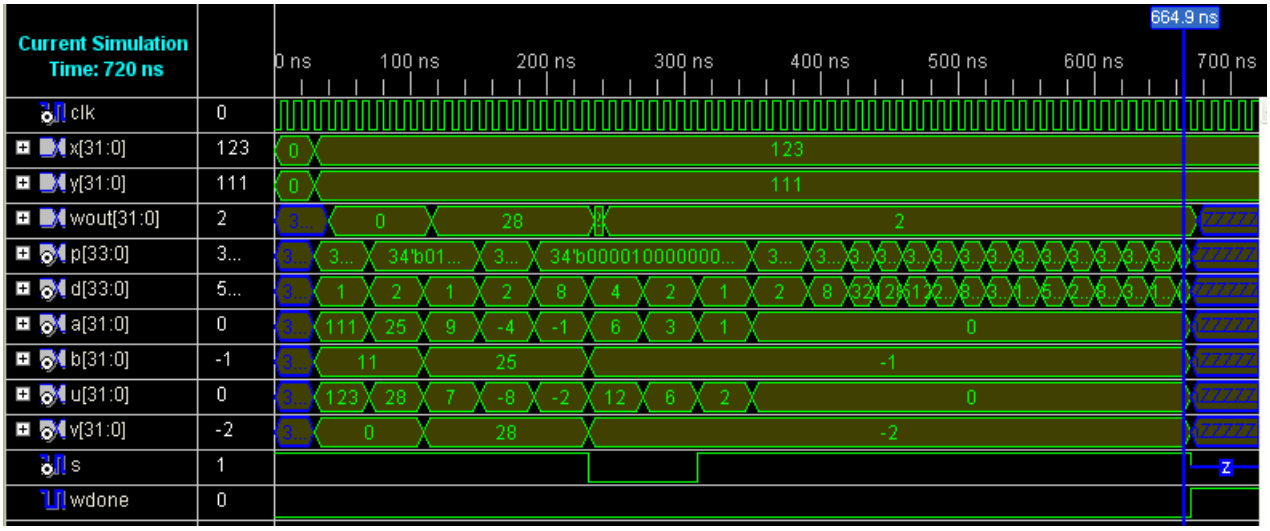


Figure 6.3: Functional Simulation for modular divider in Table 6.2.

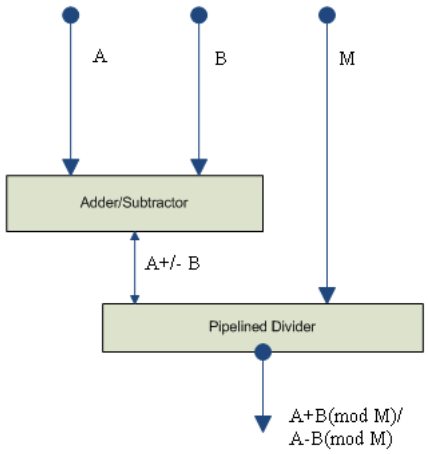


Figure 6.4: Modular Adder/Subtractor

includes calculating $A + B$, and then if the result is larger than modulus M, the modulus M is subtracted from the sum.

As the adder chains in FPGAs are optimized, in-built adders were used. Since, the results of the operation are always larger than M, a comparator is required to compare the sum with the modulus M. However, this may result in multiple comparisons which can slow down the operation. Hence, a pipelined divider was used to perform the operation modulo prime M. As the modular adder/subtractor is integrated in the processing unit, the functional simulation result for the processing unit also tests the modular adder/subtractor. the functional block diagram of the logic unit is shown in Figure 6.4.

6.3 Modular Correction

Modular correction is the process of subtracting the modulus from an input if it is greater than or equal to the modulus. As finite field arithmetic requires that the results of the operations must always lie within the declared field, modular correction is required to make sure that the result is bounded before further calculations. This unit is employed at the end of every operation and is integrated with each modular arithmetic unit. This is done by dividing the results with modulus M.

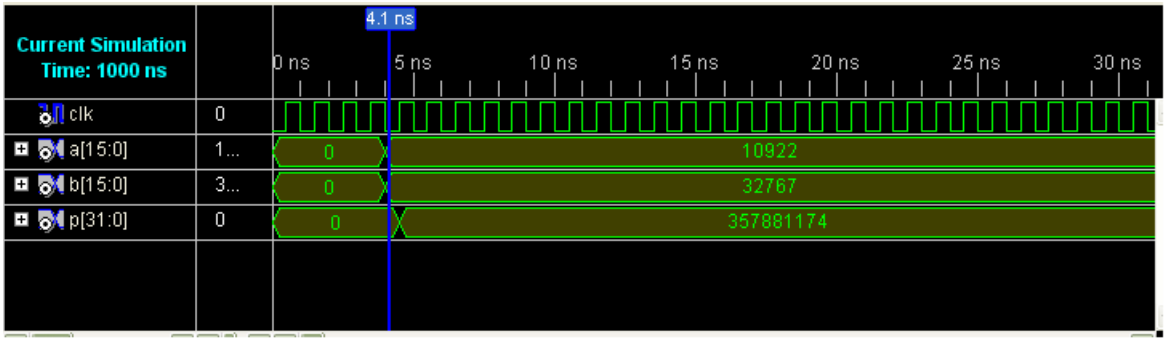


Figure 6.5: Simulation result for Pipelined Multiplier

The figure above shows the simulation result for a 16-bit pipelined multiplier. The multiplier was generated using xilinx core generator. The maximum range of the multiplier is 64-bit. For implementation above 64-bit, pipelined multiplier in[36] was implemented.

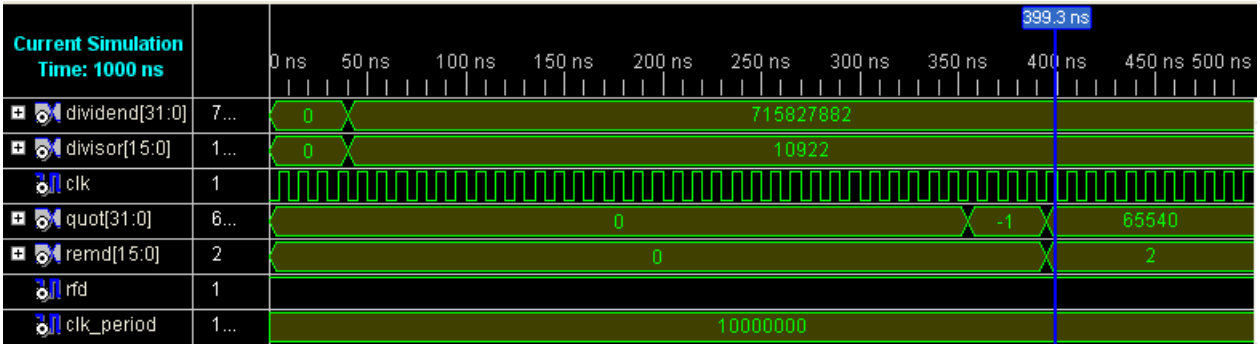


Figure 6.6: Simulation result for Pipelined Divider

The figure above shows the simulation result for a 32-bit pipelined divider. The divider was generated using xilinx core generator.

6.3.1 Pipelined Multiplier

As the resulting architecture was targeted to Xilinx device, core generators were used to achieve maximum results. The multiplier is capable of performing upto 64-bit signed parallel multiplication and based on requirements one can select the pipeline depth.

The functional simulation of the multiplier is shown below in Figure 6.5:

6.3.2 Pipelined Divider

The pipelined divider is a high speed core designed for DSP and other micro-processor environment. The module divides an M-bit variable by an N-bit variable divisor. The divider architecture is fully pipelined and the pipeline depth can be varied as per the area requirements. The results are quotient and remainder and the output is available at M clock cycles. Figure 6.6 shows the functional simulation of the pipelined divider when the dividend is 32-bit and the divider is 16 bit:

Chapter 7

Results

This chapter first explores the synthesis results for the basic design building blocks, and then gives the resulting specifications and simulation results for the entire design. These results are tabulated in table 4.

7.1 Testing

Testing was conducted at many points in the design using different methodologies. Early in the design process, behavioral testing of individual functional units was carried out. Functional wave forms for each functional entity is shown in the Appendix. After testing every module separately, a test bench to test the whole design was developed. Functional simulation for one of the test cases is shown in Figure 7.1- 7.4. Next, a post and route simulation was conducted and the results are shown in Table 7.4. The reports are provided in Appendix. An example for Diffie-Hellman Key Exchange is illustrated in the next section. This example was directly taken from [2].

- Alice and Bob want to exchange a key. In order to do so, the users chose a elliptic curve and the field p. Let the curve be E: $y^2 \equiv x^3 + ax + b \ (mod\ p)$ where $p = 7211$, $a = 1$, $b = 7206$.

- Alice and Bob agree on a common basepoint G on the curve. Let the point be $G(3, 5)$.

- Alice choses a $N_a = 12$ and Bob chooses $N_b = 23$. These numbers are secret. The result of the elliptic scalar multiplication N_aG and N_bG is published.
- For the presented case, we have $N_aG = 12G = (1794, 6375)$(Fig 7.1).
- For $N_b = 23$, $N_bG = 23G = (3861, 1242)$(Fig 7.2)
- Alice then takes N_bG and multiplies by N_a. She gets the key: $N_a(N_bG) = 12(3861, 1242) = (1472, 2098)$.(Fig 7.3)
- Bob takes N_aG and multiplie by N_b to get the key: $N_b(N_aG) = 23(1794, 6375) = (1472, 2098)$.(Fig 7.4)

7.2 Analysis of results

Table 7.1 provides a comparison between the prototype Modular divider implementation of [10] with the presented modular division architecture. The modular divider is implemented on Vertex2Pro-v1000 device and the results are tabulated in table 7.1. Our implementation is faster than the prototype design[10] however, this improvement is achieved at the cost of higher device utilization.

For the overall design, the critical path was found to be along the modular division stage. A summary of the design sizes can be seen in table. Table 7.3 summarizes the FPGA implementation results for n=32,64 and 128-bit elliptic curve ALU. Table 7.4 provides the post and route performance results for the ECC ALU obtained using Xilinx software v10.1. The target device for these results was Xilinx Virtex2 xc2v2000-6 FPGA. The performance results are not easily comparable to previous implementations over GF(p) due to different

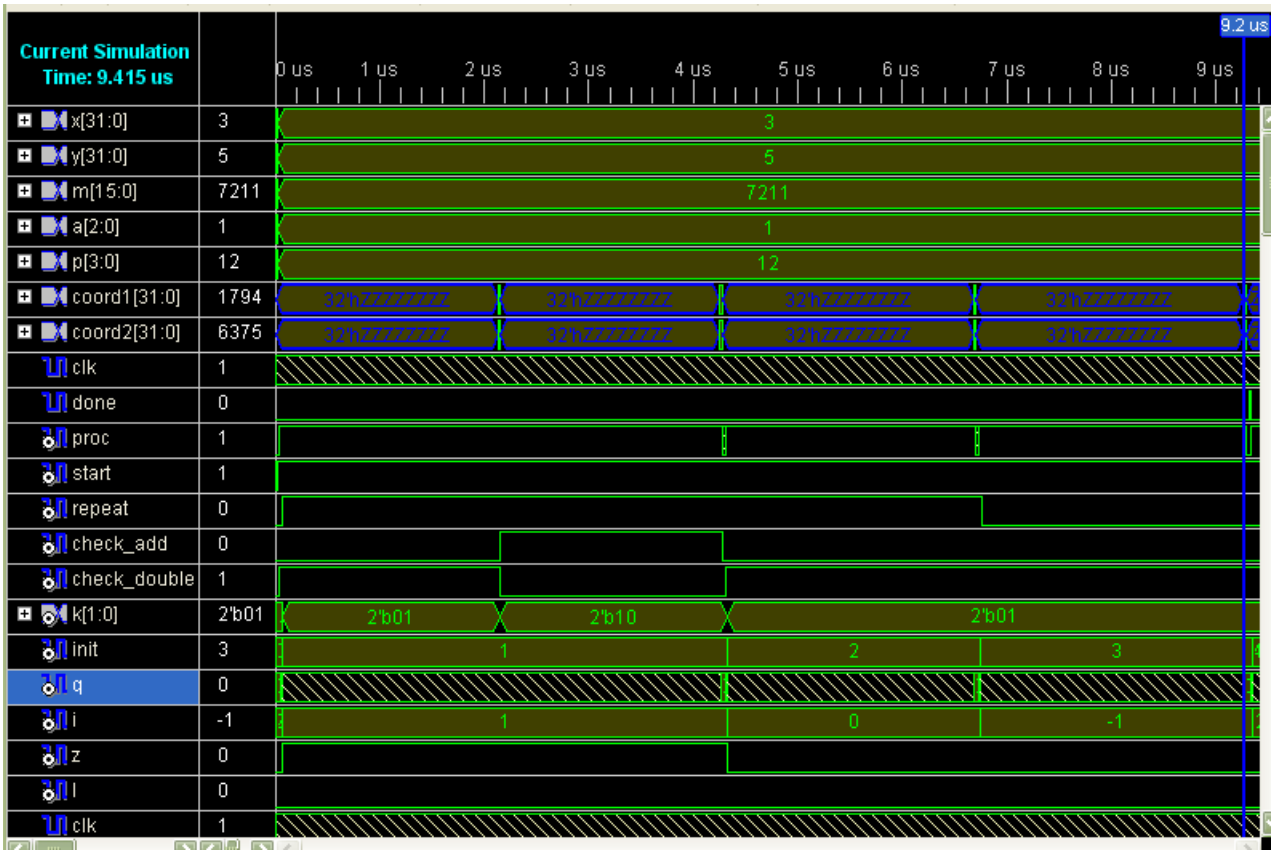


Figure 7.1: Elliptic Curve Scalar multiplication: $k = 12$

The figure above is the simulation result for the main controller performing Elliptic Curve Scalar multiplication for the parameters:: E:$y^2 \equiv x^3 + ax + b(mod\ p)$; $a = 1, b = 7206, p = 7211, x = 3, y = 5,$ $k = 12 : kG = 12(3, 5) = (coord1, coord2 = (1974, 6375)$

N	Area(slices)	Area	Freq(MHz)	Freq.	Area Increase(%)	Speed Increase(%)
	[10]	Proposed	[10]	Proposed		
64	461	1443	83	134.418	45.5%	62%
128	927	3445	75	106.769	36.8%	42.4%
160	3780	4297	77	93.371	13%	21.2%
256	6428	6917	77	87.834	7%	14%

Table 7.1: Performance Comparison Table: Modular Divider

The table above provides a comparison between the design in [10] to the proposed design in this thesis. The comparison is done in terms of area requirements and speed obtained. Target device: FPGA Xilinx Vertex XCV2000e

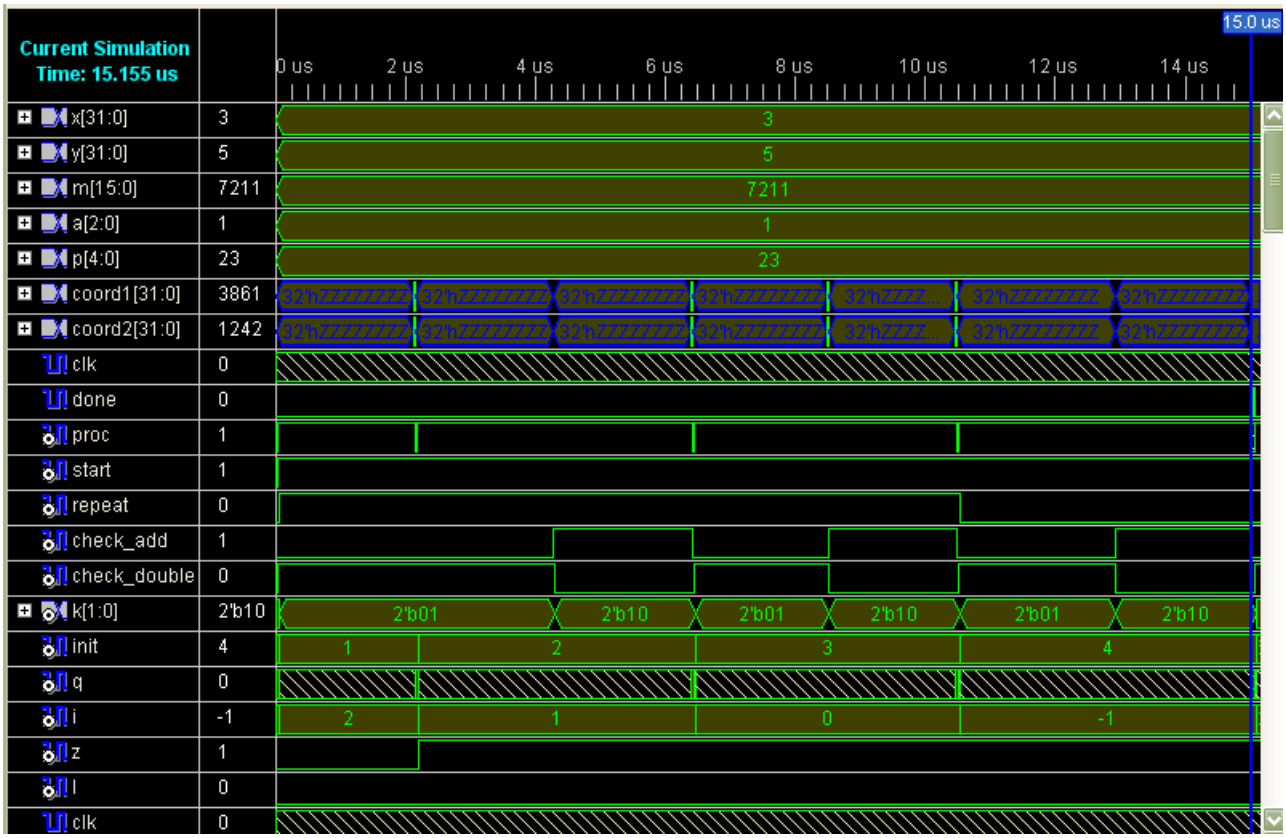


Figure 7.2: Elliptic Curve Scalar multiplication: $k = 23$

The figure above is the simulation result for the main controller performing Elliptic Curve Scalar multiplication for the parameters:: E:$y^2 \equiv x^3 + ax + b(mod\ p)$; $a = 1$, $b = 7206$, $p = 7211$, $x = 3$, $y = 5$, $k = 23$:: $kG = 23(3, 5) = (coord1, coord2) = (3861, 1242)$

N	Area(slices)	Max clock Freq(MHz)	Max combinational path delay(ns)
64	1473	134.418	12.522
128	3445	106.769	15.249
160	4297	93.371	16.838
256	6917	87.834	20.947

Table 7.2: FPGA Implementation Results

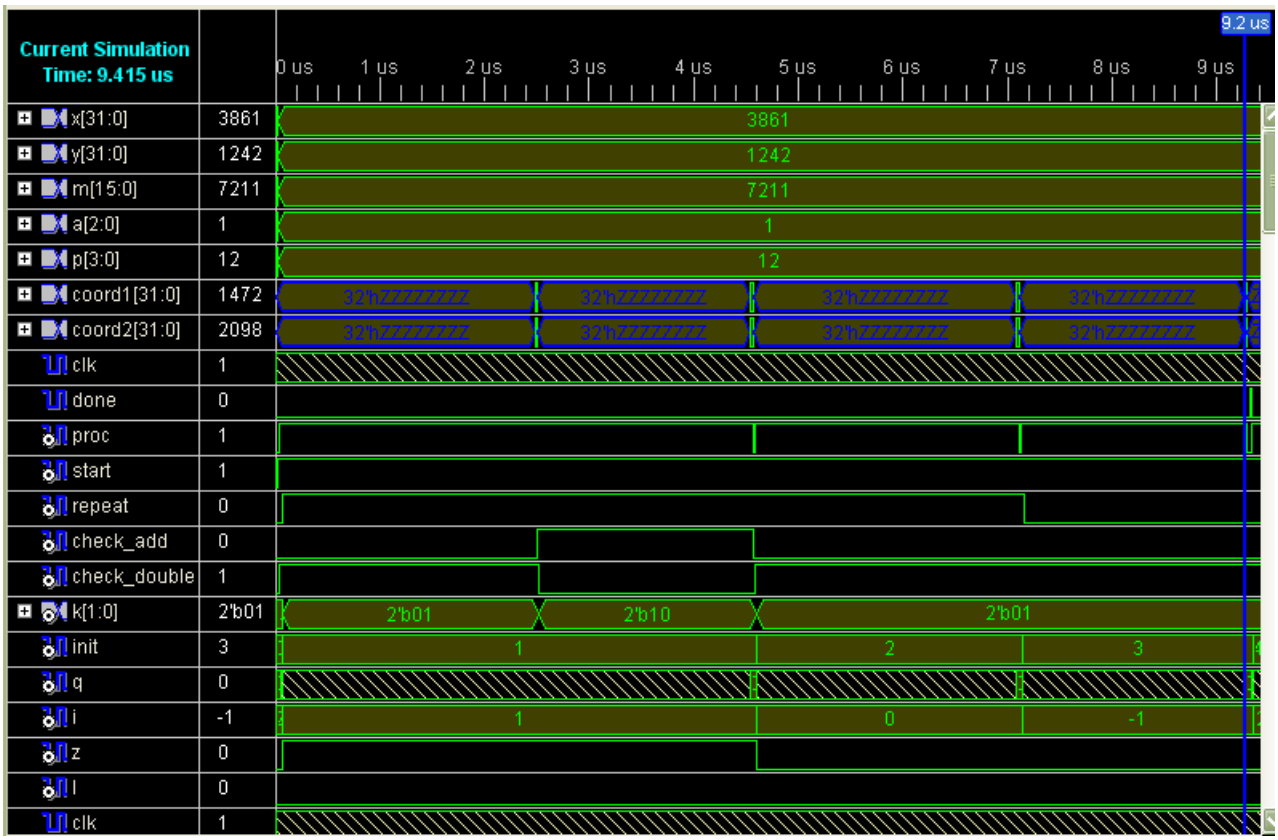


Figure 7.3: Elliptic Curve Scalar multiplication:$k = 12$

The figure above is the simulation result for the main controller performing Elliptic Curve Scalar multiplication for the parameters:: E:$y^2 \equiv x^3 + ax + b(mod\ p)$; $a = 1$, $b = 7206$, $p = 7211$, $x = 3861$, $y = 1242$, $k = 12$:: $kG = 12(3861, 1242) = (coord1, coord2) = (1472, 2098)$

N	Area(slices)	Area	Freq(MHz)	Freq.	Area Increase(%)	Speed Increase(%)
	[9]	Proposed	[9]	Proposed		
64	4850	4940	45	58.862	2%	30%
128	6225	6928	38	58.862	11%	54.9%

Table 7.3: Performance Comparison Table: Elliptic Curve ALU utilizing a modular divider

The table above provides a comparison between the design in [9] to the proposed design of ECC ALU in this thesis. The comparison is done in terms of area requirements and speed obtained. Target device: FPGA Xilinx Vertex X2V6000

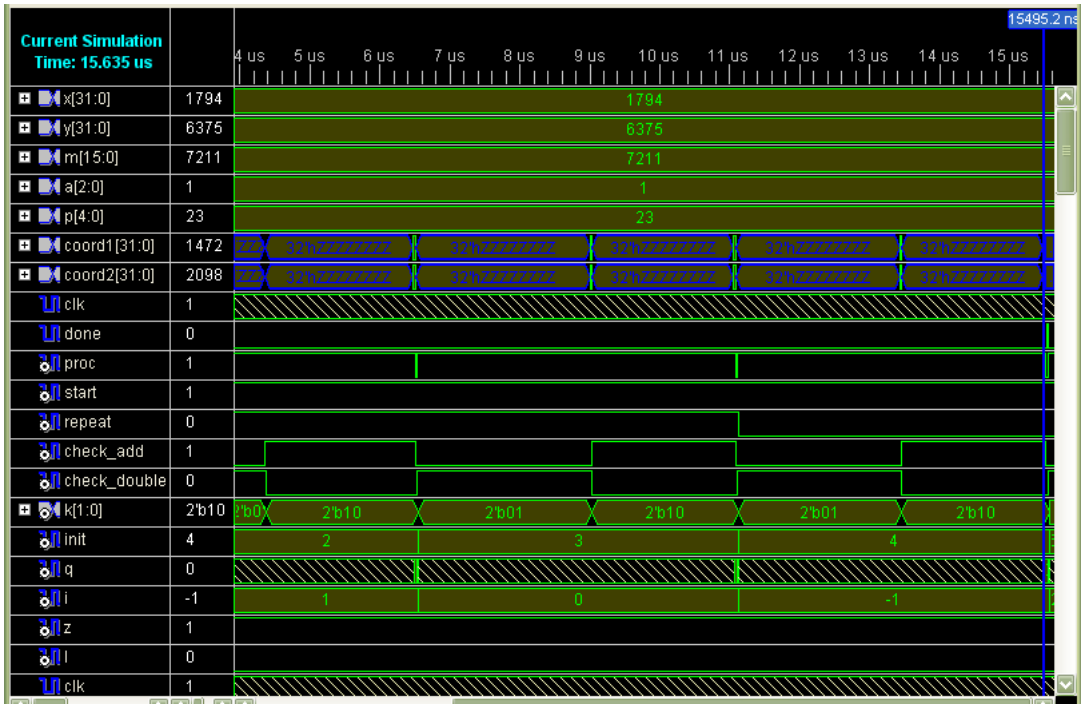


Figure 7.4: Elliptic Curve Scalar multiplication:$k = 23$

The figure above is the simulation result for the main controller performing Elliptic Curve Scalar multiplication for the parameters:: E:$y^2 \equiv x^3 + ax + b(mod\ p)$; $a = 1$, $b = 7206$, $p = 7211$, $x = 1794$, $y = 6375$, $k = 12$:: $kG = 23(1794, 6375) = (coord1, coord2) = (1472, 2098)$

N	Number of slices	Frequency(MHz)
32	3044	76.173
64	4940	58.862
128	6928	58.862

Table 7.4: Implementation Results: Elliptic Curve ALU

Target device: FPGA Xilinx Vertex X2V6000

platforms, however a general comparison is made in order to understand the various approaches and improvements that can be made in this field.

In [21], a versatile GF(p) ALU capable of performing all modular operations including modular inversion was presented. It was observed that 128-bit ALU operated at a maximum frequency of 56.93 MHz and required 1617 number of slices. In our implementation, 128-bit ALU operates at a frequency of 58.862 MHz and required 6928 number of slices. An improvement in the speed was made at the cost of area. Maximum area was utilized by the modular divider.

In comparison to the work in [9] where a modular divider was implemented for ECC ALU in GF(p), the proposed designed in this thesis is faster as they achieved a frequency of 45 MHz for 64-bit implementation while we achieved a frequency of 58.862 MHz. The architecture presented in [9] is the only one employing a modular divider based on Shantz's modular division algorithm[22] for ECC ALU over GF(p). Our design is faster and utilizes a more efficient algorithm for modular division.

The latest work in this field is by McIvor et all[24] where a 256-bit ECC processor architecture over GF(p) was presented capable of performing main prime field arithmetic functions and four different types of modular inversions. Based on the post and route results of 256-bit modular division module presented here a comparison was made with the modular multiplier and modular inversion module of [24]. The critical delay through the design in [24] is 25.34 ns, giving a maximum clock speed of 39.46 MHz. In our implementation, for the overall design, the critical path was found to be along the modular division stage. The modular divider was reported at a frequency of 87 MHz for 256-bit implementation. Since, the performance of the entire design is based on the modular divider unit and based on the results obtained for 64 and 128-bits we can safely assume that a 256-bit implementation of ECC ALU presented in this thesis will be 40 MHz. Our 128-bit implementation utilized 6928 slices equivalent to 1732 CLBs. For the implementation in [24], 15,755 CLBs are required for 256-bit which is about 9 times more than our 128-bit implementation. As the implementation in this thesis employs the variant of extended

Binary GCD algorithm for modular division it is easier to implement it as one logic unit thereby providing flexibility and ease of implementation.

Chapter 8

Conclusions and Future Work

8.1 Summary and Conclusions

It was a great learning experience throughout the development of the thesis. The thesis presents a new architecture for ECC ALU over $GF(p)$ implementing a modular divider. It can be easily concluded that the goal of the thesis was achieved. The work presented here concentrates on developing a high performance ECC ALU based on Extended Binary GCD modular divider, suitable for FPGA implementation. The presented architecture for modular division unit achieved a maximum frequency of 89 Mhz utilizing 1728 CLBs. The presented architecture for ECC ALU was implemented on Virtex2 Pro FPGA and achieved a speed of 59 MHz for 128-bit implementation utilizing 1732 CLBs. Since the hardware design was based on bottom-up method, there were lot of timing issues regarding coordination between various logic units. This was solved by implementing extra control and status signals.

The early chapters in this report highlights the research work that was done to understand the basic arithmetic behind elliptic curves and cryptography. A detailed survey of many ECC implementations in $GF(p)$ and $GF(2^m)$ was reviewed. The review helped in understanding the common practices, latest trends and highlighted some of the major implementations. Further, a study of algorithms and their variant was described that are suitable for modular division implementations. In this regard, two different design implementations for modular division were investigated and the final selected design was

optimized for speed. The selected modular divider is based on Takagi's implementation of Extended Binary GCD algorithm. Once the modular divider was designed, it was tested. Results showed optimized results as compared to the prototype implementation.

The resulting architecture was used to implement an ECC ALU capable of performing elliptic curve scalar multiplication. The overall performance was measured in terms of speed and area. The results were compared with some of the prototype implementations in $GF(p)$.

The architecture is well suited for FPGA implementations and thus, can be reconfigured to implement different curves in $GF(p)$. Since, scalar multiplication is the main operation in the elliptic curve Diffie-Hellman(EC-DH) KAS algorithm, the presented work is suitable for implementation of EC-DH KAS and other cryptographic algorithms in chapter 1.

Based on the research done in this thesis and various test results, we can conclude that using a modular divider to perform scalar multiplication in ECC has it's advantages and disadvantages. The advantage being in terms of using one logic unit for modular division thereby providing ease of implementation and faster operation. However, this is achieved at the cost of area. In comparison to the work in [24] and [9], the architecture presented here is faster and utilizes lesser area and uses a more efficient algorithm for modular division. However, in comparison to the work in [21], the presented architecture is faster but utilizes more area. This is credited to the use of Montgomery multiplication and inversion algorithms used in [21].

8.2 Future Work

The following work can be done in future to make improvements to the design:

1. Further optimizations can be achieved by a pipeline implementation of the modular divider enabling parallel computations.

2. The modular division logic unit can be extended to perform modular multiplication.

3. In chapter 2, there was a brief introduction to Intruder-in-the-middle attack which is a vulnerability in the DH-KAS scheme to an adversary between the two users engaged in a session. The attack can be thwarted by implementing a Authentication scheme, before the two users begin the key exchange process. There are few algorithms that fall under Identification Scheme and Entity Authentication like the Schnorr Identification scheme, the Okamota Identification scheme and others. Most of the algorithms in these scheme require a exponentiation operation in $GF(p)$. Elliptic curves can be used to implement such a system. So, the recommendation for future work in this regard would be to implement a Authenticated Diffie-Hellman KAS.

4. The ECC ALU can be used to implement various other cryptographic applications like elliptic curve encryption or a Elliptic curve based Digital signature.

Bibliography

[1] W. Diffie and M. Hellman. *New directions in cryptography*. Information Theory, IEEE Transactions on, 22(6):644–654, Nov 1976.

[2] W. Trappe and L.C. Washington. *Introduction to Cryptography with Coding Theory*. Prentice-Hall, Inc. Upper Saddle River, NJ, USA, 2005.

[3] R. Schroeppel, H. Orman, and S. OMalley. *Fast Key Exchange with Elliptic Curve Systems*. group, 2:155.

[5] N. Koblitz. *A Course in Number Theory and Cryptography*. Springer, 1994.

[6] H. Eberle, N. Gura, and S. Chang-Shantz. *A cryptographic processor for arbitrary elliptic curves over GF (2^m)*. In Application-Specific Systems, Architectures, and Processors, 2003. Proceedings. IEEE International Conference on, pages 444– 454, 2003.

[7] DJ Guan. *Montgomery Algorithm for Modular Multiplication*. Departement of Computer Science, National Sun Yar-Sen University, Taiwan, August, 25, 2003.

[8] D.E. Knuth. *The art of computer programming, volume 1: fundamental algorithms*. Addison Wesley Longman Publishing Co., Inc. Redwood City, CA, USA, 1997.

[9] A. Daly, W. Marnane, T. Kerins, and E. Popovici. *Fast Modular Division for Application in ECC on Reconfigurable Logic*. Proc. of 13th International Conference on Field Programmable Logic and Application, FPL, 2778:786–795, 2003.

[10] G. M. de Dormale, P. Bulens, and J. Quisquater. *Efficient Modular Division Implementation ECC over GF(p) Affine Coordinates Application. Field-Programmable Logic And Applications*: 14th International Conference, FPL 2004, Antwerp, Belgium, August 30-September 1, 2004: Proceedings, 2004.

[11] M.E. Kaihara and N. Takagi. *A Hardware Algorithm for Modular Multiplication/Division*. IEEE TRANSACTIONS ON COMPUTERS, pages 12–21, 2005.

[12] G. Chen, G. Bai, and H. Chen. *A New Systolic Architecture for Modular Division*. IEEE TRANSACTIONS ON COMPUTERS, pages 282–286, 2007.

[13] Lo'ai Tawalbeh. *A Novel Unified Algorithm and Hardware Architecture for Integrated Modular Division and Multiplication in GF(p) and GF(2^n) Suitable for Public-key Cryptography*. PhD thesis, Oregon State University, 2004.

[14] R.C.C. Cheung, W. Luk, and P.Y.K. Cheung. *Reconfigurable Elliptic Curve Cryptosystems on a Chip*. In Design, Automation, and Test in Europe: Proceedings of the conference on Design, Automation and Test in Europe-, volume 1, pages 24–29, 2005.

[15] M. Bednara, M. Daldrup, J. von zur Gathen, J. Shokrollahi, and J. Teich. *Reconfigurable implementation of elliptic curve crypto algorithms*. Parallel and Distributed Processing Symposium., Proceedings International, IPDPS 2002, Abstracts and CDROM, pages 157–164, 2002.

[15] Y.B.Wang, X.J. Dong, and Z.G. Tian. *FPGA Based Design of Elliptic Curve Cryptography Coprocessor*. In Natural Computation, 2007. ICNC 2007. Third International Conference on, volume 5, 2007.

[16] K. Jarvinen, M. Tommiska, and J. Skytta. *A scalable architecture for elliptic curve point multiplication*. In Field-Programmable Technology, 2004. Proceedings. 2004 IEEE International Conference on, pages 303–306, 2004.

[17] WN Chelton and M. Benaissa. *Fast Elliptic Curve Cryptography on FPGA*. Very Large Scale Integration (VLSI) Systems, IEEE Transactions on, 16(2):198–205, 2008.

[18] G. Orlando and C. Paar. *A Scalable GF (p) Elliptic Curve Processor Architecture for Programmable Hardware*. LECTURE NOTES IN COMPUTER SCIENCE, pages 348–363, 2001.

[19] G. Orlando. *Efficient Elliptic Curve Processor Architectures for Field Programmable Logic*. PhD thesis, WORCESTER POLYTECHNIC INSTITUTE, 2002.

[20] S. Berna Ors, L. Batina, B. Preneel, J. Vandewalle, and BV SafeNet. *Hardware Implementation of an Elliptic Curve Processor over GF(p)*. In Proceedings of Application-Specific Systems, Architectures and Processors (ASAP 03), pages 433–443, 2003.

[21] A. Daly, W. Marnane, T. Kerins, and E. Popovici. *An FPGA implementation of a GF (p) ALU for encryption processors*. Microprocessors and Microsystems, 28(5-6):253–260, 2004.

[22] S.C. Shantz. *From Euclid's GCD to Montgomery Multiplication to the Great Divide*. 2001.

[23] C. McIvor, M. McLoone, and JV McCanny. *FPGA Montgomery modular multiplication architectures suitable for ECCs over GF(p)*. In Circuits and Systems, 2004. ISCAS'04. Proceedings of the 2004 International Symposium on, volume 3, 2004.

[24] C.J. McIvor, M. McLoone, and J.V. McCanny. *Hardware Elliptic Curve Cryptographic Processor Over GF(p)*. Circuits and Systems I: Regular Papers, IEEE Transactions on [Circuits and Systems I: Fundamental Theory and Applications, IEEE Transactions on], 53(9):1946–1957, 2006.

[25] K. Sakiyama, B. Preneel, and I. Verbauwhede. *A Fast Dual-Field Modular Arithmetic Logic Unit and Its Hardware Implementation*. cell (i, j), 500:1.

[26] Wu Shuhua and Zhu Yuefei. *A timing-and-area tradeoff gf(p) elliptic curve processor architecture for fpga*. Communications, Circuits and Systems, 2005. Proceedings. 2005 International Conference on, 2:–1312, May 2005.

[27] A.A.A. Gutub, M.K. Ibrahim, and A. Kayali. *Pipelining GF (P) Elliptic Curve Cryptography Computation*. In Proceedings of the IEEE International Conference on Computer Systems and Applications, 2006.-Volume 00, pages 93–99. IEEE Computer Society Washington, DC, USA, 2006.

[28] A.A.A. Gutub and MK Ibrahim. *High radix parallel architecture for GF (p) elliptic curve processor*. In Acoustics, Speech, and Signal Processing, 2003. Proceedings.(ICASSP'03). 2003 IEEE International Conference on, volume 2, 2003.

[29] G.A. Jones and J.M. Jones. *Elementary Number Theory*. Springer, 1998.

[30] J.H. Silverman. *The Arithmetic of Elliptic Curves*. Springer Verlag, 1986.

[31] S.E. Eldridge and C.D. Walter. *Hardware implementation of montgomery's modular multiplication algorithm*. Computers, IEEE Transactions on, 42(6):693–699, Jun 1993.

[32] P.K. Mishra. *Pipelined Computation of Scalar Multiplication in Elliptic Curve Cryptosystems (Extended Version)*. IEEE TRANSACTIONS ON COMPUTERS, pages 1000–1010, 2006.

[33] K. Sakiyama, B. Preneel, and I. Verbauwhede. *A Fast Dual-Field Modular Arithmetic Logic Unit and Its Hardware Implementation*. cell (i, j), 500:1.

[34] D.R. Stinson. *Cryptography: Theory and Practice*. CRC Press, 2006.

[35] E. Erguner and I. Sogukpinar. *Smart card based remote authentication scheme with strengthened security via ECDH*. In Computer and Information Sciences, 2007. ISCIS 2007. 22nd International International Symposium on, pages 1–6, 2007.

[36] M. Hatamian and G. L. Cash, "*A 70-MHz 8-bit×8-bit parallel pipelined multiplier in 2.5-μm CMOS*,"IEEE J. Solid-State Circuits, vol. SC-21, no. 4, Aug. 1986.

[37] HW Lenstra. *Factoring integers with elliptic curves*.

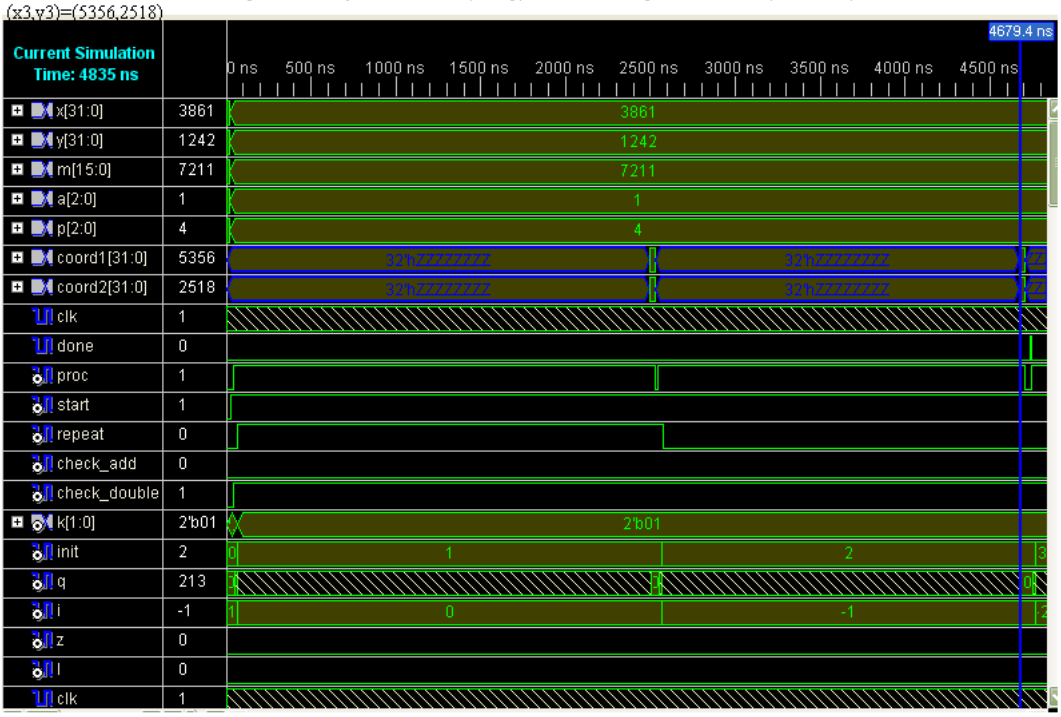
[38] IF Blake, G Seroussi, and NP Smart. *Elliptic Curves in Cryptography*. 1999.

[39] Eric Von Tork. *Elliptic Curves over Finite Fields.* George Mason University, 1992.

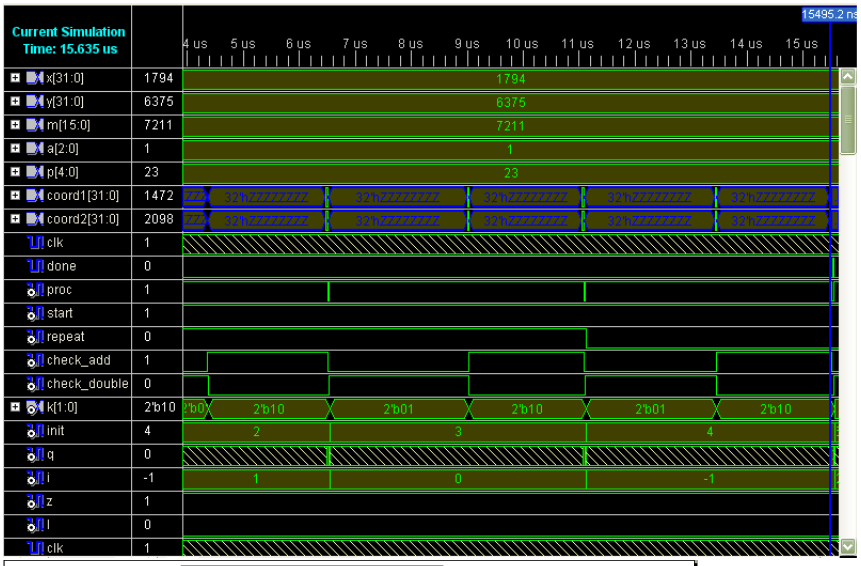
Appendix A

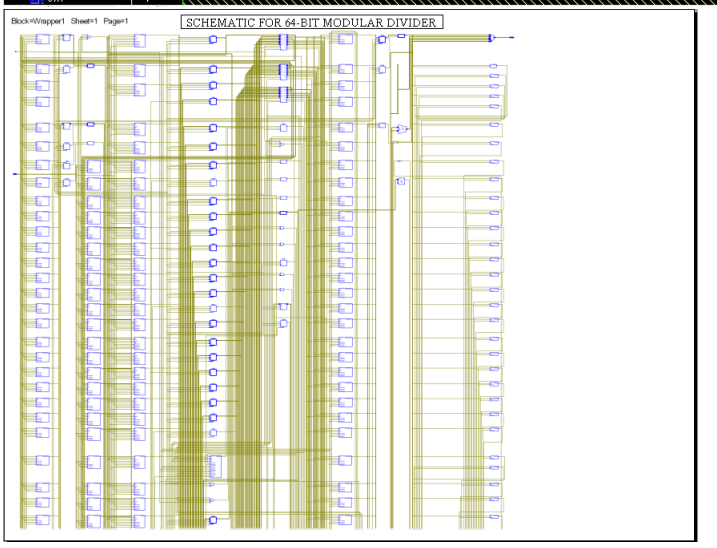
Top module Behavioral simulation and schematic

Simulation Result for 4G on elliptic curve E:y2=x3 + ax + b(mod p); a=1; b=7206; p=7211 and G=(3861,1242)
(x3,y3)=(5356,2518)

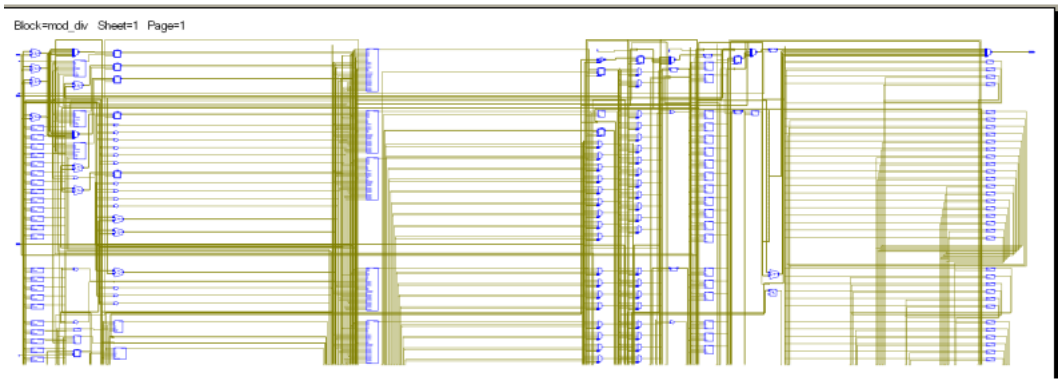


Simulation result for 23G on E:y2=x3+ax+b(mod p); p=7211, a=1, b=7206, G=(1794,6375); x3,y3=1472,2098

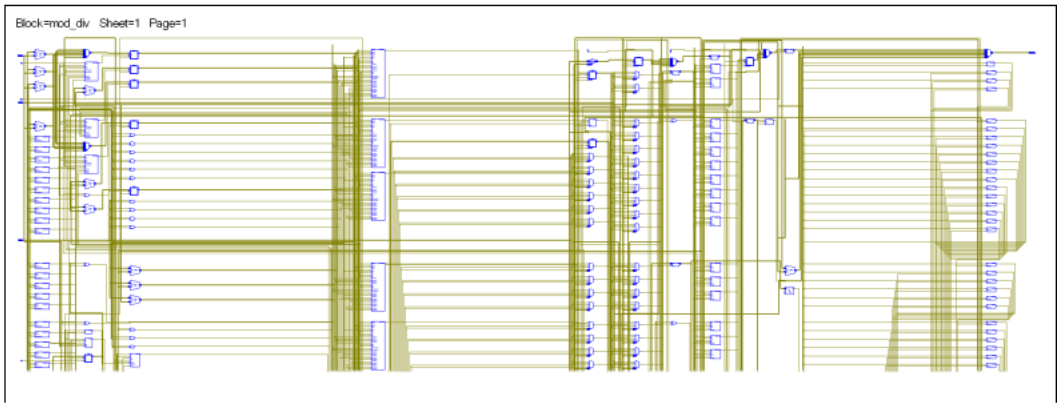




SCHEMATIC FOR 160-BIT MODULAR DIVIDER



SCHEMATIC FOR 128-BIT MODULAR DIVIDER



Appendix B

Synthesis Reports

Appendix C

Post_simulation reports

Appendix D

VHDL Files