On Fast Implementation of Eta-T Pairing on FPGAs

By
Waseem Aslam
2011-NUST-MS-CCS-034

Supervisor
Dr. Nazar Abbas Saqib
Department of Computing

A thesis submitted in partial fulfillment of the requirements for the degree of Masters in Computer and Communication Security (MS CCS)

In
School of Electrical Engineering and Computer Science,
National University of Sciences and Technology (NUST),
Islamabad, Pakistan.

(May 2014)

Approval

It is certified that the contents and form of the thesis entitled "**On Fast Implementation of Eta-T Pairing on FPGAs**" submitted by **Waseem Aslam** have been found satisfactory for the requirement of the degree.

Advisor: **Dr. Nazar Abbas Saqib**

Signature: _____

Date: _____

Committee Member 1: **Dr. Sohail Iqbal**

Signature: _____

Date: _____

Committee Member 2: **Dr. Adnan Khalid Kiani**

Signature: _____

Date: _____

Committee Member 3: **Mr. Arshad Nazir**

Signature: _____

Date: _____

Abstract

The big challenge in an online transaction is the distribution of secret keys in a secure way as it is the core feature of any security system. With advancement in technology and recent security threats there is a need to develop a secure and long term solution for key sharing. Currently, concept of pairing in cryptography is emerging to share the key between the users. Pairing over elliptic curves proves much resilient against most of the attacks. It is efficiently computed in a single round compared to two round Diffie-Hellman key exchange protocol. Eta-T is the most efficient and widely used pairing algorithm based on elliptic curves. In this thesis, our work is based on hardware accelerator for Pairing by using FPGAs platform. We align parts of pairing algorithm to be executed in parallel saving number of clock cycles. We work on efficient computation of finite field arithmetic operators like multiplication to achieve reduced critical paths. We implement Karatsuba multiplier for multiplying two operands in parallel and then its reduction in the same clock cycle. We implement Adder, Multiplier and Cube over F_3^{97}. Our implementation calculates Eta-T pairing in just 36.4 μS by consuming 3597 clock cycles which proves to be 25% less in number of clock cycles than contemporary implementations in FPGAs. We provide a fair comparison of our results with state of the art implementations at the end.

Certificate of Originality

I hereby declare that this submission is my own work and to the best of my knowledge it contains no materials previously published or written by another person, nor material which to a substantial extent has been accepted for the award of any degree or diploma at NUST SEECS or at any other educational institute, except where due acknowledgement has been made in the thesis. Any contribution made to the research by others, with whom I have worked at NUST SEECS or elsewhere, is explicitly acknowledged in the thesis.

I also declare that the intellectual content of this thesis is the product of my own work, except for the assistance from others in the project's design and conception or in style, presentation and linguistics which has been acknowledged.

Author Name: **Waseem Aslam**

Signature: _____

Acknowledgment

This thesis would not have been possible without the continuous support of my supervisor Dr. Nazar Abbas Saqib. Thanks to my supervisor from the very depth of my heart.
Thanks to my committee members, Dr. Sohail Iqbal, Dr. Adnan Khalid Kiani, and Mr. Arshad Nazir, for their guidance and support throughout my research.
The research in this thesis is sponsored through the project titled "Secure-Dial" (Secure Telephone and Fax), No. .20-1688/RD/2010//11 dated June 15, 2012
Finally, special thanks to my parents Muhammad Aslam and Nusrat Parveen as well as my younger brother Nadeem Aslam and numerous friends who endured this long process with me, always offering support and love.

Dedication

To my parents Mr. and Mrs. Muhammad Aslam

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

In 1976, Diffie and Hellman[1] revolutionized the world with the latest key sharing algorithm which is known as Diffie Hellman key exchange method. This is the first time in which public and private key was used to share the key between the users. The complexity of this algorithm lies in the Diffie Hellman problem. Despite this algorithm a well known algorithm RSA[2] exist in which public and private key pair is used and the complexity of this algorithm lies in factoring large prime number which is a difficult problem.

Elliptic curves was first introduced in cryptography by Koblitz [3] and Miller [4] in 1987. They introduced the associated group to solve the discrete logarithmic problem. The group used to solve the problem of discrete logarithmic is multiplicative groups and even there is no any single algorithm to solve the discrete logarithmic problem by polynomials. Up till now, all the algorithms to solve the discrete logarithmic problem are exponential. This is very hard on the elliptic curves even if we take the key length very small and check the complexity with other systems. Comparing the complexity of elliptic curve cryptosystems with the other systems like RSA or any other, elliptic curve cryptosystems are much better than the other systems in terms of security.

The tool used to prove the properties of the elliptic curves are the Weil [11] and Tate [12] pairing. But first time breakthrough came when in 1993, Menezes, Okamoto and Vanstone (MOV) showed that to convert the discrete logarithmic to discrete logarithmic problem by using Weil [11] pairing in some other extension field which is derived from the original base field. But later on, Frey and Ruck [5] had done the same problem to convert it to discrete logarithmic problem by using the Tate [12] pairing.

In 2000, Sakai et al.[6] and Joux [7] proposed two independent cryptosystems by using pairing over elliptic curves. Diffie Hellman key exchange protocol

was used to share the key between two users but Joux [7] proposed an idea to share the key between three users instead of two and Sakai et al. [6] proposed the use of pairing in identity based cryptosystems. The notation used to identify the identity based cryptosystems was give by Shamir [8] in 1984. It consists of public and private key cryptosystems. Public key is bounded with the entity and the person having the private key corresponding to Public key can only decrypt the message.

In 2001, Boneh and Franklin [9] and Cocks [10] proposed two algorithms separately for encryption based upon identity. In both the algorithms, one is based upon Weil [11] pairing and the other is on quadratic properties. Therefore, pairing is getting more attraction and it becomes the most popular area for key sharing. Once pairing based cryptosystems had developed and now the direction of research turned around to optimize the pairing based cryptosystems as well as key sharing protocol by using the elliptic curves.

1.1 Elliptic Curves

1.1.1 General Equation

The general equation for the elliptic curve is given as

$$y^2 + axy + by = x^3 + cx^2 + dx + e \tag{1.1}$$

This equation shows that a,b,c,d and e are the coefficients and are the real numbers as well as x,y are also the chosen from the real numbers. If we define $\mathbf{K}$ be the field and these values from the field $\mathbf{K}$. Where (x, y) be the real numbers and are the solutions of the above elliptic curves and x,$y \in \mathbf{K}^2$. There is also a point at Θ. The coordinates of the point can be written as (0,1,0) which is in the projective coordinates. But for simplicity we can say that the coordinates of the point which has its x component zero and the y component is 1 are the point at Θ.

1.1.2 Weierstrass Equation

Equation 1.1 is the general form of the elliptic curves which can be reduced to the Weierstrass equation and can be written as,

$$y^2 = x^3 + ax + b \tag{1.2}$$

The degree of polynomial in equation 1.2 is 3, which is the highest power of the any variable and a,b are the values which we have to choose for the elliptic curve which decides the characteristics of the curve.

$$y = \sqrt{x^3 + ax + b} \tag{1.3}$$

Equation 1.3 shows that the curve is symmetric about y=0 for the given values of a and b. The main task is to choose the appropriate values of a and b such that there is no singular points on the curve. Singular point is that the curve does not intersect at any point if we extend it to Θ. This is only possible if this mathematical equation holds true,

$$4a^3 + 27b^2 \neq 0 \tag{1.4}$$

Let us suppose that if we take a=1 and b=0 in equation 1.2 the the equation becomes,

$$y^2 = x^3 + x \tag{1.5}$$

The points which satisfy the above equation are the points on the elliptic curve. The only condition to find the points on the curve is that the discriminant should not be zero. As b is zero in our case so the value of discriminant is $(4).(1)^3 = 4 \neq 0$. As we are working in characteristics three so we can choose the value of a and b between 0, 1 and 2. Equation 1.2 is derived from the general equation as discussed in the equation 1.1

1.2 Groups

Set of points on an elliptic curve not intersecting each other hold some of the properties which leads us to some conclusion and the properties which the points on curve hold are listed as.

- Suppose $P \in E(K)$and $P+ \Theta=P$ and $\Theta+P=P$. So we can say Θ is an additive identity which does not effect on the point P. If $P=\Theta$ then $-P=\Theta$.

- If $p(x, y)$ and -$P = (x, y)$ which is equal to -$P = (x, -\mathrm{y})$, If we add P+(-P)=Θ. Then we can say that -P is the inverse of P and Θ is the identity which is obtained after adding a point to its additive inverse.

- If $P(x_1,\ y_1)$ and $Q(x_2,\ y_2)$ where $x_1 \neq\ x_2$ i.e. P and Q are different points on the elliptic curve and the addition of these points P+Q=-R results in -R which is the negative of the sum of the two points. In other words -R is the reflection of R and R is the point of intersection joining the line P and Q with E. Addition law is described in figure 1.1.

- If $P(\mathrm{x}_1,\ y_1)$ and we have to add P+P=-R, this is called the point doubling and -R is the negative of the sum of the two points. In other words -R is the reflection of R in the x-axis and R is the point of intersection joining the line P and P with E.

1.3 Order of E($\mathbf{F}_q$)

1.3.1 Hasse Theorem

It is possible to generate points on an elliptic curves starting from the first point because by adding points the result lies on the curve until we reach the point Θ which is known as the point at infinity and this point is considered as identity point. Total number of points lying on the curve is called the order of the curve. First roughly approach to count the number of points on the curve is less than $2q + 1$. If we substitute the value of x in the equation 1.3 then we have two values of y, one is positive and the other is negative. So $2q$ points are obtained and one point lies at infinity is also included in these points then the total points are $2q + 1$ which is not the exact measure of the number of points lying on the curve. The exact formula to find the number of points on the elliptic curve is given by Hasse [17] and the points lies between this limit.

$$q + 1 + 2\sqrt{q} \leq |E(F_q)| \leq q + 1 + 2\sqrt{q} \tag{1.6}$$

The proof of this formula to find the limit of the number of points can be found [15][16] here.

1.3.2 Determining the Order

For some of the curves to find the order is very simple. If we can write t=$|E(F_q)|$-q-1 and t$\equiv$ 0 (mod p) then $|E(F_q)| \equiv 0$(mod p). The converse is also true, so the order of the curve is found very easily by using schoof's algorithm if we can write it as t= $|E(F_q)|$-q-1. Further we use Chinese Remainder theorem for further calculation of t. In our scenario we have to find the order of E_q=$E_p k$. So the order of E_p can be calculated very easily by using the Schoof's algorithm if p is small i.e. p= 2 or p= 3. Furthermore, the calculation of $E_p k$ is calculated by using the following theorem, where k is any positive integer.

Theorem 1:

Suppose, $|E(F_q)|$ = p+1-t and α, β be the roots of a trinomial given as $T^2 - xT + P$. Therefore we can write it as,

$$T^2 - xT + P = (T - \alpha)(T - \beta) for all k \geq 1 \tag{1.7}$$

Then we have,

$$|E(F_q)| = p + 1 + \alpha^k + \beta^k \tag{1.8}$$

For proof of this theorem we refer to the following paper[18][19].

1.4 Order of Point

Order of point is different from order of curve. Total number of points lying on the curve and the point at infinity is called the order of the curve but order of point is somewhat related to scalar multiplication. Suppose we have a scalar quantity n and a point P lying on the curve having coordinates $P(x, y)$. To get the order of the point we have to add the point P with itself to n times so that we get the result nP=Θ. The n is called the order of the point.

1.4.1 Torsion Points

Once we have calculated the order of curve, next task is to find the order of the point as we already discussed above. Suppose we have a scalar quantity m and a point P lying on the curve having coordinates $P(x, y)$. To get the order of the point we have to add the point P with itself to m times so that we get the result mP=Θ. Points have both finite and infinite order. The

points which have infinite order are of no use. We only focus on the points which have the finite order. The points which have finite order i.e. the points which reach to $mP{=}\Theta$. Then m is called the torsion of the point. All the points on the curve has not the same order. The points having same order are grouped in the same group which is also called the torsion group. If a curve is defined over the finite field F_q then all the points are the torsion points and the value of torsion is different because all the points do not have the same order and the order of the point divides the order of the curve.

Definition 1:

Suppose r is any positive integer then, $E[r]{=}\mathrm{E}{\in}\ (F_q)|\mathrm{rP}{=}\ \Theta$

Then r is called the torsion of the curve. There is a set of the points which are algebraically closed is called the algebraic closure. In this set all the points are not of the order r. If a point P is of order n then i.e. $nP{=}\Theta$ then n is called the smallest integer which satisfies the condition $nP{=}\Theta$ and the point P does not belong to the set which have the order n i.e. $E[n]$ but also a member of $E[r]$ because we know that n divides r. We suppose that if k is any positive integer and then $knP{=}\Theta$ means that the torsion group not only contains the points which have order n but it consists of the points which has the order multiple of n.

1.4.2 Supersingular Curve

E is said to be supersingular if it has not a single point of order p, where P is the characteristics of the curve $E[p]{=}\{\Theta\}$

- If r is the power of P then $E[r]{=}\{\Theta\}$ because E is supersingular or $E[r]$ is isomorphic $\mathbf{Z}_r$
- If r is co prime to p then $E[r]$ is isomorphic i.e. $\mathrm{E}_r\ {\times}E_r$. This simply means that $E[r]$ has r^2 elements but not a single element has of the order r^2.

  As p and r are coprime and these groups of points are of great interest to calculate the pairing for key sharing and the identity point Θ is of order 1.

1.4.3 MOV degree

We now introducing a new term MOV related to r which is very useful for the security parameter.

Definition 2:

We suppose that E is an elliptic curve and r is any positive integer then we can say that $E[r]= E(\mathrm{F}_q^k)[r]$ where k is a small integer then k is called the MOV degree or the embedding degree. In other words k is the smallest extension of the field which have all the points of the order r. Sometimes it is also called security parameter. MOV stands for Menezes Okmoto Vanstone[24]. Finding the MOV degree is not an easy task. To achieve this task we have to understand the following theorem.

Theorem 2:

Let, E is an elliptic curve and r is any positive integer such that r divides $|E(F_q)|$ but it does not divide $q-1$. Then we can say that $E(F_q)$ contains r^2 points of order r if and only if q^k-1 is divided by r. If our hypothesis verifies that r divides the order of the curve and it does not divide the $q-1$ then we can say that it is the smallest positive integer that divides q^k-1.

Theorem 3:

Let, E is a Supersingular elliptic curve and r is a prime divisor of $|E(F_q)|$ then the embedding degree is less than or equal to 6.

Supersingular curves are of great interest and it has very small MOV [24] degree i.e. it has all the points which have $E[r]$. Once we found that the embedding degree is less than or equal to 6 then it is very easy to find the points of $E[r]$ and we have to work maximum in the field F_6.

1.4.4 Point Addition and Point Doubling

Point addition is the main operation on elliptic curves. Suppose P_1 and P_2 be the two points on the curve and O be the additive identity. Identity is the term which is added to any point and there is no effect on the result. So if we add it to point P_1 i.e. P_1+O=P_1 and P_2+O=P_2 (i.e. no effect on the point). If we add P_1 with P_2 which intersect at -P_3. After taking the negative of the point -P_3 which is P_4 and our desired result of the addition of the points P_1 and P_2. This can be shown in the fig. 1.1.

The coordinates of points are given as, $P_1(x_1, y_1)$, $P_2(x_2, y_2)$ and the coordi-

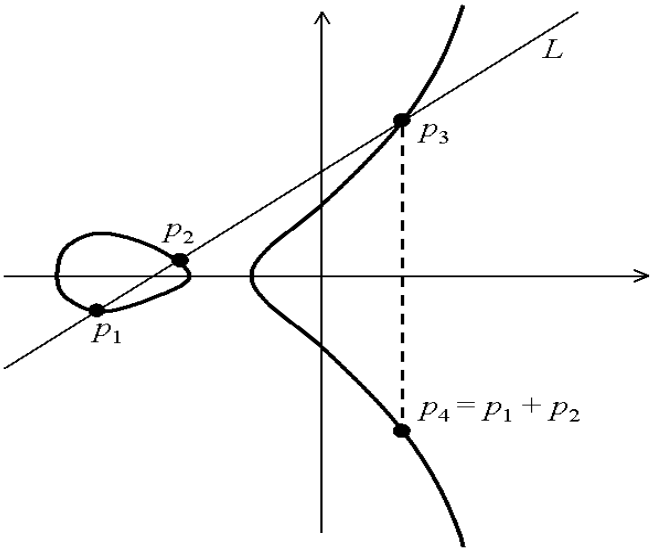

Figure 1.1: Point Addition

nates of point $P_4(x_4, y_4)$ can be found by this formula. First we find the slop of the points as,

$$\Delta = (y_2 - y_1/x_2 - x_1) \tag{1.9}$$

and the coordinates of the point p_4 is calculated as,

$$x_4 = \Delta^2 - x_1 - x2 \tag{1.10}$$

$$y_4 = -y_1 + \Delta(x_1 - x_4) \tag{1.11}$$

This is the formula if both the points are different. If the points are same then the formula for addition is different and is calculated as,

$$x_4 = (3x_1^2 + a/2y_1) - 2x_1 \tag{1.12}$$

$$y_4 = (3x_1^2 + a/2y_1)(x_1 - x_4) - y_1 \tag{1.13}$$

1.4.5 Scalar Multiplication

Scalar multiplication is most likely as that of point addition. Suppose n is any scalar number and we have to multiply it with a point $P(x_1, x_2)$. In

elliptic curves simple multiplication is not possible and we have to add this point n times to get the result nP.

$$nP = \underbrace{1 + 2 + ... + n}\, P \tag{1.14}$$

To achieve the result of scalar multiplication we have to start adding the point P with itself until we reach our desired value which is nP. For scalar multiplication we need point addition and point doubling which we have already discussed in the section of point addition. For simplicity we use double and add algorithm [14] so that we need to do less computation rather than the simple point addition formula.

In this research we focus on Eta-T pairing algorithm and before Eta-T [13] pairing many research publications and books were on Weil [11] and Tate [12] pairing. These pairings can be calculated by using elliptic curves and bilinearity property. So there is a need to be a protocol that lies in the middle of it and further form this we can calculate the Tate [12] and Weil [11] pairing very efficiently. So the answer of this question is Eta-T pairing. The algorithm to calculate the Eta-T pairing [13] is already give in some of the research papers and literature survey.

The purpose of this research is to optimize the Eta-T pairing [13] so that we can achieve much improvement in the calculation time of Tate [12] and Weil[11] pairing. Before our contribution a lot of modification to this has happened but still there are some weaknesses which still required further improvement. In this research we contribute to the Eta-T pairing[13] algorithm by changing at the implementation level. The algorithm which, I have implemented calculates pairing by serial computation and it consumes a lot of clock cycles. To optimize this algorithm we have to go through a lot of study in which firstly we studied elliptic curves and their properties how it behaves when it is used in pairing and what is the complexity of the pairing algorithm when elliptic curves are used. Then we have studied what are the points on the curve and Torsion points and how it behaves when it is used in pairing. Then we came to know how points can be represented in the form of divisors and functions which is the further requirement of the Tate pairing. On elliptic curves scalar multiplication is not simple but it is the addition of points to calculate the multiplication which is the main strength of this pairing algorithm. By getting all this information we have to check the complexity of this algorithm which we came to know that discrete logarithmic problem is very hard on the elliptic curves which makes it less vulnerable to the attacks. In the end we have a comparison between the one round key exchange protocol and two round key exchange protocol. Diffie Hellman [1]

is two round key exchange protocol which is used to share the key between the three users in two rounds. In first round partial key is shared between the users and in the second round full key is shared between the users so that they agreed upon a shared key which is further used to encrypt the messages.

Compared with two rounds key exchange protocol pairing is introduced to share the key between the users in a single round. In one round key exchange users broadcast their message to other users. Each user get the response from the other two users and calculate the pairing of two values which he got and take the power with his private key value. The result obtained is the shared key between the users. This step is performed by each user and all three users have the same value and this is performed in a single round. Therefore, we proffered one round key sharing protocol rather than two round key sharing protocol which is Diffie Hellman [1] key exchange protocol. Getting advantages of one round key exchange protocol and further optimization to this protocol leads to overall optimization of security system.

1.5 Motivation

Security is the key feature of any security system and very less work has been done on this field so far, especially in countries like Pakistan. Now a days online key sharing is a big problem. If key is compromised then the whole system is compromised and no value of the security system. Finally, we also want to shift key sharing protocol from two rounds to a single round protocol by using pairing so that key is more secure and efficiently computed. Two round key sharing protocol provides less security as compared to one round protocol which uses pairing over elliptic curves.

1.6 Problem Statement

"Working on pairing algorithm for low hardware area and high speed to ease the key sharing protocol for advanced security application"

1.7 Thesis Contribution

Before our research pairing is calculated in multiple research publications. Among those we have implemented three to four algorithms. We critically analyze those algorithms and found that algorithms uses cube roots, more number of iterations, more hardware area, large number of clock cycles and

serial computation in the extension field. Based on this weakness in the key sharing protocol we have implemented an algorithm which uses less number of iteration and no cube root but it uses serial computation in the extension field and we have calculated it in parallel. After successful implementation and comparison with previous results shows that we saved 25% less clock cycles which results in the overall optimization of key sharing protocol as well as further calculation Tate pairing.

1.8 Thesis Organization

The organization of thesis is as in first chapter we discussed introduction in which motivation, problem statement and thesis contribution is described. In chapter 2 literature survey is described in which we have critically analyzed the pairing algorithm. In chapter 3 hardware implementation is discussed in detail and in chapter 4 results and comparison is discussed and in chapter 5 conclusion and in the last appendix.

Chapter 2

Literature Survey

2.1 Bilinear Pairing

As we have discussed all the details about elliptic curves and all its related contents which is useful for further use. Now we move towards our main task which is known as bilinear pairing. In this section we will also discuss rational function and divisor.

2.1.1 Rational function

This is very technical topic and in mathematics rational functions are those in which we can write in the form of p/q.

Theorem 4:

Suppose that $E(K)$ is an elliptic curve with equation $f(x, y)$=0. Then it can be proved that a polynomial with the zero function on the curve $E(K)$ if and only if it is a multiple of f. There is an equivalence relationship and we can say that $E[K]= \ K[x, y]/< f >$, then we can say that $E[K]$ is rational function of E.

Theorem 5:

Suppose E is an elliptic curve and f is any rational function. Then there is zeros and poles on the function and furthermore, the number of zeros are equal to the number of poles. If the number of poles as well as number of zeros is zero then it is called the constant function. The function we will use has as many zeros as that of number of poles. But classical function over real field is not possible and rational function over the elliptic curves are defined

over algebraically closed field $E(\bar{K})$ with which a point at infinity is included Θ which is the identity point.

2.1.2 Divisor

Divisor is the last tool which we have to understand further to go to the final pairing. First we give the formal definition then we go for example to describe it. Let E be an elliptic curve which is defined over a field $\bar{K} i.e.$E $(\bar{K})$. For point $P \in E(\bar{K})$ we describe the symbol for divisor $[p]$.

Definition 2:

The divisor D on E is the linear combinations of the points as given below,

$$D = \sum_i a_j p[j], with a_j \in \mathbf{Z} \tag{2.1}$$

The group of divisors is denoted by $DivE$ and the symbol for the divisor is $[p]$. Now we discuss the two main functions in the Divisor.

Degree of the divisor is from the function we can write it as.

$$\sum_j a_j p[j] = \sum_j a_j \tag{2.2}$$

The sum of the points is described as,

$$\sum_j a_j p[j] = \sum_j a_j p[j] \tag{2.3}$$

Subgroup of divisor zero is of much interest for pairing.

2.1.3 Pairing

We have discussed all the tools needed to describe pairing. We now move towards that how pairings are constructed and which properties are they hold. We suppose that G and G' are the two additive groups and H is another group which is multiplicatively defined. Additive group is defined by the sign "+" and the multiplicative group denoted by "." We denote the

pairing by $\hat{e}$ Such that $G \times G' \rightarrow H$. The most important property in calculating pairing is bilinearity property which is defined as:

$$\forall S_1, S_2 \in G \; and \; U_1, U_2 \in G', \hat{e}(S_1 + S_2, U_1) = \hat{e}(S_1, U_1).\ \hat{e}(S_2, U_1) \tag{2.4}$$

$$\hat{e}(S_1, U_1 + U_2) = \hat{e}(S_1, U_1).\ \hat{e}(S_1, U_2) \tag{2.5}$$

In modern cryptography we use pairing on two groups e.g G and H and the pairing is defined between these two groups and it can be written as
G $\times G \rightarrow H$.

Suppose X, Y, Z $\in G$ then,

$$\hat{e}(X + Y, Z) = \hat{e}(X, Z).\ \hat{e}(Y, Z) \tag{2.6}$$

$$\hat{e}(X, Y + Z) = \hat{e}(X, Y).\ \hat{e}(X, Z) \tag{2.7}$$

Now the second property by using the straight algebra and the bilinearity property we can say that if a, b are the positive integers and P is any point which is $P \in \mathbf{G}$ as,

$$\hat{e}(aP, bP) = \hat{e}(abP, P) = \hat{e}(P, abP) = \hat{e}(P, P)^{ab} \tag{2.8}$$

The last property we discussed is very useful for the calculation of identity based protocols. Now we have discussed all the information and the pairing, further task is to construct Eta-T pairing which is also useful for the calculation of Tate pairing.

2.2 Miller's Algorithm

The concept of pairing is first time given by Miller[20]. We suppose that P $\in$E $(\mathrm{F}_q)[l]$ and Q $\in$E$(\mathrm{F}_q k)[l]$ and the reduced Tate[12] pairing equation is given by

$$e_l(P, Q) = f_{l,p}(D_q)^{((q^k-1)/l)} \tag{2.9}$$

Where $f_{l,p}$ is a rational function defined over the elliptic curve E and its divisor is written as $l(P) - l(\Theta)$ and D_q is a divisor equivalent to $(Q) - (\Theta)$ and

the last part consists of $(q^k - 1/l)$ is called the final exponentiation. Tate[12] pairing satisfies the property of non degeneracy and bilinearity which are given as.

- **Bilinearity:** Suppose a is an integer, then

$$(aP, Q) = \hat{e}(P, aQ) = \hat{e}(P, Q)^a \tag{2.10}$$

- **Non Degeneracy :** if there exists e_l(P,Q) = 1 for all Q$\in$ $E(q_k)[l]$ then there should be P=Θ

The above discussed properties and algorithm formula is for the reduced Tate pairing[12]which was first discussed by Miller[20] in the form of Weil pairing[20]. Then this algorithm goes through a lot of modifications[21][22] [23]to achieve significantly better results. Furthermore, Barreto et al.[21] exploited the distortion map to enhance the Miller's algorithm[20]. Final comparison between the modified Tate pairing [12] and Eta-T pairing is given as.

$$(\eta_T(P, Q)^W)^{3T2} = \hat{e}(P, Q)^Z \tag{2.11}$$

2.3 FPGA and ASIC implementations of the Eta-T pairing in characteristic three

First paper added in this research which is closely related to the our work is "FPGA and ASIC implementations of the η_T pairing in characteristic three". In this paper two processors were used to calculate the reduced η_T [13] pairing. First processor is used to calculate the pairing of two points and second processor is used for the final exponential of the pairing value. In this paper they used the η_T pairing [13] approach to calculate the modified Tate pairing [12] and save iterations from 3m to just m. By using this approach both Miller and final exponential algorithms are modified.

In this paper algorithm used to calculate the η_T[13] pairing did not require any cube root and need only two points P and Q in the base field and it returns the η_T pairing of these points. Furthermore, it uses Fermat's little theorem to calculate the final exponential of this reduced η_T pairing value. In final exponential algorithm this uses P Q in the extension field F_3^{6m} and cube operator is required just for cubing.

Algorithm used to calculate the multiplication in the field F_3^m and F_3^{6m} is Most Significant Element(MSE) [25] first. This algorithm is already discussed

above. The remaining operator used in this paper are same as we have used in our research which is cubing operator and lastly it calculates the final exponential so it also uses the inversion over the base field. They also used sharing of memory up to three multipliers so that accessing to the memory is minimized.

2.4 An Algorithm for the Eta-T pairing in Characteristics Three and its Hardware implementation

The second paper which is the most relevant to our implementation is "An Algorithm for the Eta-T pairing in Characteristics Three and its Hardware implementation" [13]. Two algorithms were given in this paper, first one uses the cube root and the second one is cube root free. In this paper author proposed the modified η_T pairing and the used the multiplier in the base field F_3^m and multiplication algorithm is (MSE)[13] which is further used to calculate the multiplication in the field F_3^{6m}.

As in this paper they used the serial multiplication over the extension field F_3^{6m} and they saved the multipliers which also result in large number of clock cycles as well as large calculation time. We exploited this vulnerability in our research and we use Karatsuba multiplier and we calculate all this computation in parallel, which result in we save upto 25% clock cycles and the calculation time. The down side of this parallelism is that we use large number of multipliers and area as compared to the original algorithm proposed by the author.

2.5 Multiplication over F_p^m on FPGA

This paper is kind of survey, in this they proposed the modified η_T pairing [30] and it explore the trade off between resources and calculation time by using large values of P, m and $f(x)$. Plus point of this modified pairing algorithm is that no restriction on the elliptic curve and we can use any value of P and m. Weak point of this paper is that it uses very complex state machine and it uses more registers. For calculation of the final η_T pairing algorithm in this paper they proposed no modification to the final exponential algorithm.

Last paper most relevant to this research is "Fast Architectures for the η_T Pairing over Small-Characteristic Supersingular Elliptic Curves" [31] and

they used the parallel multiplier over the field in characteristics two and three. They used parallel pipeline Karatsuba multiplier for calculating Millers algorithm [20] and used sparse multiplication to calculate the multiplication over F_3^{6m}. The main task in this multiplier is to keep busy during the whole multiplication.

2.6 Research Methodology

Now, come to the main point how we go and prove our research. There are two types to prove your research one is called inductive research and the other is deductive. Deductive research is called top-down approach and inductive research is called bottom-up approach. In our research we have in our mind to calculate the η_T pairing which is the top of our research and we used different modules to prove our result, which are working at the bottom of the pairing.

In the first phase we did the literature survey about this field. This is very important to get information about this field and without this we will not know about the problem in this area. Once we have realized the problem in this domain which helps us to compile the problem statement as discussed in the chapter Introduction.

In second phase of the research is to implement the existing algorithm used for pairing. Only one algorithm cannot tell us to optimize it so that we have to implement two to three algorithms. Once we have chosen the algorithm which needed less clock cycles, we need to optimize that one. For optimization we use karatsuba multiplier which parallel compute the multiplication and second we compute parallel computation in the extension field F_3^{6m} which is very useful for reducing the number of clock cycles which is discussed in the chapter Hardware Implementation.

Last phase of the research is to take results of our implemented protocol and compare it with the previous results. If these results are significantly better than previous results then this are called good research otherwise we have to change our approach and go back to the implementation phase to reconsider our implementation.

Chapter 3

Hardware Implementation

3.1 Addition and Subtraction over $\mathbf{F}_3^m$

The addition and subtraction operation are very simple and just a look up table. As we are working in Galois field F_p^m. So P in our case is 3 and m is 97. The values which comes in the prime field p are (0, 1, 2) which is represented by two bits in binary. Increasing the field from 2 to 3 the number of bits used are increased to double e.g in base 2 there comes only two values (0, 1) but in base field 3 it comes (0, 1, 2). In base 2 only one bit is needed to represent the field members. So by increasing the field the complexity also increases and it brute force or any other attack is much difficult. So there should be one lower bit and one higher bit. The lower bit is represented by a_iL and the higher bit is represented by a_iH. The bits obtained from this is calculated as: the lower bit is $a_iL = a_i \mathrm{mod} 2$ and the higher bit is calculated as $a_iH = a_i/2$.

When we achieve the binary bits then further it is a very simple operation. This is just a look up table to calculate the output. First we make it for just two binary bits which represents adder from F_3^1 which is further extended to F_3^{97}. So for $m = 97$, the binary bits are equal to 194 bits. The input of the Adder and Subtractor module is 194 bits and the output is also of 194 bits. If we have to calculate the addition of three inputs then we have to calculate the sum or subtraction of two values and store result in wire connection if intermediate value is not needed for further use.

If we need this intermediate value then we have to store this value in a register so that we can further use it. All this was done on Virtex-II pro-100 and the complete specification is xc2vp100-6ff1696. The number of 4 input LUTs used for the base adder are 2 and the total number of slices used for adder

F_3^1 is only one.

3.2 Multiplication over F_3^m

The algorithm used for multiplication is a bit complex rather than the algorithm used in addition and subtraction. For multiplication we use the algorithm Most Significant Element (MSE) first as described in the paper by parhi and song [25] which multiplies the two polynomials $a(x)$ and $b(x)$ and stores result say $c(x)$. As we know this is finite field multiplication so the result is also a polynomial which is also a member of this field. When we say a finite field then there is an irreducible polynomial of the given field. We say to the irreducible polynomial $f(x)$. Now we can write the equation as,

$$c(x) = a(x)b(x) \ mod \ f(x) \tag{3.1}$$

In this research we did experiments taking from $D = 1$ to $D = 16$, where D is the number of coefficients processed per cycle by the multiplier. This multiplication algorithm completes in two steps. In first step we have to add up all the D partial products which are obtained by multiplying with the polynomial powers as described in the algorithm 1. This addition requires a polynomial of degree $(m + D - 2)$. We say to that polynomial $t(x)$ and $t(x)$ can be written as,

$$t(x) = a_{3i+2} \ x^2 b(x) \ mod \ f(x) + a_{3i+1} x b(x) \ mod \ f(x) + a_{3i} b(x) \tag{3.2}$$

In our case $m = 97$ so total number of cycles needed to complete this multiplication process are cycles $= [m/D]$. For simplicity we take $D = 3$ and the equation 3.2 is also written for $D = 3$. Therefore the number of cycles needed for the whole multiplication is 32 as shown in figure 3.1 . Now $a(x)$ and $b(x)$ are the two polynomials and the degree of each polynomial is 96. The maximum power that can be on any coefficient is 96. At first cycle we calculate the product as $p(x)= a_{96}b(x)$. $p(x)$ is the final value which we will achieve after 32 clock cycles.

Now we will discuss the entire process which is used to calculate the equation $t(x)$. In the above equation i ranges from 32 down to 1, but for $i = 32$ the above equation is calculated which is a simple multiplication and there is no addition in that equation in first clock cycle. Now, form $i = 31$ to 1 the equation 3.2 is calculated as, for second clock cycle the coefficient of $(3i + 2)$ is multiplied by x^2. After multiplying x^2 it is multiplied by $b(x)$.

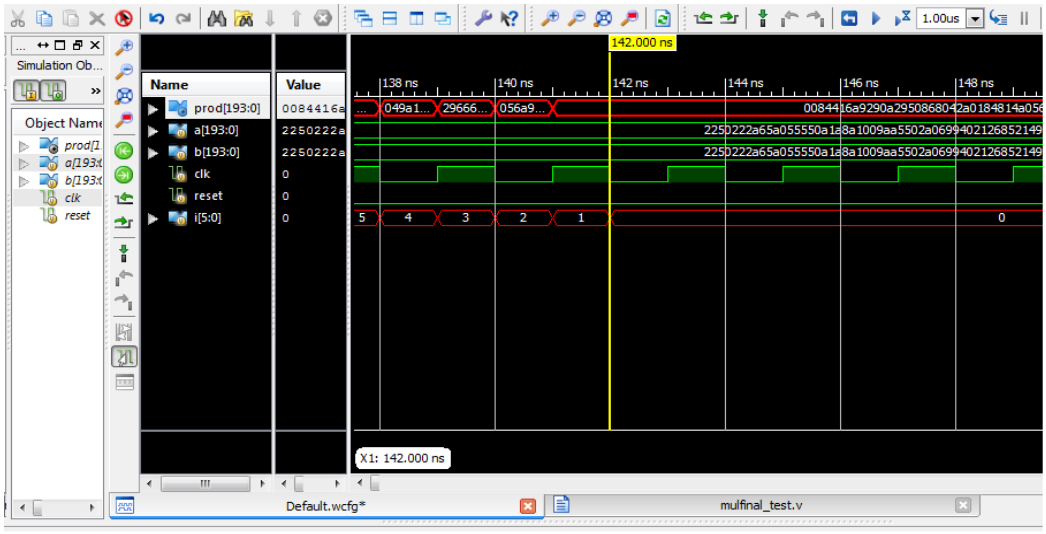

Figure 3.1: No. of cycles for F_3^m

After these multiplications the result is greater than our field so it is reduced by the irreducible polynomial $f(x)$ so that result lies within the field. After reduction the result is stored in the temporary register for the further use.

As we are using $D = 3$, so there should be three partial products. First partial product is calculated in the above paragraph and now for the second partial product the coefficient of $(3i + 1)$ is multiplied by x. After multiplying x it is multiplied by $b(x)$. After these multiplications the result is greater than our finite field. So to get the result into our defined finite field, divide the result by the irreducible polynomial and now the result is within the field. This result is stored in the register for further use which we will need to sum all the three partial products. To get the final partial product, we multiplied the coefficient of $(3i)$ is multiplied by $b(x)$, this result is already within the field and we do not need any division to get the result back into the field.

Now we have all the three partial products which we have stored in the registers. We have to add these three partial product and the result is stored to the polynomial $t(x)$, which we have used for the summation of partial products. Now we come to the update mechanism at every clock cycle. The equation for this update mechanism and complete multiplication for the two

polynomials $a(x)$ and $b(x)$ is given as,

$$p(x) = p(x)x^3 \; mod \; f(x) + a_{3i+2}x^2 b(x) \; mod \; f(x) + a_{3i+1}xb(x) \; mod \; f(x) + a_{3i}b(x) \tag{3.3}$$

$$p(x) = p(x)x^3 \; mod \; f(x) + t(x) \tag{3.4}$$

In equation 3.4 the whole equation consists of two parts. First part is the equation with polynomial $p(x)$. First time this polynomial is calculated at first clock in which coefficient a_{96} is multiplied by the polynomial $b(x)$ and stored in the $p(x)$. This is just for the first clock cycle and for the second clock cycle the existing $p(x)$ is multiplied by the x^3. By multiplying $p(x)$ with x^3 the result is greater than the finite field in which we are working. To get the result back into the finite field we divide the result by the irreducible polynomial of finite field and the result is stored into the register. Now we have both the results e.g $p(x)$ and the $t(x)$ add both these results to get the final $p(x)$ value which is the net result after one clock cycle excluding first clock cycle.

This process continues for 32 clock cycles and after 32 clock cycles we get the final product value of two polynomials $a(x)$ and $b(x)$. The special thing about this multiplication algorithm is that we do not need any reduction algorithm to reduce the final product. Basically we are multiplying and reducing the polynomials of partial products at each step or at each clock cycle. The advantage of this algorithm is that we have not a final product which is long polynomial. And one thing about partial products these are calculated in parallel to avoid the additional clock cycles. As we have all the equations for $D = 3$ so that we can easily understand the mechanism of multiplication and reduction. According to our synthesis and place and route results the number of slices used by the multiplier given $D = 3$ and $D = 4$ are 1131 and 1350 respectively.

Now we will discuss the detail about the multiplication algorithm how it works. For simplicity we wrote the algorithm for $D = 3$. The reason behind to choose this D is that we can easily explain it and the equation are small. The multiplier takes 32 clock cycles. At first clock cycle i.e. for $i = 32$, we calculate $p(x)$ by multiplying a_{96} with $B(x)$. For first cycle $t(x)$ is zero and no contribution to the $p(x)$. For i form 31 down to zero $t(x)$is calculated by multiplying the specific coefficients with x^2, x and then multiplied with the input polynomial $b(x)$. If the answer is greater than the irreducible polynomial then it is reduced to within the finite field. At step 5 $p(x)$ is

updated as written in the algorithm, i.e. multiplying $p(x)$ by x^3 and reduced to finite field and then add up to the polynomial $t(x)$. After completing all these cycles, last value of $p(x)$ is the final product value by multiplying $a(x)$ and $b(x)$and reduction.

3.3 Multiplication over $\mathbf{F}_3^{2m}$

Before going to the multiplication between the two members of F_3^{6m}, we need another tower field extension which is known as a field F_3^{2m}. The theory about this field is already given above; here we discuss the implementation of this field and the mechanism which we followed to get the result. We suppose that A and $B$are the inputs of this module and C is the output. As the field is F_3^{2m}, So there are two values of each input variable i.e. A consists of two parts. To understand this logic we can write A and B as $A = a_1 + a_2\sigma$ and $B = b_1 + b_2\sigma$, where a_1, a_2, b_1 and b_1 are the members of the base field F_3^m.

Multiplication between the elements of field F_3^{2m} needed 32 clock cycles because there is no restriction on our design e.g we cannot use multipliers more than this. Therefore, we have used three multipliers which computes multiplication in parallel. These multiplication are within the field F_3^{97}. We also need five (Adders/Subtractor) in which three are adders and the two subtractor are used. As all the processes are running in parallel so we do not have face any latency in terms of clock cycles. The number of cycles needed to perform multiplication in field are F_3^{2m} are equal to the number of cycles needed to perform multiplication in the field F_3^m. The slices used by this module to multiply the two elements lying in the field F_3^{2m} are 3840.

3.4 Addition and Subtraction over $\mathbf{F}_3^{2m}$

The addition and subtraction over the field F_3^{2m} are very simple operations. Here we consider that A and B are the inputs of this module and C is the output. As the field is F_3^{2m}, So there are two values of each input variable i.e. A consists of two parts. To understand this logic we can write A and B as $A = a_1 + a_2\sigma$ and $B = b_1 + b_2\sigma$, where a_1, a_2, b_1 and b_2 are the members of the base field F_3^m. And the output C is also consists of two parts and we can write it as $C = c_1 + c_2\sigma$. Therefore the addition in the field F_3^{2m} is simply written as c_1=$a_1 + b_1$ and c_2=$a_2 + b_2$. One thing to remember is that while

adding in the field F_3^{2m} is that a_1, a_2, b_1, b_2, c_1 and c_2 are in the field F_3^m.

The subtraction operation is also very simple and almost same as that of addition. For subtraction we consider another variables X and Y are the inputs of this module and Z is the output. As the field is F_3^{2m}, So there are two values of each input variable i.e. X consists of two parts. To understand this logic we can write X and Y as $X = x_1 + x_2\sigma$, $Y = y_1 + y_2\sigma$, where x_1, x_2, y_1 and y_2 are the members of the base field F_3^m. And the output Z also consists of two parts. We can write it as $Z = z_1 + z_2\sigma$. Therefore the subtraction in the field F_3^{2m} is simply written as z_1=$x_1 - y_1$ and $z_2 = y_2 - y_2$. One thing to remember is that while adding in the field F_3^{2m} is that x_1, x_2, y_1, y_2, z_1 and z_2 are in the field F_3^m.

3.5 Multiplication over F_3^{6m}

Now we come to the main module which is the multiplication of of two input variables which lies in the field F_3^{6m} and we say to the variables as $R0$ and $R1$. Field F_3^{6m} is nothing but a simple extension of the base field F_3^m. To construct the field F_3^{6m} is very simple. We just need the basis vectors to construct the extension field. We discuss here a brief introduction here just to recall it and the complete study about the extension field is given in the literature survey as discussed above.

Taking advantage of Karatsuba Ofman algorithm [26] which provides the net result by calculating the constant coefficients of the input variables $R0$ and $R1$ in parallel. As we have no restrictions on the area in terms of slices or we have no restriction to use the number of multipliers exceeding the limit. For complete multiplication in the field F_3^{6m} we have used 40 adders which are used to calculate the addition in the field F_3^m and 18 subtractor which subtracts the input variables in the base field F_3^m and all the operation take place are in the base field F_3^m.

Now, come to the number of multipliers used in the multiplication of the field F_3^{6m}. We have used total 18 multipliers. We used 6 times the multiplier in the field F_3^{2m} in which we have used 3 multipliers in one multiplication module. So by using six times we totally used 18 multipliers in the base field F_3^m. As we totally focused on reducing the number of clock cycles for this multiplication. Therefore we used maximum number of adder, subtractor and multipliers which result in the minimum number of clock cycles. The number of cycles used for the multiplication in the field F_3^{6m} is exactly the same as that of cycles used in by the multiplier in the field F_3^m as shown in

figure 3.2 as cycles are 32. The number of cycles can be calculated as $[m/D]$ where D is the number of coefficients processed per cycle by the multiplier.

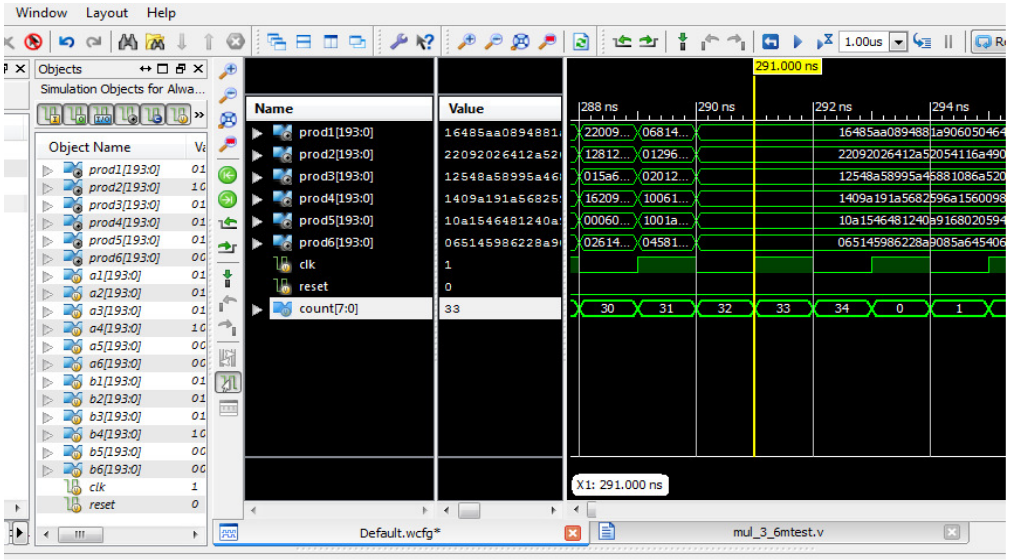

Figure 3.2: No. of cycles for F_3^{6m}

Suppose,

$$A = a_0 + a_1\sigma + a_2\rho + a_3\rho\sigma + a_4\rho^2 + a_5\sigma\rho^2 \tag{3.5}$$

$$B = b_0 + b_1\sigma + b_2\rho + b_3\rho\sigma + b_4\rho^2 + b_5\sigma\rho^2 \tag{3.6}$$

These are the two variables of the finite field F_3^{6m} where a_0, a_1, a_2, a_3, a_4, a_5 and b_0, b_1, b_2, b_3, b_4, b_5 are the vectors which lies on the field F_3^m and 1, σ, ρ, $\sigma\rho$, ρ^2, $\sigma\rho^2$ are the basis vectors corresponding to the vectors a_0, a_1, a_2, a_3, a_4, a_5 respectively. Now come to the point how multiplication take place, first multiply a_0, a_1 with b_0, b_1 and we store the result at temporary place called temp1 and temp2 and at second a_2, a_3 with b_2, b_3 and a_4, a_5 with b_4, $b_5$and store at some place temp3, temp4, temp5 and temp6 respectively. All three multiplication are calculated in parallel and being two values a_0, a_1, b_0, b_1 of A and B this makes it to the field F_3^{2m}. So the result is also in two vectors form.

Now we have to add the a_5, a_6 with a_3, a_4. This becomes the input of the addition module in the field F_3^{2m} and the output is also of two vectors. These

results are stored in temporary storage place so that we can use it for further calculation. The storage values are called aux11 and aux12. This addition operation also applies for the second variable of field F_3^{2m} B. Add the b_5, b_6 with b_3, b_4. This becomes the input of the addition module in the field F_3^{2m} and the output is also of two vectors. The results are stored in aux21 and aux22. Now, by using the stored values we multiply aux11, aux12 with aux21, aux22. These values becomes the input of the multiplication module in the field F_3^{2m} and calculates the answer lying in this field which also consists of two vectors. These results are stored to the temp0rary values called temp7 and temp8. All the storage values are declared as wires.

Before coming to final result vectors, the results already calculated and stored at temporary storage are uses for the calculation of the vectors of final result after multiplication in the field F_3^{6m}. First we discuss how the first two vectors are calculated corresponding to the basis vectors 1 and σ. Add the temp3, temp4 with temp5, temp6. This becomes the input of the addition module in the field F_3^{2m} and the output is also of two vectors. These results are stored in temporary storage place so that we can use it for further calculation of final vectors of the product. The result is stored to aux31 and aux32. Then add the temp7, temp8 with temp1, temp2 . This becomes the input of the addition module in the field F_3^{2m} and the output is also of two vectors. These results are stored in aux41 and aux42. After calculating this, vectors corresponding to the basis vectors 1 and σ are calculated as, subtract aux31, aux32 from aux41, aux42. These are the inputs of the subtraction module in the field F_3^{2m}. So the answer is also in the two vectors form which are stored to the output called as prod1 and prod2. Prod1 is the vector corresponding to the basis vector 1 and prod2 is the vector corresponding to the basis vector σ.

Now we discuss how to calculate the components or vectors corresponding to the basis vectors ρ and $\sigma\rho$. Add the vectors a_1, a_2 with a_3, a_4 and store the results to the auxb1 and auxb2 and also add the vectors b_1, b_2 with b_3, b_4 and store the results to the auxb3 and auxb4. These are the inputs and outputs to the two independent modules for addition. Because we did not compromise on a single clock cycle and the entire processing takes place are parallel. The saved results in the temporary values at auxb1 and auxb2 and auxb3 and auxb4 are the inputs to the inputs of the multiplication module in the field F_3^{2m} and after multiplication the result is stored in arbitrary values named as temp9 and temp10.

This multiplication is calculated in just 32 clock cycles which is exactly the

same as in the field of F_3^m, because all the multiplication in the field F_3^{2m} takes place parallel. The addition module used in calculation of third and fourth coefficient are just the look up tables and need no clock cycle and the addition module outputs are straight forward input to the multiplication module and are just the wire connections. Now to calculate the final coefficients corresponding to the basis vectors ρ and $\sigma\rho$ requires some extra addition and subtractions which are calculated as, add temp9, temp10 with the temp7 and temp8 and we get the result and stored in aux51 and aux52.

Furthermore, this aux51, aux52 are added to the temp3 and temp4 and store results in the arbitrary value called aux61 and aux62. Up to this step all the necessary calculation has been calculated by using addition, subtraction and the multiplication modules. To achieve the vector corresponding to the basis vector ρ and $\sigma\rho$ is calculate by subtracting the temp1, temp2 from aux61 and aux62 and store result in the out named as prod3 and prod4. Prod3 is the vector corresponding to the base ρ and prod4 corresponding to the base vector $\sigma\rho$

Last two vectors corresponding to the basis vectors $\rho^2, \sigma\rho^2$ are calculate as, add the vectors a_1, a_2 with a_5, a_6 and store the results to the aux71 and aux72 and also add the vectors b_1,b_2 with b_5, b_6 and store the results to the aux81 and aux82. These are the inputs and outputs to the two independent modules for addition. Because we did not compromise on a single clock cycle and the entire processing takes place are parallel. The saved results in the temporary values at aux71 and aux72 and aux81 and aux82 are the inputs to the inputs of the multiplication module in the field F_3^{2m} and after multiplication the result is stored in arbitrary values named as aux91 and aux92. This multiplication is calculated in just 32 clock cycles which is exactly the same as in the field of F_3^m, because all the multiplication in the field F_3^{2m} takes place parallel.

The addition module used in calculation of fifth and sixth coefficient are just the look up tables and need no clock cycle and the addition module outputs are straight forward input to the multiplication module and are just the wire connections. Now to calculate the final coefficients corresponding to the basis vectors ρ^2 and $\sigma\rho^2$ requires some extra addition and subtractions which are calculated as, add aux91, aux92 with the temp3 and temp4 and we get the result and stored in aux101 and aux102. Upto this step all the necessary calculation has been calculated by using addition, subtraction and the multiplication modules. To achieve the vector corresponding to the basis vector ρ^2 and $\sigma\rho^2$ is calculate by subtracting the aux101, aux102 from temp1,

temp2 and store result in the output port named as prod5 and prod6. Prod5 is the vector corresponding to the base ρ^2 and prod6 corresponding to the base vector $\sigma\rho^2$

3.6 Cubing over $\mathbf{F}_3^m$

Cubing is a very simple arithmetic operation which is just like that of the addition and subtraction modules as discussed in the previous topics. Suppose $a(x)$ and $b(x)$ be the two polynomials which lies in the field F_3m and we have to compute the cube of $a(x)$, we can write it as,

$$b(x) = a(x)^3 = \sum_{i=0}^{m-1} a_i x^{3i} \ mod \ f(x) \tag{3.7}$$

Where $f(x)$ is the irreducible polynomial of degree m which is 97 in our case which is $f(x) = x^{97} + x^{12} + 2$ and i is simple counter which varies from zero to $m - 1$. As the equation is very simple and can be solved very easily. We wrote a simple Matlab code for the calculation of the coefficients. Formula for the coefficient calculation over the field F_3^m is given as,

$$\begin{gathered} b_0 = a_{93} + a_{89} + a_0,\ b_1 = a_{65} - a_{61}, \\ b_2 = a_{33},\ b_3 = a_{94} + a_{90} + a_1, \\ . \\ . \\ . \\ b_{96} = a_{32}. \end{gathered}$$

After getting all the equations cubing is as simple as addition or subtraction, just the addition of two or three polynomials. The addition of two polynomials is very simple and we just need on adder circuit for the calculation of one coefficient but for the addition of the polynomials we have to add the two polynomials and store it in arbitrary value and then add this result to the third polynomial to get the final answer of specific coefficient. The most complex operation above all the operations is the addition of three components or coefficients which we handle by adding two elements and storing it to somewhere and then add the third element to get the final coefficient value.

3.7 Cubing over F_3^{6m}

As the field F_3^{6m} is simply the extension of the field F_3^m. So cubing over the field F_3^{6m} is simply replaced by six cubing and addition over the field F_3^m and two negations over the field F_3^m. The calculation takes place as first we take the cube of all the vectors corresponding to the basis vectors e.g $a_1, a_2, a_3, a_4, a_5, a_6$ and store the result in the temporary values called temp1, temp2, temp3, temp4, temp5 and temp6. Pre-calculation is complete and now move towards the final answer of the cube in the field F_3^{6m} which is attained after some additions, subtractions and negations. First we add temp1 with temp2 and store the result in the arbitrary place sum1 and then add sum1 with the temp5 and get the answer of the first coefficient corresponding to the basis vector 1.

For second vector calculation, add temp2 with temp4 and store the result in the arbitrary place sum2 and then add sum2 with the temp6 and store the result in the arbitrary place sum3. After this the result of sum3 is multiplied with 2 to get the negative of this answer and then we get the answer of the second coefficient corresponding to the basis vector σ. Third vector corresponding to the basis vector ρ is very simple and we just need to subtract the value stored in the temp5 from the value stored in temp3 and we get our final answer corresponding to the basis vector ρ. Fourth vector corresponding to the basis vector $\sigma\rho$ is also very simple and we just need to subtract the value stored in the temp4 from the value stored in temp6 and we get our final answer corresponding to the basis vector $\sigma\rho$.

One thing to remember is that all the calculations taken place are in the base field F_3^m and the field F_3^{6m} is just the extension of the base field which is attained after some operations with the basis vectors of the field. The fifth vector corresponding to the basis vector ρ^2 is pretty simple and we have to just assign the value to this basis vector and we do not need any addition, subtraction and multiplication. For last vector we need a single multiplier which multiplied the value stored in temp6 variable with 2 just to get the negative of this vector and assign it to the vector corresponding to the basis vector $\sigma\rho^2$. From hardware prospective all the temporary values which we have used are the simple wire connections and the cube in the field F_3^{6m} is as simple as in the base field F_3^m. Because it just need a simple look up table to find the cube in the extension of the base field.

3.8 Final Eta-T pairing

We have described all the modules and its hardware implementations needed to calculate the final Eta-T pairing. We take the algorithm as described in the [11]. We have not changed its round or any other calculation but we have contributed at its implementation level by calculating Eta-T pairing in parallel rather than serial as described in the paper by Buchat [11].The algorithm which we have implemented is given as:

Pairing Algorithm

Input: $P = (x_p, y_p)$ and $Q = (x_q, y_q) \in E^b(\mathbb{F}_{3^m})[l]$.
The algorithm requires R_0 and $R_1 \in \mathbb{F}_{3^{6m}}$, as well as $r_0 \in \mathbb{F}_{3^m}$ and $d \in \mathbb{F}_3$ for intermediate computations.
Output: $\eta_T(P,Q)^{3^{(m+1)/2}}$

1: **if** $b = 1$ **then**
2: $y_p \leftarrow -y_p$;
3: **end if**
4: $r_0 \leftarrow x_p + x_q + b$;
5: $d \leftarrow b$;
6: $R_0 \leftarrow -y_p r_0 + y_q \sigma + y_p \rho$;
7: **for** $i = 0$ to $(m-1)/2$ **do**
8: $r_0 \leftarrow x_p + x_q + d$;
9: $R_1 \leftarrow -r_0^2 + y_p y_q \sigma - r_0 \rho - \rho^2$;
10: $R_0 \leftarrow (R_0 R_1)^3$;
11: $y_p \leftarrow -y_p$;
12: $x_q \leftarrow x_q^9$; $y_q \leftarrow y_q^9$;
13: $d \leftarrow (d - b) \bmod 3$;
14: **end for**
15: Return R_0;

Figure 3.3: Eta-T Pairing Algorithm

After discussing the algorithm which is implemented we move towards the diagram in which we can show that how two input points enter into the modules and and after several round of calculation it returns the final value of pairing between the two points and output comes after specific rounds. The diagram of the overall pairing system is given as in figure 3.4

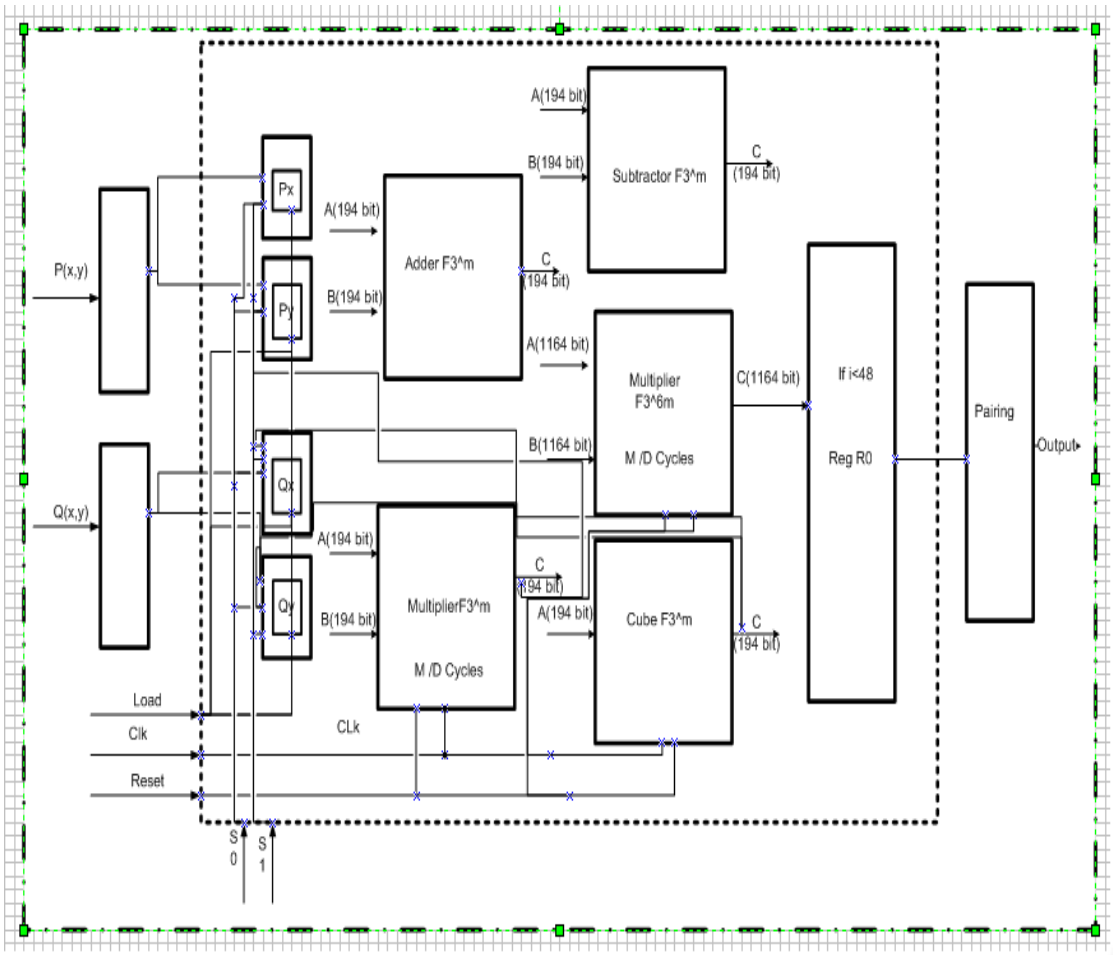

Figure 3.4: Complete Eta-T pairing System

Chapter 4

Results and Comparison

4.1 Efficient implementation results on Virtex-IV

All the proposed architecture and its implementation are done on project Navigator 14.2 Xilinx family which is Virtex-IV xc4vlx200-11ff1513 device. For simulation we use its ISim from where we see how much clock cycles it will take to calculate the final Eta-T pairing. Our main purpose to implement this pairing algorithm is to reduce the maximum number of clock cycles as we can. Therefore, we utilize maximum parallelization of the modules to reduce the number of clock cycles. Number of clock cycles depend upon D (No. of coefficients processed per cycle by the multiplier) and also the area occupied by the pairing algorithm. As we only focus on the number of cycles and we do not bother about the area. In the table 4.1 we gave the details about the number of clock cycles, area in terms of slices and the total time required by the pairing module to calculate the complete pairing.

In the above table 4.1 D represents the number of coefficients per second by the multiplier. We started D from 1 to 16 and get the results. Top to bottom D increases up to 16 and the curve remains the same for every which is elliptic curve. The curve we use in this case is $y^2 = x^3 - x + 1$ and we choose $b = 1$ and $a = -1$. In the third column of FPGA we use the device Virtex-IV for every D. As the value of D increases downwards the number of cycles decreases. For $D = 3$ the number of cycles used are 3597 and decreasing downward with the increase of D. Area at the top is less and started increasing as we increase the value of D. For $D = 3$ the slices used for the computation of whole pairing are 32223 and is gradually increasing

Table 4.1: Efficient implementation Virtex-IV results

D	Curve	FPGA	Clock cycles	Clock Frequency	Area	Calculation Time
1	Elliptic	Virtex-IV	9613	146.4	24465	65.70
2	Elliptic	Virtex-IV	5005	124	28562	40.36
3	Elliptic	Virtex-IV	3597	114.65	32668	31.37
4	Elliptic	Virtex-IV	2773	104.94	37168	26.42
5	Elliptic	Virtex-IV	2287	92.45	38509	24.73
6	Elliptic	Virtex-IV	1997	92.62	45450	21.55
7	Elliptic	Virtex-IV	1705	83.83	46622	20.32
8	Elliptic	Virtex-IV	1609	78.36	50791	20.53
9	Elliptic	Virtex-IV	1415	73.88	54978	19.15
10	Elliptic	Virtex-IV	1317	69.58	59166	18.92
11	Elliptic	Virtex-IV	1219	66.81	63134	18.24
12	Elliptic	Virtex-IV	1219	64.66	67662	18.85
13	Elliptic	Virtex-IV	1121	61.74	70343	18.15
14	Elliptic	Virtex-IV	1023	59.89	71232	17.08
15	Elliptic	Virtex-IV	1023	56.7	74489	18.04
16	Elliptic	Virtex-IV	1023	52.42	77465	19.51

downward.

Now, move towards the last column which is the calculation time for the Eta-T pairing. Calculation time is highest for $D = 1$ and it start decreasing with the increase in the D. When D reaches at 11 time decreases and from $D = 11$ to $D = 12$ time increases due to the decrease in the clock frequency but the number of cycles remains the same as that of when $D = 11$. So this causes the Time to increase. From $D = 12$ to $D = 14$ calculation time again starts decreasing. At $D = 14$ we get the minimum time to calculate the Eta-T pairing. After $D = 14$ it starts increasing and there is slight decrease in the time when at specific value of D at which there is decrease in the clock cycle and the clock frequency. Because after $D = 14$ the there is a very less effect on the clock cycles and the clock frequency is decreasing gradually. At $D = 14$ the calculation time is minimum as shown in the graph 4.1. So for safe value of D we have to choose $D = 14$ or less than 14.

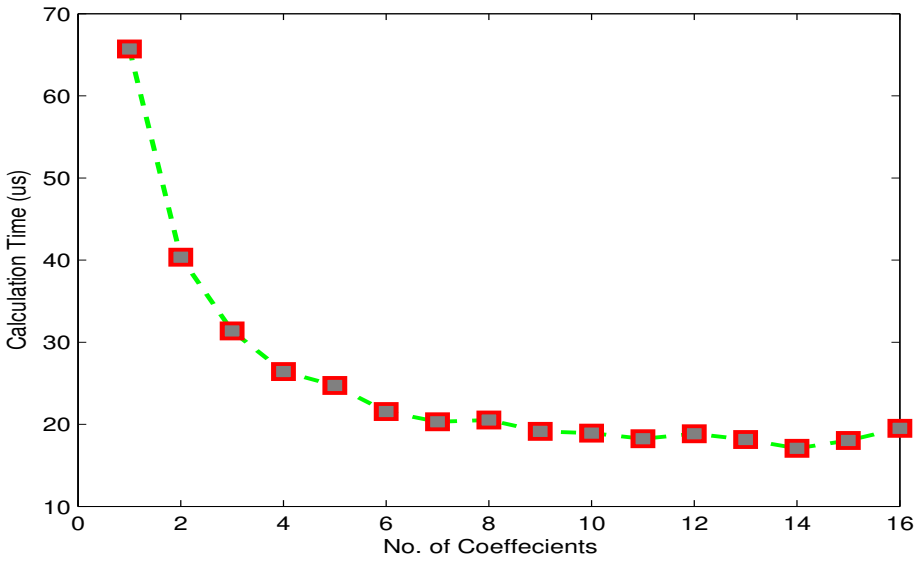

Figure 4.1: Calculation Time Graph in Virtex-IV

4.2 Efficient implementation results on Virtex-II

All the proposed architecture and its implementation are done on a Xilinx family which is Virtex-II device. Our main purpose to implement this pairing algorithm is to reduce the maximum number of clock cycles as we can. Therefore, we utilize maximum parallelization of the modules to reduce the number of clock cycles. Number of clock cycles depend upon D (No. of coefficients processed per cycle by the multiplier) and also the area occupied by the pairing algorithm. As we only focus on the number of cycles and we do not bother about the area. In the table we gave the details about the number of clock cycles, area in terms of slices and the total time required by the pairing module to calculate the complete pairing.

In the above table, D represents the number of coefficients per second by the multiplier. We started D from 1 to 16 and get the results. Top to bottom D increases up to 16 and the curve remains the same for every which is elliptic curve. The curve we use in this case is $y^2 = x^3 - x + 1$ and we choose $b = 1$ and $a = -1$. In the third column of FPGA we use the device Virtex-II for every D. As the value of D increases downwards the number of cycles decreases. For $D = 3$ the number of cycles used are 3597 and decreasing downward

Table 4.2: Efficient implementation Virtex-II results

D	Curve	FPGA	Clock cycles	Clock Frequency	Area	Calculation Time
1	Elliptic	Virtex-II	9613	130.1	25476	73.88
2	Elliptic	Virtex-II	5005	108.4	29683	46.17
3	Elliptic	Virtex-II	3597	98.79	33465	36.41
4	Elliptic	Virtex-II	2773	87.34	38675	31.74
5	Elliptic	Virtex-II	2287	76.12	38981	30.04
6	Elliptic	Virtex-II	1997	76.32	46121	26.16
7	Elliptic	Virtex-II	1705	67.21	46622	25.36
8	Elliptic	Virtex-II	1609	63.61	51734	25.29
9	Elliptic	Virtex-II	1415	58.4	55342	24.22
10	Elliptic	Virtex-II	1317	54.72	60645	24.06
11	Elliptic	Virtex-II	1219	51.8	64535	23.53
12	Elliptic	Virtex-II	1219	47.46	68612	25.68
13	Elliptic	Virtex-II	1121	44.34	71002	25.28
14	Elliptic	Virtex-II	1023	42.56	73087	24.03
15	Elliptic	Virtex-II	1023	39.61	76462	25.82
16	Elliptic	Virtex-II	1023	35.32	79822	28.96

with the increase of D. Area at the top is less and started increasing as we increase the value of D. For $D = 3$ the slices used for the computation of whole pairing are 32223 and is gradually increasing downward.

Now, move towards the last column which is the calculation time for the Eta-T pairing. Calculation time is highest for $D = 1$ and it start decreasing with the increase in the D. When D reaches at 11 time decreases and from $D = 11$ to $D = 12$ time increases due to the decrease in the clock frequency but the number of cycles remains the same as that of when $D = 11$. So this causes the Time to increase. From $D = 12$ to $D = 14$ calculation time again starts decreasing. At $D = 14$ we get the minimum time to calculate the Eta-T pairing. After $D = 14$ it starts increasing as shown in the graph4.2 and there is slight decrease in the time when at specific value of D at which there is decrease in the clock cycle and the clock frequency. Because after $D = 14$ the there is a very less effect on the clock cycles and the clock frequency is decreasing gradually. So for safe value of D we have to choose $D = 14$ or less than 14.

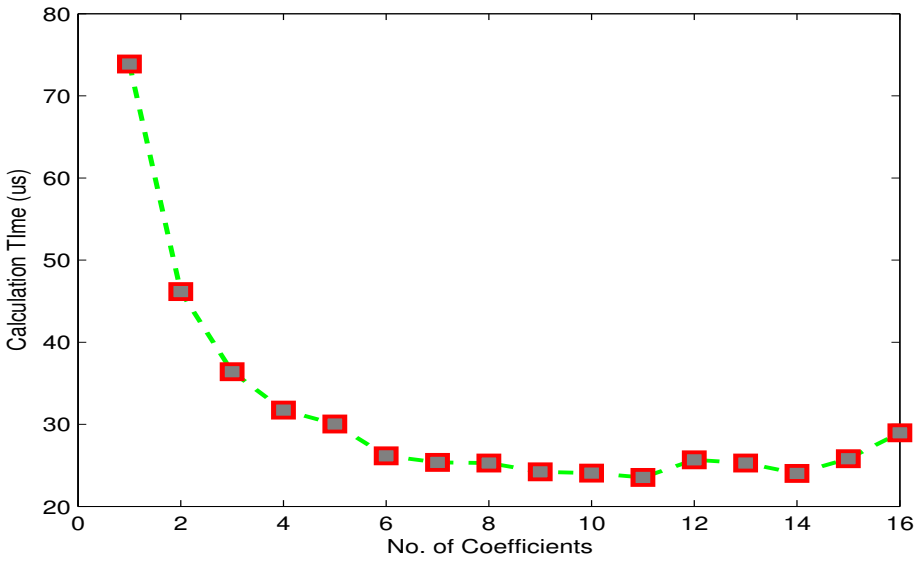

Figure 4.2: Calculation Time Graph in Virtex-II

4.3 Comparison

To the best of our knowledge, the fastest calculation time of Eta-T pairing is 34μs. by Shy, Kown and Gaj [27]. The field used was F_2^{239} and the curve is elliptic and the device used is Xilinx Virtex-II pro100 as well as total multipliers used in this pairing module are six. They achieved the maximum clock frequency 84MHz and the total area occupied by this is 25287 slices. Ronan et al.[11] calculate the Eta-T pairing and the calculation time of Eta-T pairing is 749μs. He used the field F_2^{103} and the curve used is the hyper elliptic curve. They used the Xilinx device Virtex-II pro125 and the total multipliers used in this pairing module are twelve. They achieved the maximum clock frequency 32.3MHz and the total area occupied by this is 43986 slices.

Grabber and Page [28] also proposed an algorithm which is the modified form of Duursma Lee algorithm. They used multiplication over the field F_3^{6m} and used Karatsuba Ofman algorithm to take the advantage of the parallelism but the main flaw in this implementation is they did not take the advantage of constant coefficient in R1 and they used 18 multipliers. The total cost of this algorithm is, they used 18 multipliers, 6 cubing modules over the field F_3^m. In this algorithm some extra circuits were also used to perform the addition, subtraction and the negation. This makes it extra heavy and the author also claims that it used almost 100% of the resources. He used

the field F_3^{97} and the curve used is the elliptic. They used the Xilinx device Virtex-II pro125 and the total multipliers used in this pairing module are 18. They achieved the maximum clock frequency 15MHz and the total area occupied by this is 55616 slices. Time needed to compute the Eta-T pairing is $859\mu s$ which is very large time for pairing calculation. This is because the very low clock frequency and the clock cycles used to calculate the pairing is also very large which 12866 cycles are. In this paper they calculate the final exponentiation which also uses half of the resources to calculate it. If we suppose that half of the resources are used for final exponential then the remaining time for the pairing calculation is also very high.

Grabber and Page [28] calculate the Eta-T pairing and the calculation time of Eta-T pairing is $399.4\mu s$. In this paper, field used is F_3^{97} and the curve is elliptic. They used the Xilinx device Virtex-II pro-IV and the total multiplier used in this pairing module is 1. They achieved the maximum clock frequency 150MHz and the total area occupied by this is 4481 slices. In this paper they did not calculate the final exponentiation which almost uses half of the resources to calculate it. If we suppose that half of the resources are used for final exponential then the time for the complete pairing becomes very high. Jean Luc Buchat [13] also proposed a new algorithm which calculates the Eta-T pairing in just $33\mu s$. They used Altera device for the calculation of the pairing and the device is Cyclone-IIEP2C35. We just give the specification of the pairing module but we did not include it into our comparison table due to difference of device. They used the field F_3^{97} and the curve used is the Elliptic. The total multipliers used in this pairing module are 9. They achieved the maximum clock frequency 149MHz and the total area occupied by this is 4849 LE's.

Now, we come to our implementation, we discuss our implementation both on Virtex-IV and Virtex-II as given in the above tables. We found most of the implementation in Virtex-II. Therefore, we compare our Virtex-II results with the other implementations which were given on Xilinx family Virtex-II. In the table all the results are given by using the D= 3 and we also give our results which are obtained by using the specific value of D which is equal to 3. The number of cycles used to calculate the pairing are 3597 and the slices needed for this module are 32212 and the curve used is the same as that of used by beuchat [13] algorithm which is on super singular elliptic curve over the field F_3^{97}. We mainly focus to reduce the number of clock cycles. We use the minimum number of clock cycles among all the algorithms proposed in other papers by different authors. Therefore, we achieved maximum parallelization which can achieved and we do not care much about the use of area. So, there

may be a bit more utilization of over resources. The calculation time is also very low for our implementation and is $36\mu s$ for D= 3 and 32 μs for D= 4. The frequency achieved by our algorithm is 98 MHz for D= 3 and 84 MHz for D= 4. As we achieved maximum parallelization so we use more multipliers as compared to the other algorithms given in the comparison table.

Table 4.3: Comparison between previous FPGA Implementations

	Ronan [11]	Grabber [28]	Kerins [29]	Buchat [13]	Our Results
Algorithm	η_T	Duursma	Duursma	η_T	η_T
Finite Field	F_2^{103}	F_3^{97}	F_3^{97}	F_3^{97}	F_3^{97}
Curve	Elliptic	Elliptic	Elliptic	Elliptic	Elliptic
FPGA	Virtex-II	Virtex-II	Virtex-II	Cyclone-II	Virtex-II
Multipliers	12	1	18	9	18
Area	43986	4481	55616	14895	33465
Clock Cycles	-	-	12866	4849	3597
Clock Frequency	32.3 MHz	150 MHz	15 MHz	149 MHz	98.79MHz
Calculation time	$749\mu s$	$399.4\mu s$	$850\mu s$	$33\mu s$	$36\mu s$
Final exponential	Yes	No	Yes	No	No

Chapter 5

Conclusion and Future Work

5.1 Conclusion

Online key sharing is big problem in the current security systems. If key is compromised during its computation then whole of the transactions are compromised. Pairing solved this problem to share the key between three users. We have proposed modified Eta-T pairing algorithm which did not require any cube root over the field F_3^{97}. Furthermore, we use Karatsuba multiplier which computes multiplication in the given field as well as the extension of the base field in same number of cycles as we used in the multiplier F_3^{97}. Cube operator is just the look up tables and calculation is very simple as that of adder and subtractor. Finally, we compute Eta-T pairing in just 3597 clock cycles for D equal to three which is the minimum number of clock cycles for this value of D. We also extend our work and calculate pairing by increasing the value of D up to 16 and explore D at which calculation time starts increasing. Up to 14 the calculation time decreases gradually because by increasing D there is large effect on the clock cycles. After $D = 14$ there is less effect on the number of clock cycles but frequency decreases gradually. Therefore, calculation time starts increasing after $D = 14$. All these results are implemented and verified on Virtex-II and Virtex-IV. In our implementation we also use maximum parallelization to reduce the maximum number of clock cycles and we use only 3597 clock cycles for $D = 3$ to calculate the Eta-T pairing which is 25% less than the algorithm proposed which we have implemented.

5.2 Future Work

The work presented can be extended to many research problems. The problems which will come in future are listed as,

- The area used in this protocol is higher because we parallelized it maximum to achieve result in minimum number of clock cycles.
- Change the elliptic curve with another and compared the result which may useful be get better results.

Bibliography

[1] W. Diffie and M. E. Hellman, New directions in cryptograpgy, IEEE Trans. Inform. Theory, IT-22(6), November 1976.

[2] L. M. Adleman, R.L.Rivest and A. Shamir, A method for obtaining digital signatures and public-key cryptosystems, Comm.ACM,21(2):120-126,1978.

[3] N. Koblitz, Elliptic curve cryptosystems, Math. Comp., 48(177):203-209, January 1987.

[4] V. Miller, Use of elliptic curves in cryptography, in A. M. Odlyzko, editor, Advances in cryptology - CRYPTO 86, volume 263, of lecture notes in comput. sci. pages 417-426, Spriner - Verlag, 1987. Proceedings, Santa Barbra (USA), August 11-15, 1986.

[5] G. Frey and H.-G. ,Ruck, A remark concerning m-divisibility and the discrete logarithm in the divisor class group of curves, Math. Comp., 62(206):865-874, April 1994.

[6] R. Sakai, K. Ohgishi and M. Kasahara, Cryptosystems based on pairings, SCIS 2000, the 2000 symposium on cryptography and information security, Okinawa, japan, January 26-28.

[7] A. Joux, A one round protocol for tripartite Diffie-Hellman, in W. Bosma, editor, ANTS-IV, volume 1838, of lecture notes in comput. sci. pages 358- 394, Spriner Verlag, 2000. 4th International Symposium, ANTS-IV, Leiden, The Netherlands, July 2-7, 2000, Proceedings.

[8] A. Shamir, Identity-based cryptosystems and signatures schemes, in Advances in cryptology-CRYPTO84, volume 196 of lecture notes in Comput. Sci., pages 47-53, Spriner Verlag 1985. 4th annual international cryptography conference, Santa Barabra USA, 19-22 august 1984.

[9] D. Boneh and M. Franklin, Identity-based encryption from the weil pairing, in J. Kilian, editor, Advances in Cryptology - CRYPTO 2001, volume 2139, of lecture notes in comput. sci. pages 213-229, Spriner Verlag, 2001.

[10] C. Cocks, An identity based encryption scheme based on quadratic residues, in B. Honary, editor, cryptograpgy and Coding, volume 2260, of lecture notes in comput. sci. pages 360-363, Spriner Verlag, 2001. 8th IMA International Conference, UK, 2001. Proceedings.

[11] The Weil Pairing, and Its Efficient Calculation VS Miller - Journal of Cryptology, 2004 - Springe.

[12] SD Galbraith, K Harrison, D Soldera - Algorithmic number theory, 2002 - Springer.

[13] An Algorithm for the Eta-T Pairing Calculation in Characteristic Three and its Hardware Implementation,Jean-Luc Beuchat, Masaaki Shirase, Tsuyoshi Takagi, and Eiji Okamoto.

[14] Simple Power Analysis of Unified Code for ECC Double and Add-Cryptographic Hardware and Embedded Systems - CHES 2004 6th International Workshop Cambridge, MA, USA, August 11-13, 2004.

[15] J. H. Silverman, The arithmetic of elliptic curves, volume 106 of Grad. Texts in math. Spriner-Verlag, 1986.

[16] L. C. Washington, Elliptic Curves: Number Theory and Cryptography, Chapman Hall/CRC, 2003.

[17] Helmut Hasse, Aabstrakte Bergrundung der Komplexen Multiplikation und Riemannsche Vermutung in Funktionenkopern, Abhandlungen aus dem mathematischen Seminar der hamburgischen universitat, 10:325-348, 1934.

[18] L. C. Washington, Elliptic Curves: Number Theory and Cryptography, Chapman Hall/CRC, 2003.

[19] R.Balasubramanian and N.Koblitz, The improbability that an elliptic curve has subexponential discrete log problem under the Menezes-Okamoto- Vanstone algorithm, J.of cryptology,11:141-145,1998.

[20] V. S. Miller. Short programs for functions on curves. Unpublished manuscript available at http://crypto.stanford.edu/miller/miller.pdf, 1986.

[21] P. S. L. M. Barreto, H. Y. Kim, B. Lynn, and M. Scott. Efficient algorithms for pairing-based cryptosystems. In M. Yung, editor, Advances in Cryptology CRYPTO 2002, number 2442 in Lecture Notes in Computer Science, pages 354368. Springer, 2002.

[22] S. D. Galbraith, K. Harrison, and D. Soldera. Implementing the Tate pairing. In C. Fieker and D. Kohel, editors, Algorithmic Number Theory ANTS V, number 2369 in Lecture Notes in Computer Science, pages 324337. Springer, 2002.

[23] S. Kwon. Efficient Tate pairing computation for supersingular elliptic curves over binary fields. Cryptology ePrint Archive, Report 2004/303, 2004.

[24] A. Menezes, T. Okamoto and S. A. Vanstone, Reducing elliptic curves logarithms to logarithms in a finite field, IEEE Trans. Inform. Theory, IT-39(5):1639-1646, september 1993.

[25] L. Song and K. K. Parhi. Low energy digit-serial/parallel finite field multipliers. Journal of VLSI Signal Processing, 19(2):149166, July 1998.

[26] D. Zuras. More on squaring and multiplying large integers. IEEE Transactions on Computers, 43(8):899908, Aug. 1994.

[27] C. Shu, S. Kwon, and K. Gaj. FPGA accelerated Tate pairing based cryptosystem over binary fields. Cryptology ePrint Archive, Report 2006/179, 2006.

[28] P. Grabher and D. Page. Hardware acceleration of the Tate Pairing in characteristic three. In J. R. Rao and B. Sunar, editors, Cryptographic Hardware and Embedded Systems CHES 2005, number 3659 in Lecture Notes in Computer Science, pages 398411. Springer, 2005.

[29] T. Kerins, W. P. Marnane, E. M. Popovici, and P. Barreto.Efficient hardware for the Tate Pairing calculation in characteristic three. In J. R. Rao and B. Sunar, editors, Cryptographic Hardware and Embedded Systems CHES 2005, number 3659 in Lecture Notes in Computer Science, pages 412426. Springer, 2005.

[30] Multiplication over Fpm on FPGA: A Survey Jean-Luc Beuchat, Takanori Miyoshi, Yoshihito Oyama, and Eiji Okamoto.

[31] Fast Architectures for the Eta-T Pairing over Small-Characteristic Supersingular Elliptic Curves Jean-Luc Beuchat, Je remie Detrey, Nicolas Estibals, Eiji Okamoto, Member, IEEE, and Francisco Rodriguez-Henriquez

Appendix A

Appendix

A.1 Eta T Pairing

A.1.1 Addition over $\mathbf{F}_3^m$

```
//////////sum3to1/////////////
module sum_3_1(x,y,sum );
input [1:0] x,y;
output [1:0] sum;
assign sum[0]=((x[0] & ~x[1] & ~y[0] & ~y[1] ) || ( ~x[0] & ~x[1]
&y[0] & ~y[1] ) ||(~x[0] &x[1] &~y[0] &y[1]) );
assign sum[1]=((~x[0] & x[1] & ~y[0] & ~y[1] ) || ( x[0] & ~x[1]
&y[0] & ~y[1] ) ||(~x[0] &~x[1] &~y[0] &y[1]) );
endmodule

//////////sum3to2/////////////
module sum_3_2(a,b,sum);
input [3:0]a,b;
output [3:0] sum;
sum_3_1 x1 (a[1:0],b[1:0],sum[1:0]);
sum_3_1 x2(a[3:2],b[3:2],sum[3:2]);
endmodule

//////////sum3to4/////////////
module sum_3_4(a,b,sum);
input [7:0]a,b;
output [7:0] sum;
sum_3_2 x3 (a[3:0],b[3:0],sum[3:0]);
sum_3_2 x4 (a[7:4],b[7:4],sum[7:4]);
```

```
endmodule

//////////sum3to8/////////////
module sum_3_8(a,b,sum);
input [15:0]a,b;
output [15:0] sum;
sum_3_4 x5 (a[7:0],b[7:0],sum[7:0]);
sum_3_4 x6 (a[15:8],b[15:8],sum[15:8]);
endmodule

//////////sum3to16/////////////
module sum_3_16(a,b,sum);
input [31:0]a,b;
output [31:0] sum;
sum_3_8 x7 (a[15:0],b[15:0],sum[15:0]);
sum_3_8 x8 (a[31:16],b[31:16],sum[31:16]);
endmodule

//////////sum3to32/////////////
module sum_3_32(a,b,sum);
input [63:0]a,b;
output [63:0] sum;
sum_3_16 x9 (a[31:0],b[31:0],sum[31:0]);
sum_3_16 x10 (a[63:32],b[63:32],sum[63:32]);
endmodule

//////////sum3to64/////////////
module sum_3_64(a,b,sum);
input [127:0]a,b;
output [127:0] sum;
sum_3_32 x11 (a[63:0],b[63:0],sum[63:0]);
sum_3_32 x12 (a[127:64],b[127:64],sum[127:64]);
endmodule

//////////sum3to97/////////////
module sum_3_97(a,b,sum);
input [193:0]a,b;
output  [193:0] sum;
sum_3_64 x13 (a[127:0],b[127:0],sum[127:0]);
sum_3_32 x14(a[191:128],b[191:128],sum[191:128]);
sum_3_1  x15(a[193:192],b[193:192],sum[193:192]);
```

```
endmodule
```

A.1.2 Multiplication over $\mathbf{F}_3^m$

```
//////////product3to1/////////////
module prod_3_1(a,b,prod);
input [1:0]a,b;
output [1:0] prod;
assign prod[0]=((a[0] & ~a[1] & b[0] & ~b[1] ) || ( ~a[0] & a[1]
&~b[0] & b[1] ) );
assign prod[1]=((a[0] & ~a[1] & ~b[0] & b[1] ) || ( ~a[0] & a[1]
& b[0] & ~b[1] ) );
endmodule

//////////product3to97/////////////
module Prod_3_97(a,b,prod );
input [193:0]a;
input [1:0]b;
output [193:0] prod;
prod_3_1 a1(a[1:0],b,prod[1:0]);
prod_3_1 a2(a[3:2],b,prod[3:2]);
prod_3_1 a3(a[5:4],b,prod[5:4]);
prod_3_1 a4(a[7:6],b,prod[7:6]);
prod_3_1 a5(a[9:8],b,prod[9:8]);
prod_3_1 a6(a[11:10],b,prod[11:10]);
prod_3_1 a7(a[13:12],b,prod[13:12]);
prod_3_1 a8(a[15:14],b,prod[15:14]);
prod_3_1 a9(a[17:16],b,prod[17:16]);
prod_3_1 a10(a[19:18],b,prod[19:18]);
prod_3_1 a11(a[21:20],b,prod[21:20]);
prod_3_1 a12(a[23:22],b,prod[23:22]);
prod_3_1 a13(a[25:24],b,prod[25:24]);
prod_3_1 a14(a[27:26],b,prod[27:26]);
prod_3_1 a15(a[29:28],b,prod[29:28]);
prod_3_1 a16(a[31:30],b,prod[31:30]);
prod_3_1 a17(a[33:32],b,prod[33:32]);
prod_3_1 a18(a[35:34],b,prod[35:34]);
prod_3_1 a19(a[37:36],b,prod[37:36]);
prod_3_1 a20(a[39:38],b,prod[39:38]);
```

```
prod_3_1 a21(a[41:40],b,prod[41:40]);
prod_3_1 a22(a[43:42],b,prod[43:42]);
prod_3_1 a23(a[45:44],b,prod[45:44]);
prod_3_1 a24(a[47:46],b,prod[47:46]);
prod_3_1 a25(a[49:48],b,prod[49:48]);
prod_3_1 a26(a[51:50],b,prod[51:50]);
prod_3_1 a27(a[53:52],b,prod[53:52]);
prod_3_1 a28(a[55:54],b,prod[55:54]);
prod_3_1 a29(a[57:56],b,prod[57:56]);
prod_3_1 a30(a[59:58],b,prod[59:58]);
prod_3_1 a31(a[61:60],b,prod[61:60]);
prod_3_1 a32(a[63:62],b,prod[63:62]);
prod_3_1 a33(a[65:64],b,prod[65:64]);
prod_3_1 a34(a[67:66],b,prod[67:66]);
prod_3_1 a35(a[69:68],b,prod[69:68]);
prod_3_1 a36(a[71:70],b,prod[71:70]);
prod_3_1 a37(a[73:72],b,prod[73:72]);
prod_3_1 a38(a[75:74],b,prod[75:74]);
prod_3_1 a39(a[77:76],b,prod[77:76]);
prod_3_1 a40(a[79:78],b,prod[79:78]);
prod_3_1 a41(a[81:80],b,prod[81:80]);
prod_3_1 a42(a[83:82],b,prod[83:82]);
prod_3_1 a43(a[85:84],b,prod[85:84]);
prod_3_1 a44(a[87:86],b,prod[87:86]);
prod_3_1 a45(a[89:88],b,prod[89:88]);
prod_3_1 a46(a[91:90],b,prod[91:90]);
prod_3_1 a47(a[93:92],b,prod[93:92]);
prod_3_1 a48(a[95:94],b,prod[95:94]);
prod_3_1 a49(a[97:96],b,prod[97:96]);
prod_3_1 a50(a[99:98],b,prod[99:98]);
prod_3_1 a51(a[101:100],b,prod[101:100]);
prod_3_1 a52(a[103:102],b,prod[103:102]);
prod_3_1 a53(a[105:104],b,prod[105:104]);
prod_3_1 a54(a[107:106],b,prod[107:106]);
prod_3_1 a55(a[109:108],b,prod[109:108]);
prod_3_1 a56(a[111:110],b,prod[111:110]);
prod_3_1 a57(a[113:112],b,prod[113:112]);
prod_3_1 a58(a[115:114],b,prod[115:114]);
prod_3_1 a59(a[117:116],b,prod[117:116]);
prod_3_1 a60(a[119:118],b,prod[119:118]);
prod_3_1 a61(a[121:120],b,prod[121:120]);
```

```
prod_3_1 a62(a[123:122],b,prod[123:122]);
prod_3_1 a63(a[125:124],b,prod[125:124]);
prod_3_1 a64(a[127:126],b,prod[127:126]);
prod_3_1 a65(a[129:128],b,prod[129:128]);
prod_3_1 a66(a[131:130],b,prod[131:130]);
prod_3_1 a67(a[133:132],b,prod[133:132]);
prod_3_1 a68(a[135:134],b,prod[135:134]);
prod_3_1 a69(a[137:136],b,prod[137:136]);
prod_3_1 a70(a[139:138],b,prod[139:138]);
prod_3_1 a71(a[141:140],b,prod[141:140]);
prod_3_1 a72(a[143:142],b,prod[143:142]);
prod_3_1 a73(a[145:144],b,prod[145:144]);
prod_3_1 a74(a[147:146],b,prod[147:146]);
prod_3_1 a75(a[149:148],b,prod[149:148]);
prod_3_1 a76(a[151:150],b,prod[151:150]);
prod_3_1 a77(a[153:152],b,prod[153:152]);
prod_3_1 a78(a[155:154],b,prod[155:154]);
prod_3_1 a79(a[157:156],b,prod[157:156]);
prod_3_1 a80(a[159:158],b,prod[159:158]);
prod_3_1 a81(a[161:160],b,prod[161:160]);
prod_3_1 a82(a[163:162],b,prod[163:162]);
prod_3_1 a83(a[165:164],b,prod[165:164]);
prod_3_1 a84(a[167:166],b,prod[167:166]);
prod_3_1 a85(a[169:168],b,prod[169:168]);
prod_3_1 a86(a[171:170],b,prod[171:170]);
prod_3_1 a87(a[173:172],b,prod[173:172]);
prod_3_1 a88(a[175:174],b,prod[175:174]);
prod_3_1 a89(a[177:176],b,prod[177:176]);
prod_3_1 a90(a[179:178],b,prod[179:178]);
prod_3_1 a91(a[181:180],b,prod[181:180]);
prod_3_1 a92(a[183:182],b,prod[183:182]);
prod_3_1 a93(a[185:184],b,prod[185:184]);
prod_3_1 a94(a[187:186],b,prod[187:186]);
prod_3_1 a95(a[189:188],b,prod[189:188]);
prod_3_1 a96(a[191:190],b,prod[191:190]);
prod_3_1 a97(a[193:192],b,prod[193:192]);
endmodule

//////////product3to97 with shift x////////////
module prod_x_3_97(a,b,prod );
input [193:0]a;
```

```
input [1:0]b;
output [193:0]prod;
wire[195:0] shift_reg;
wire [193:0] prod1;
//wire [1:0]red1;
assign shift_reg =prod1<<2;
assign prod[23:2]=shift_reg[23:2];
assign prod[193:26]=shift_reg[193:26];
Prod_3_97 a11(a,b,prod1);
sum_3_1 a1(shift_reg[1:0],shift_reg[195:194],prod[1:0]);
diff_3_1 a24(shift_reg[25:24],shift_reg[195:194],prod[25:24]);
endmodule

//////////product3to97 with shift x^2/////////////
module prod_x2_3_97(a,b,prod );
input [193:0]a;
input [1:0]b;
output [193:0]prod;
wire [193:0] prod1;
wire [197:0] shift_reg;
//wire [3:0] red1;
assign shift_reg=prod1<<4;
assign prod[23:4]=shift_reg[23:4];
assign prod[193:28]=shift_reg[193:28];
Prod_3_97 a11(a,b,prod1);
//prod_3_1 z1(2'b10,shift_reg[195:194],red1[1:0]);
//diff_3_1 a1(shift_reg[1:0],red1[1:0],prod[1:0]);
//prod_3_1 z3(2'b10,shift_reg[197:196],red1[3:2]);
//diff_3_1 a3(shift_reg[3:2],red1[3:2],prod[3:2]);
sum_3_1 a1(shift_reg[1:0],shift_reg[195:194],prod[1:0]);
sum_3_1 a2(shift_reg[3:2],shift_reg[197:196],prod[3:2]);
diff_3_1 a24(shift_reg[25:24],shift_reg[195:194],prod[25:24]);
diff_3_1 a27(shift_reg[27:26],shift_reg[197:196],prod[27:26]);
//prod_x_3_97 a2(prod1,2'b01,irr_poly,clk,prod);
endmodule

//////////product3to97 with shift x^3/////////////
module prod_x3shift_3_97(a,prod );
input [193:0]a;
output [193:0]prod;
wire [199:0] shift_reg;
```

```
wire [5:0] red1;
assign shift_reg=a<<6;
assign prod[23:6]=shift_reg[23:6];
assign prod[193:30]=shift_reg[193:30];
prod_3_1 z1(2'b10,shift_reg[195:194],red1[1:0]);
diff_3_1 a1(shift_reg[1:0],red1[1:0],prod[1:0]);
prod_3_1 z3(2'b10,shift_reg[197:196],red1[3:2]);
diff_3_1 a3(shift_reg[3:2],red1[3:2],prod[3:2]);
prod_3_1 z5(2'b10,shift_reg[199:198],red1[5:4]);
diff_3_1 a5(shift_reg[5:4],red1[5:4],prod[5:4]);
diff_3_1 a24(shift_reg[25:24],shift_reg[195:194],prod[25:24]);
diff_3_1 a27(shift_reg[27:26],shift_reg[197:196],prod[27:26]);
diff_3_1 a30(shift_reg[29:28],shift_reg[199:198],prod[29:28]);
//prod_x2_3_97 a1(a,b,irr_poly,clk,prod1);
//prod_x_3_97 a2(prod1,2'b01,irr_poly,clk,prod);
endmodule

//////////Final multiplication code which calculate
multiplication in 33 clock cycles/////////////
module fin_mul_red_3_97(a,b,clk,reset,prod );
input [193:0]a;
input [193:0]b;
input clk,reset;
output [193:0]prod;
reg en=1,en2=1;
reg [5:0]i;
reg [191:0]shift_reg;
reg [193:0] px;
wire [193:0]prod1,prod2,prod3,prod4,prod5;
wire [193:0] sum1,sum2;
wire [5:0]bb;
assign bb=shift_reg[191:186];
//reg [5:0] count1;
Prod_3_97 a1(a,b[193:192],prod5[193:0]);
Prod_3_97 a11(a,bb[1:0],prod1);
prod_x_3_97 a2(a,bb[3:2],prod2 );
prod_x2_3_97 a3(a,bb[5:4],prod3 );
prod_x3shift_3_97 a4(px,prod4 );
sum_3_97 a5(prod1,prod2,sum1);
sum_3_97 a6(sum1,prod3,sum2);
sum_3_97 a7(sum2,prod4,prod);
```

```
always@ (clk)//( clk  )//or reset) //clk & en1
begin
 if(reset)
 begin
 i<=6'd32;
 en<=1'b1;
 shift_reg<=192'h0;
 px<=194'b0;
 //count1<=6'd0;
 en2<=1'b1;
 end

 else
   begin
  if( en2==1'b1)
       begin
     //count1<=count1+1'b1;
         if( en==1'b1)
 begin
  px<=prod5;
  en<=1'b0;
  shift_reg<=b[191:0];
 end
         else
begin
if(i>1)
  begin
shift_reg<=shift_reg<<6;
px<=prod;
i<=i-1;
end
else
begin
en2<=1'd0;
i<=6'd0;
end
            end
  end
else
en2<=en2;
```

```
end
end
endmodule
```

A.1.3 Multiplication over $\mathbf{F}_3^{6m}$

```
//////////////////sum3to2m///////////
module sum_3_2m(a1,a2,b1,b2,sum1,sum2);
input [193:0] a1,a2,b1,b2;
output [193:0] sum1,sum2;
sum_3_97 c1(a1,b1,sum1);
sum_3_97 c2(a2,b2,sum2);
endmodule

//////////////////sum3to2m///////////
module diff_3_2m(a1,a2,b1,b2,diff1,diff2);
input [193:0] a1,a2,b1,b2;
output [193:0] diff1,diff2;
diff_3_97 c1(a1,b1,diff1);
diff_3_97 c2(a2,b2,diff2);
endmodule

//////////////////mul3to2m///////////
module mul_3_2m(a1,a2,b1,b2,clk,reset,prod1,prod2);
input [193:0] a1,a2,b1,b2;
input clk,reset;
output [193:0] prod1,prod2;
wire  [193:0] temp1,temp2,temp3,sum1,sum2,sum3;
reg [5:0] count=0;
fin_mul_red_3_97 c1(a1,b1,clk,reset,temp1 );
fin_mul_red_3_97 c2(a2,b2,clk,reset,temp2 );
sum_3_97 c3(a1,a2,sum1);
sum_3_97 c4(b1,b2,sum2);
fin_mul_red_3_97 c25(sum1,sum2,clk,reset,temp3 );
diff_3_97 c6(temp1,temp2,prod1);
sum_3_97 c7(temp1,temp2,sum3);
diff_3_97 c8(temp3,sum3,prod2);
always @(  clk  or reset)  //clk &
if(reset)
count<=0;
else
```

```
begin
count<=count+1;
if(count==7)
count<=0;
end
endmodule

/////////////////mul3to6m///////////
module mul_3_6m(a1,a2,a3,a4,a5,a6,b1,b2,b3,b4,b5,b6,clk,reset,
prod1,prod2,prod3,prod4,prod5,prod6 );
input [193:0] a1,a2,a3,a4,a5,a6,b1,b2,b3,b4,b5,b6;
input clk,reset;
output [193:0] prod1,prod2,prod3,prod4,prod5,prod6;
wire [193:0] temp1,temp2,temp3,temp4,temp5,temp6,temp7,
temp8,temp9,temp10;
wire [193:0] aux11,aux12,aux21,aux22,aux31,aux32,auxb1,auxb2,aux41,
auxb3,auxb4,aux42,aux51,aux52,aux61,aux62,aux71,aux72,aux81,aux82,
aux91,aux92,aux101,aux102;
reg [7:0] count=0;
mul_3_2m k1(a1,a2,b1,b2,clk,reset,temp1,temp2);
mul_3_2m k2(a3,a4,b3,b4,clk,reset,temp3,temp4);
mul_3_2m k3(a5,a6,b5,b6,clk,reset,temp5,temp6);
sum_3_2m k4(a5,a6,a3,a4,aux11,aux12);
sum_3_2m k5(b5,b6,b3,b4,aux21,aux22);
mul_3_2m k6(aux11,aux12,aux21,aux22,clk,reset,temp7,temp8);
///////////////first 2 components///////////
sum_3_2m k7(temp3,temp4,temp5,temp6,aux31,aux32);
sum_3_2m k8(temp7,temp8,temp1,temp2,aux41,aux42);
diff_3_2m k9(aux41,aux42,aux31,aux32,prod1,prod2);
//////////////3rd and 4th component///////////////
mul_3_2m k31(auxb1,auxb2,auxb3,auxb4,clk,reset,temp9,temp10);
sum_3_2m k41(a1,a2,a3,a4,auxb1,auxb2);
sum_3_2m k42(b1,b2,b3,b4,auxb3,auxb4);

sum_3_2m k10(temp9,temp10,temp7,temp8,aux51,aux52);
sum_3_2m k11(aux51,aux52,temp3,temp4,aux61,aux62);
diff_3_2m k12(aux61,aux62,temp1,temp2,prod3,prod4);
//////////////last 2 cofficients/////////////
mul_3_2m k21(aux71,aux72,aux81,aux82,clk,reset,aux91,aux92);
sum_3_2m k22(a1,a2,a5,a6,aux71,aux72);
sum_3_2m k23(b1,b2,b5,b6,aux81,aux82);
```

```
sum_3_2m k16(aux91,aux92,temp3,temp4,aux101,aux102);
diff_3_2m k17(aux101,aux102,temp1,temp2,prod5,prod6);
always@( clk  or reset)
if(reset)
count<=0;
else
begin
count<=count+1;
  if(count==7)
   count<=0;
end
endmodule
```

A.1.4 Cube over $\mathbf{F}_3^m$

```
module cube_3_m(a,clk,out1 );
input [193:0] a;
input clk;
output  reg[193:0]out1;
wire [193:0] out;
wire [63:0] temp1;
assign out[5:4]=a[67:66];
assign out[11:10]=a[69:68];
assign out[17:16]=a[71:70];
assign out[23:22]=a[73:72];
assign out[73:72]=a[25:24];
assign out[79:78]=a[27:26];
assign out[85:84]=a[29:28];
assign out[91:90]=a[31:30];
assign out[97:96]=a[33:32];
assign out[103:102]=a[35:34];
assign out[109:108]=a[37:36];
assign out[115:114]=a[39:38];
assign out[121:120]=a[41:40];
assign out[127:126]=a[43:42];
assign out[133:132]=a[45:44];
assign out[139:138]=a[47:46];
assign out[145:144]=a[49:48];
assign out[151:150]=a[51:50];
assign out[157:156]=a[53:52];
assign out[163:162]=a[55:54];
```

```
assign out[169:168]=a[57:56];
assign out[175:174]=a[59:58];
assign out[181:180]=a[61:60];
assign out[187:186]=a[63:62];
assign out[193:192]=a[65:64];
sum_3_1 a1(a[1:0],a[179:178],temp1[1:0]);
sum_3_1 a2(a[187:186],temp1[1:0],out[1:0]);
diff_3_1 a3(a[131:130],a[123:122],out[3:2]);
sum_3_1 a4(a[3:2],a[181:180],temp1[3:2]);
sum_3_1 a5(a[189:188],temp1[3:2],out[7:6]);
diff_3_1 a6(a[133:132],a[125:124],out[9:8]);
sum_3_1 a7(a[5:4],a[183:182],temp1[5:4]);
sum_3_1 a8(a[191:190],temp1[5:4],out[13:12]);
diff_3_1 a9(a[135:134],a[127:126],out[15:14]);
sum_3_1 a10(a[7:6],a[185:184],temp1[7:6]);
sum_3_1 a11(a[193:192],temp1[7:6],out[19:18]);
diff_3_1 a12(a[137:136],a[129:128],out[21:20]);
diff_3_1 a13(a[9:8],a[179:178],out[25:24]);
sum_3_1 a14(a[123:122],a[171:170],temp1[9:8]);
sum_3_1 a15(a[139:138],temp1[9:8],out[27:26]);
diff_3_1 a16(a[75:74],a[67:66],out[29:28]);
diff_3_1 a17(a[11:10],a[181:180],out[31:30]);
sum_3_1 a18(a[125:124],a[133:132],temp1[11:10]);
sum_3_1 a19(a[141:140],temp1[11:10],out[33:32]);
diff_3_1 a20(a[77:76],a[69:68],out[35:34]);
diff_3_1 a21(a[13:12],a[183:182],out[37:36]);
sum_3_1 a22(a[127:126],a[135:134],temp1[13:12]);
sum_3_1 a23(a[143:142],temp1[13:12],out[39:38]);
diff_3_1 a24(a[79:78],a[71:70],out[41:40]);
diff_3_1 a25(a[15:14],a[185:184],out[43:42]);
sum_3_1 a26(a[129:128],a[137:136],temp1[15:14]);
sum_3_1 a27(a[145:144],temp1[15:14],out[45:44]);
diff_3_1 a28(a[81:80],a[73:72],out[47:46]);
diff_3_1 a29(a[17:16],a[187:186],out[49:48]);
sum_3_1 a30(a[131:130],a[139:138],temp1[17:16]);
sum_3_1 a31(a[147:146],temp1[17:16],out[51:50]);
diff_3_1 a32(a[83:82],a[75:74],out[53:52]);
diff_3_1 a33(a[19:18],a[189:188],out[55:54]);
sum_3_1 a34(a[133:132],a[141:140],temp1[19:18]);
sum_3_1 a35(a[149:148],temp1[19:18],out[57:56]);
diff_3_1 a36(a[85:84],a[77:76],out[59:58]);
```

```
diff_3_1 a37(a[21:20],a[191:190],out[61:60]);
sum_3_1 a38(a[135:134],a[143:142],temp1[21:20]);
sum_3_1 a39(a[151:150],temp1[21:20],out[63:62]);
diff_3_1 a40(a[87:86],a[79:78],out[65:64]);
diff_3_1 a41(a[23:22],a[193:192],out[67:66]);
sum_3_1 a42(a[137:136],a[145:144],temp1[23:22]);
sum_3_1 a43(a[153:152],temp1[23:22],out[69:68]);
diff_3_1 a44(a[89:88],a[81:80],out[71:70]);
sum_3_1 a45(a[139:138],a[147:146],temp1[25:24]);
sum_3_1 a46(a[155:154],temp1[25:24],out[75:74]);
diff_3_1 a47(a[91:90],a[83:82],out[77:76]);
sum_3_1 a48(a[141:140],a[149:148],temp1[27:26]);
sum_3_1 a49(a[157:156],temp1[27:26],out[81:80]);
diff_3_1 a50(a[93:92],a[85:84],out[83:82]);
sum_3_1 a51(a[143:142],a[151:150],temp1[29:28]);
sum_3_1 a52(a[159:158],temp1[29:28],out[87:86]);
diff_3_1 a53(a[95:94],a[87:86],out[89:88]);
sum_3_1 a54(a[145:144],a[153:152],temp1[31:30]);
sum_3_1 a55(a[161:160],temp1[31:30],out[93:92]);
diff_3_1 a56(a[97:96],a[89:88],out[95:94]);
sum_3_1 a57(a[147:146],a[155:154],temp1[33:32]);
sum_3_1 a58(a[163:162],temp1[33:32],out[99:98]);
diff_3_1 a59(a[99:98],a[91:90],out[101:100]);
sum_3_1 a60(a[149:148],a[157:156],temp1[35:34]);
sum_3_1 a61(a[165:164],temp1[35:34],out[105:104]);
diff_3_1 a62(a[101:100],a[93:92],out[107:106]);
sum_3_1 a63(a[151:150],a[159:158],temp1[37:36]);
sum_3_1 a64(a[167:166],temp1[37:36],out[111:110]);
diff_3_1 a65(a[103:102],a[95:94],out[113:112]);
sum_3_1 a66(a[153:152],a[161:160],temp1[39:38]);
sum_3_1 a67(a[169:168],temp1[39:38],out[117:116]);
diff_3_1 a68(a[105:104],a[97:96],out[119:118]);
sum_3_1 a69(a[155:154],a[163:162],temp1[41:40]);
sum_3_1 a70(a[171:170],temp1[41:40],out[123:122]);
diff_3_1 a71(a[107:106],a[99:98],out[125:124]);
sum_3_1 a72(a[157:156],a[165:164],temp1[43:42]);
sum_3_1 a73(a[173:172],temp1[43:42],out[129:128]);
diff_3_1 a74(a[109:108],a[101:100],out[131:130]);
sum_3_1 a75(a[159:158],a[167:166],temp1[45:44]);
sum_3_1 a76(a[175:174],temp1[45:44],out[135:134]);
diff_3_1 a77(a[111:110],a[103:102],out[137:136]);
```

```
sum_3_1 a78(a[161:160],a[169:168],temp1[47:46]);
sum_3_1 a79(a[177:176],temp1[47:46],out[141:140]);
diff_3_1 a80(a[113:112],a[105:104],out[143:142]);
sum_3_1 a81(a[163:162],a[171:170],temp1[49:48]);
sum_3_1 a82(a[179:178],temp1[49:48],out[147:146]);
diff_3_1 a83(a[115:114],a[107:106],out[149:148]);
sum_3_1 a84(a[165:164],a[173:172],temp1[51:50]);
sum_3_1 a85(a[181:180],temp1[51:50],out[153:152]);
diff_3_1 a86(a[117:116],a[109:108],out[155:154]);
sum_3_1 a87(a[167:166],a[175:174],temp1[53:52]);
sum_3_1 a88(a[183:182],temp1[53:52],out[159:158]);
diff_3_1 a89(a[119:118],a[111:110],out[161:160]);
sum_3_1 a90(a[169:168],a[177:176],temp1[55:54]);
sum_3_1 a91(a[185:184],temp1[55:54],out[165:164]);
diff_3_1 a92(a[121:120],a[113:112],out[167:166]);
sum_3_1 a93(a[171:170],a[179:178],temp1[57:56]);
sum_3_1 a94(a[187:186],temp1[57:56],out[171:170]);
diff_3_1 a95(a[123:122],a[115:114],out[173:172]);
sum_3_1 a96(a[173:172],a[181:180],temp1[59:58]);
sum_3_1 a97(a[189:188],temp1[59:58],out[177:176]);
diff_3_1 a98(a[125:124],a[117:116],out[179:178]);
sum_3_1 a99(a[175:174],a[183:182],temp1[61:60]);
sum_3_1 a100(a[191:190],temp1[61:60],out[183:182]);
diff_3_1 a101(a[127:126],a[119:118],out[185:184]);
sum_3_1 a102(a[177:176],a[185:184],temp1[63:62]);
sum_3_1 a103(a[193:192],temp1[63:62],out[189:188]);
diff_3_1 a104(a[129:128],a[121:120],out[191:190]);
always@(clk)
//begin
//if (reset)
//begin
//out1<=0;
//end
//else
out1<=out;
//end
endmodule
```

A.1.5 Cube over $\mathbf{F}_3^{6m}$

```
module cube_3_6m(a1,a2,a3,a4,a5,a6,clk,cubee1,cubee2,cubee3,
```

```
cubee4,cubee5,cubee6 );
input [193:0] a1,a2,a3,a4,a5,a6;
input clk;
output reg [193:0] cubee1,cubee2,cubee3,cubee4,cubee5,cubee6;
wire [193:0] temp1,temp2,temp3,temp4,temp5,temp6,cube1,cube2,
cube3,cube4,cube5,cube6;
wire [193:0] sum1,sum2,sum3;
cube_3_m  b1(a1,clk,temp1 );
cube_3_m  b2(a2,clk,temp2 );
cube_3_m  b3(a3,clk,temp3 );
cube_3_m  b4(a4,clk,temp4 );
cube_3_m  b5(a5,clk,temp5 );
cube_3_m  b6(a6,clk,temp6 );
//////////////cube1/////////
sum_3_97 b7(temp1,temp3,sum1);
sum_3_97 b8(sum1,temp5,cube1);
//////////cube2/////////
sum_3_97 b9(temp2,temp4,sum2);
sum_3_97 b10(sum2,temp6,sum3);
Prod_3_97 b11(sum3,2'b10,cube2 );
//////////cube3/////////
diff_3_97 b12(temp3,temp5,cube3);
/////////cube4/////////////////
diff_3_97 b13(temp6,temp4,cube4);
//////////cube5///////////////
assign cube5=temp5;
///////////////cube6/////////
Prod_3_97 b14(temp6,2'b10,cube6 );
always@(posedge clk )

begin
cubee1<=cube1;cubee2<=cube2;cubee3<=cube3;
cubee4<=cube4;cubee5<=cube5;cubee6<=cube6;

end
endmodule
```

A.1.6 Final Pairing Top module

```
module pairing(px1,py1,qx1,qy1,reset,load,clk,pairing );
input [193:0]px1,py1,qx1,qy1;
```

```
input clk,reset,load;
output reg [193:0] pairing;
reg [9:0] count1,count2,count3,count4;
reg[19:0]count6=0;
reg [193:0]tem1=0,tem2=0,temmp1=0;
reg [193:0] init,ak1,ak2,ak3,ak4,ak5,ak6,ak7;
reg [5:0]state,loop;
reg [1:0] c,d,g;
wire [193:0] out1,sum1,prod1,prod2,prod3;
reg reset3,en1,en2,en11,en12,en5,reset4,en3;
reg [1163:0]R0=0,R1=0;
reg [193:0] a1=0,a2=0,a3=0,a4=0,a5=0,a6=0,b1=0,b2=0,b3=0,b4=0,b5=0,
b6=0,ad1=0,ad2=0,ad3=0,ad4=0,ad5=0,ad6=0;
wire[193:0] prodd1,prodd2,prodd3,prodd4,prodd5,prodd6,cube1,cube2,
cube3,cube4,cube5,cube6;
//reg [193:0] fin1=0,fin2=0,fin3=0,fin4=0,fin5=0,fin6;
reg [193:0] pow9qx,pow9qy,midpy,midqy=0,midqx=0;
wire [1:0] diff;
reg [193:0]px,py,qx,qy;
///////////////modules called//////////
cube_3_6m  asd(ad1,ad2,ad3,ad4,ad5,ad6,clk,cube1,cube2,cube3,cube4,
cube5,cube6 );
Prod_3_97 m1(init,c,out1);
sum_3_97  m2(ak1,ak2,sum1);
fin_mul_red_3_97 ab7(ak3,ak4,clk,reset3,prod1 );
fin_mul_red_3_97 ab8(ak5,ak6,clk,reset3,prod2 );
cube_3_m m3(ak7,clk,prod3 );
diff_3_1  m6(g,2'd1,diff);
mul_3_6m aa7(a1,a2,a3,a4,a5,a6,b1,b2,b3,b4,b5,b6,clk,reset4,prodd1,
prodd2,prodd3,prodd4,prodd5,prodd6 );
always@(clk & en3)
begin
count6<=count6+1;
end
always@(clk or reset)
begin
if(reset)
begin
//    count<=0;
 count1<=0;count2<=0;count3<=0; count4<=0;
//  count5<=0;
```

```
 //count6=0;
// state<=8;
 d<=0;reset3<=0;reset4<=0;
// reset5<=1;
 en1<=1;en2<=1;en3<=1;
//  en4<=1;
 en5<=1;en11<=1;en12<=1;
//  reset1<=1;
 loop<=0; R0<=0;R1<=0; pow9qx<=0;pow9qy<=0; midqx<=0;midqy<=0;
         midpy<=0;a1<=0; a2<=0; a3<=0;a4<=0;a5<=0; a6<=0; b1<=0; b2<=0;
 b3<=0; b4<=0; b5<=0;b6<=0;ad1<=0; ad2<=0;ad3<=0; ad4<=0; ad5<=0;
 ad6<=0; a2<=0; a3<=0; a4<=0; a5<=0; a6<=0; b1<=0; b2<=0; b3<=0;
 b4<=0; b5<=0; b6<=0; ad1<=0; ad2<=0; ad3<=0; ad4<=0; ad5<=0;
 ad6<=0;px<=0; py<=py1; qx<=qx1;qy<=qy1; state<=0;
end

else
  begin
   if(load)

  begin
  px<=px1;
  py<=py1;
  qx<=qx1;
  qy<=qy1;
  state<=0;
  end

else


 case(state)

6'd0:
begin
init<=py;
c<=2'd2;
ak1<=px;
ak2<=qx;
state<=6'd1;
d<=1;
```

```
reset3<=1;
end
 6'd1:
    begin
temmp1<=out1;
c<=2'd2;
ak1<=sum1;
ak2<=d;
state<=6'd2;
 end
6'd2:
    begin
      ak3<=out1;
ak4<=sum1;
reset3<=0;
state<=6'd3;
R0<=0;
    end
  6'd3:
    begin
 count1<=count1+1;
 if(count1==34)    //change
   begin
tem1<=prod1;
count1<=0;
state<=6'd4;
end
end
6'd4:
    begin
 count1<=count1+1;
 if(count1==0)
   begin
      init<=prod1;
c<=2;
      end
else if(count1==1)
begin
R0[193:0]<=out1;
R0[387:194]<=qy;
R0[581:388]<=temmp1;
```

```
R0[775:582]<=194'd0;
R0[969:776]<=194'd0;
R0[1163:970]<=194'd0;
count1<=0;
state<=6'd5;
// reset1<=1;
end
else
count1<=0;
     end
6'd5:
begin
if(loop<48)
  begin
count4<=count4+1;
case(count4)
10'd0:
 if(en1)
 begin
 ak1<=px;
 ak2<=qx;
//  ak7<=qx;
//  ak8<=qy;
//  reset1<=0;
 end
else
 begin
 ak1<=px;
 ak2<=pow9qx;
//  ak7<=pow9qx;
//  ak8<=pow9qy;
//  reset1<=0;
 end
  10'd1:
 begin
 ak1<=sum1;
 ak2<=d;
 reset3<=1;
 R1<=0;
  tem1<=0;
  en1<=0;
```

```
 end
10'd2:
   if(en2)
 begin
 tem1<=sum1;
 ak3<=sum1;
 ak4<=sum1;
 reset3<=0;
 ak5<=temmp1;
 ak6<=qy;
 end
 else
 begin
 tem1<=sum1;
 ak3<=sum1;
 ak4<=sum1;
 reset3<=0;
 ak5<=midpy;
 ak6<=pow9qy;
 end
10'd3:
   begin
      init<=tem1;  ///r0
c<=2'd2;
   end
10'd4:
   begin
tem2<=out1;////2ro
   en2<=0;
end
10'd5:
begin
if(en11)
ak7<=qx;
else
ak7<=pow9qx;
end
10'd6:
midqx<=prod3;
10'd7:
ak7<=midqx;
```

```
10'd8:
pow9qx<=prod3;

//10'd31:
//   begin
//   midqx<=prod3;
// midqy<=prod4;
// end
//10'd32:
//    reset1<=1;
//10'd33:
// begin
//     reset1<=0;
//   ak7<=midqx;
//   ak8<=midqy;
//   end

10'd35:     //mul cycles +3
     begin
  init<=prod1;   ///2r02
c<=2'd2;
//reset1<=1;
//tem3<=prod3;
  end
10'd36:
begin
R1[193:0]<=out1;
R1[387:194]<=prod2;
R1[581:388]<=tem2;
R1[775:582]<=194'd0;
R1[969:776]<=194'd2;
R1[1163:970]<=194'd0;
en2<=0;
en1<=0;
count4<=0;
state<=6;
reset4<=1;
en11<=0;
end
   default:
en2<=en2;
```

```
   endcase
end
else
begin
en3<=0;
pairing<= R0[193:0];
end
end
6'd6:
  begin
count2<=count2+1;
  case(count2)
  10'd0:
       begin
    a1<=R0[193:0];
 a2<=R0[387:194];
 a3<=R0[581:388];
 a4<=R0[775:582];
 a5<=R0[969:776];
 a6<=R0[1163:970];
 b1<=R1[193:0];
 b2<=R1[387:194];
 b3<=R1[581:388];
 b4<=R1[775:582];
 b5<=R1[969:776];
 b6<=R1[1163:970];
 reset4<=0;
  end
10'd1:
begin
if(en12)
ak7<=qy;
else
ak7<=pow9qy;
end
10'd2:
midqy<=prod3;
10'd3:
ak7<=midqy;
10'd4:
pow9qy<=prod3;
```

```
10'd5:
begin
 if(en5)
 begin
 init<=temmp1;
 c<=2;
 end
 else
 begin
 init<=midpy;
 c<=2'd2;
 end
end
10'd6:
   midpy<=out1;
10'd7:
begin
g<=d;
end
10'd8:
d<=diff;
10'd34:  //cycles +1
begin
state<=7;
      count2<=0;
// reset5=1;
en5<=0;
en12<=0;
     end
  default:
  en2<=en2;
 endcase
 end
6'd7:
   begin
count3<=count3+1;
case(count3)
10'd0:
 begin
ad1<=prodd1;
ad2<=prodd2;
```

```
ad3<=prodd3;
ad4<=prodd4;
ad5<=prodd5;
ad6<=prodd6;
// reset5=0;
R0<=0;
 end
10'd1:
     begin
      R0[193:0]<=cube1;
R0[387:194]<=cube2;
R0[581:388]<=cube3;
R0[775:582]<=cube4;
R0[969:776]<=cube5;
R0[1163:970]<=cube6;
//////////////////
state<=5;
      count3<=0;
count2<=0;
count4<=0;
en5<=0;
// en4<=0;
en1<=0;
en11<=0;
en12<=0;
en2<=0;
// reset1<=1;
loop<=loop+1;
//en3<=0;
     end
  endcase
end
default:
state<=66;
  endcase
end
end
endmodule
```