Verilog Implementation of Optimized Elliptic Curve Crypto Processor for FPGA platform and its Performance analysis

M. Jotheeshkumar
Department of ECE
PSG College of Technology

S. HemaChitra
Department of ECE
PSG College of Technology

Abstract—In this paper the design and implementation of an efficient high performance Elliptic Curve Crypto processor (ECCP) for an elliptic curve over the finite field GF(2^233) is made. ECC algorithm is implemented based on Vedic Mathematics. The speed of the system mainly depends on number of multiplications, multipliers and adders. To improve the speed of the system, modification of algorithm is proposed and Hybrid Karatsuba multiplier is used. Using this algorithm number of partial products in the multiplier architecture is reduced to half and thus it speeds up the operation. Effectively number of multipliers are required is reduced drastically. Further speed of the encryption decryption process is improved by changing the type of multiplier used in the system. The processor is synthesized for Xilinx FPGA.

Keywords— Binary Field, Elliptic Curve Cryptography, Karatsuba Multiplier

I. INTRODUCTION

In recent years there is a dramatic increase in communications over the wired and wireless networks. Everyday thousands of transactions take place over the world wide web. Several of these transactions have critical data which need to be confidential, transactions that need to be validated, and users authenticated. These requirements need a rugged security framework to be in force. To match the ever increasing requirement for speed in today’s applications, hardware acceleration of the cryptographic algorithms is a necessity. As a further challenge, the designs have to be robust against side channel attacks. Therefore, an efficient design and implementation of Encryption System is required.

This paper explores Elliptic curve cryptography so that high security can be achieved with small size keys. To improve the speed of the system, new algorithm is proposed and various types of multipliers are used. Using this algorithm number of partial products in the multiplier architecture is reduced to half and thus it speeds up the operation. Effectively no multipliers are required and number of adders required is reduced drastically. The most significant aspect of this project is the simple algorithm while final small sized multiplications is done using the proposed algorithm. The multiplier thus obtained has the best area and time product compared to reported literature. The coding is done in Verilog HDL and FPGA implementation using Xilinx Spartan 6 library.

II. ELLIPTIC CURVE CRYPTOGRAPHY

Elliptic curves have been studied for over hundred years and have been used to solve a diverse range of problems. For example, elliptic curves are used in proving Fermat’s last theorem, which states that $x^n + y^n = z^n$ has non zero integer solutions for $x, y, z$ when $n > 2$. The use of elliptic curves in public key cryptography was first proposed independently. Since then, there has been an abundance of research on the security of ECC. In the 1990’s ECC began to get accepted by several accredited organizations, and several security protocols based on ECC were standardized. The main advantage of ECC over conventional asymmetric crypto systems is the increased security offered with smaller key sizes. For example, a 256 bit key in ECC produces the same level of security as a 3072 bit RSA key. The smaller key sizes lead to compact implementations and increased performance. This makes ECC suited for low power resource constrained devices. An elliptic curve is the set of solutions $(x, y)$ to equation, together with the point at infinity (0). This equation is known as the Weierstra equation

$$y^2 + a_1 xy + a_3 y = x^3 + a_2 x^2 + a_4 x + a_6$$

For cryptography, the points on the elliptic curve are chosen from a large finite field. The set of points on the elliptic curve form a group under the addition rule. The point 0 is the identity element of the group. The operations on the elliptic curve, i.e. the group operations are point addition, point doubling and point inverse. Given a point $P = (x, y)$ on the elliptic curve, and a positive integer $n$, scalar multiplication is defined as

$$nP = P + P + \cdots + P \text{ (n times)}$$

The order of the point $P$ is the smallest positive integer $n$ such that $nP = 0$. The points $\{0, P, 2P, 3P, \cdots (n - 1)P\}$ form a group generated by $P$. The group is denoted as $\langle P \rangle$.

The security of ECC is provided by the elliptic curve discrete logarithm problem (ECDLP), which is defined as follows: Given a point $P$ on the elliptic curve and another point $Q \in \langle P \rangle$, determine an integer $k$ ($0 < k$) such that $Q = kP$. The difficulty of ECDLP is to calculate the value of the scalar $k$ given the points $P$ and $Q$. $k$ is called the discrete logarithm of $Q$ to the base $P$. $P$ is the generator of the elliptic curve and is called the basepoint.

III. ECC IMPLEMENTATION

The implementation of elliptic curve crypto systems...
constitutes a complex interdisciplinary research field involving mathematics, computer science, and electrical engineering. Elliptic curve crypto systems have a layered hierarchy as shown in Figure 1.

The crypto algorithm should not be the bottleneck on the application's performance. These implementations must also be highly flexible. Operating parameters such as algorithm constants, etc. should be reconfigurable. Reconfiguration can easily be done in software, however software implementations do not always scale to the performance demanded by the application. Such systems require to use dedicated hardware to speedup computations. When using such hardware accelerators, the clock cycles required, frequency of operation, and area are important design criteria. The clock cycles should be low and frequency high so that the overall latency of the hardware is less. The area is important because smaller area implies more frequency high so that the device's throughput increases.

IV. MATHEMATICAL BACKGROUND

Understanding Elliptic Curve Cryptography (ECC) requires a good understanding of the underlying mathematics. ECC relies heavily on abstract algebra for its construction. This chapter therefore starts with a brief overview of the primitive algebraic structures, namely groups, rings, and fields. The second part of this chapter is dedicated to the mathematics behind elliptic curves. In specific, elliptic curves over finite fields of the form \( \mathbb{F}_p \) are considered. The operations on this form of elliptic curve are discussed.

Definition: An elliptic curve \( E \) over the field \( \mathbb{F}_p \) is given by the simplified form of the Weierstraß equation mentioned below.

\[
y^2 + xy = x^3 + ax^2 + b
\]

with the coefficients \( a \) and \( b \) in \( \mathbb{F}_p \) and \( b \neq 0 \).

If \( b \neq 0 \), then the curve in above equation is a non-singular curve. A point on the curve is said to be singular if its partial derivatives vanish. The set of points on the elliptic curve along with a special point \( 0 \) called the point at infinity, form a group under addition. The identity element of the group is the point at infinity \( (0,0) \). The arithmetic operations permitted on the group are point inversion, point addition and point doubling which are described as follows.

Point Inversion: Let \( P = (x, y) \) be a point on the elliptic curve and \( P \neq 0 \). The inverse of this point \( -P \) is defined as \( (x, -y) \).

Point Addition: Let \( P = (x_1, y_1) \) and \( Q = (x_2, y_2) \) be two distinct points on the elliptic curve, then the sum of two points \( R = P + Q \) is given by the formula:

\[
\lambda = \frac{y_2 - y_1}{x_2 - x_1} \\
x_3 = \lambda^2 + x_1 + x_2 \\
y_3 = \lambda(x_1 + x_2) + y_1
\]

where \( \lambda = \frac{y_2 - y_1}{x_2 - x_1} \).

Point Doubling: Let \( P = (x, y) \) be a point on the curve and \( P \neq 0 \). Then the double of \( P \) is defined as \( 2P = (x_3, y_3) \) obtained by drawing a tangent to the curve at \( P \). The inverse of the point at which the tangent intersects the curve is the double of \( P \). The equation for computing \( 2P \) is given as

Algorithm 1: Double and Add algorithm for scalar multiplication

Input: Basepoint \( P = (px, py) \) and Scalar \( k = (k_{m-1}, k_{m-2}, \ldots, k_0) \), where \( k_{m-1} = 1 \)

Output: Point on the curve \( Q = kP \)

1. \( Q = P \)
2. For \( i = m \) to 2 do
3. \( Q = 2 \cdot Q \)
4. If \( k_i = 1 \) then
5. \( Q = Q + P \)
6. End
7. End

Table 1: Scalar Multiplication using Double and Add to find \( 22P \)

<table>
<thead>
<tr>
<th>( k )</th>
<th>Operation</th>
<th>( Q )</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Double only</td>
<td>2P</td>
</tr>
<tr>
<td>2</td>
<td>Double and Add</td>
<td>5P</td>
</tr>
<tr>
<td>1</td>
<td>Double and Add</td>
<td>11P</td>
</tr>
<tr>
<td>0</td>
<td>Double only</td>
<td>22P</td>
</tr>
</tbody>
</table>

The fundamental algorithm for ECC is the scalar multiplication. The basic double and add algorithm to perform scalar multiplication is shown in Algorithm 3.1. The input to the algorithm is a basepoint \( P \) and a \( m \) bit scalar \( k \). The result is the scalar product \( kP \).

Projective Coordinate Representation

The complexity of a finite field inversion is typically eight times that of a finite field multiplier in the same field. Therefore, there is a huge motivation for an alternate point representation which would require lesser inversions. The two point coordinate system \( (x, y) \) is used in affine representation. It has been shown that each affine point on the elliptic curve has a one to one correspondence with a unique equivalence class in which each point is represented by three coordinates \( (X, Y, Z) \). The three point coordinate system is called the projective representation [11]. In the projective representation, inversions are replaced by multiplications. The projective form of the WeierstraaBequation can be obtained by replacing \( x \) with \( X/Z \) and \( y \) by \( Y/Z \). There are several projective coordinates.
systems proposed. The most commonly used projective coordinate system is the standard where \( c = 1 \) and \( d = 1 \), the Jacobian with \( c = 2 \) and \( d = 3 \) and the Lopez-Dahab (LD) coordinates which has \( c = 1 \) and \( d = 2 \). The LD coordinates system [30] allows point addition using mixed coordinates. i.e. one point in affine while the other in projective. Replacing \( X \) by \( X/Z \) and \( y \) by \( Y/Z^2 \) in Equation 3.5 results in the LD projective form of the Weierstraf equation.

\[
Y_2 + X \cdot Y \cdot Z = X_3 + a \cdot X_2 \cdot Z_2 + b Z_4
\]

Let \( P = (X_1, Y_1, Z_1) \) be an LD projective point on the elliptic curve, then the inverse of point \( P \) is given by \( -P = (X_1, Z_1, Y_1) \). Also, \( P + (-P) = 0 \), where \( 0 \) is the point at infinity. In LD projective coordinates \( 0 \) is represented as \((1, 0, 0)\). The equation for doubling the point \( P \) in LD projective coordinates [30] results in the point \( 2P = (X_3, Y_3, Z_3) \). This is given by the following equation.

\[
\begin{align*}
Z_1 &= X_2 \cdot Z_2 \\
X_3 &= X_4 + b \cdot Z_4 \\
Y_3 &= b \cdot Z_1^2 + X_3 \cdot (a \cdot Z_3 + Y_1^2 + b \cdot Z_1^3)
\end{align*}
\]

The equations for doubling require 5 finite field multiplications and zero inversions. The equation in LD coordinates for adding the affine point \( Q = (X_2, Y_2) \) to \( P \), where \( Q = -P \), The resulting point is

\[
\begin{align*}
P + Q &= (X_1, Y_1, Z_1) \cdot A = Y_2 \cdot Z_2^2 + Y_1 \\
B &= X_2 \cdot Z_2 + X_1 \\
C &= 2b \cdot B \\
D &= B^2 \cdot (C + a \cdot Z_1^2) \\
Z_3 &= C^2 \\
E &= A \cdot C \\
X_3 &= A^2 + D + E \cdot F = X_3 + X_2 \cdot Z_3 \\
G &= (x_2 + y_2) \cdot Z^2 \\
Y_3 &= (E + Z_3) \cdot F + G
\end{align*}
\]

Point addition in LD coordinates now require 9 finite field multiplications and zero inversions. For an \( m \) bit scalar with approximately half the bits one, the running time expected is given in above. One inversion and 2 multiplications are required at the end to convert the result from projective coordinates back into affine.

V. ARCHITECTING AN EFFICIENT FINITEFIELD MULTIPLIER ON FPGA PLATFORMS

The finite field multiplier forms the most important component in the elliptic curve cryptography processor (ECCP). It occupies the most area on the device and also has the longest latency. The performance of the ECCP is affected most by the multiplier. Finite field multiplication of two elements in the field \( GF(2^m) \) is defined as

\[
C(x) = (A(x) \cdot B(x) \mod P(x)
\]

where \( C(x), A(x), \) and \( B(x) \) are in \( GF(2^m) \) and \( P(x) \) is the irreducible polynomial that generates the field \( GF(2^m) \). Implementing the multiplication requires two steps. First, the polynomial product \( C(x) = A(x) \cdot B(x) \) is determined, then the modulo operation is done on \( C(x) \). This chapter deals with polynomial multiplication. The basic recursive Karatsuba multiplier cannot be applied directly to ECC because the binary extension fields used in standards such as [14] have a prime degree. There have been several published works which implement a modified Karatsuba algorithm for use in elliptic curves. There are two main design approaches followed. The first approach is a sequential circuit having less hardware and latency but requiring several clock cycles to produce the result. Generally at every clock cycle the outputs are fed back into the circuit thus reusing the hardware. The advantage of this approach is that it can be pipelined. Examples of implementations following this approach can be found in [48-51]. The second approach is a combinational circuit having large area and delay but is capable of generating the result in one clock cycle.

VI. ANALYZING KARATSUBA MULTIPLIERS ON FPGA PLATFORMS

In this section we discuss the mapping of various Karatsuba algorithms on an FPGA. We estimate the amount of FPGA resources that is required for the implementations. Recursive Karatsuba Multiplier: In an \( m = 2^l \) bit recursive Karatsuba multiplier the basic Karatsuba algorithm of [12] is applied recursively. Each recursion reduces the size of the input by half while tripling the number of multiplications required. At each recursion, except the final, only XOR operations are involved. Let \( n = 2^{(\log_2 m)} \) the size of the inputs (A and B) for the \( k \)th recursion of the \( m \) bit multiplier. There are \( 3^s \) such \( n \) bit multipliers required. The A and B inputs are split into two: \( A_0, A_1 \) and \( B_0, B_1 \) respectively with each term having \( n/2 \) bits. \( n/2 \) two input XORs are required for the computation of \( A_0 \cdot A_1 + B_0 \cdot B_1 \) respectively. Each two input XOR requires one LUT on the FPGA, thus in total there are \( n \) LUTs required. Determining the output bits \( n - 2 \) to \( n/2 \) and \( n/2 - 2 \) to \( n \) requires \( 3(n/2 - 1) \) two input XORs each. The output bit \( n - 1 \) requires 2 two input XORs. In all \( (3n - 4) \) two input XORs are required to add the partial products. The number of LUTs required to combine the partial products is much lower. This is because each LUT implements a four input XOR. Each output bit \( n/2 \) to \( 3n/2 - 2 \) requires one LUT, therefore \( n - 1 \) LUTs are required for the purpose. In total, \( 2n - 1 \) LUTs are required for each recursion on the FPGA. The final recursion has \( 3(\log_2 m - 1) \) two bit Karatsuba multipliers. The equation for the two bit Karatsuba multiplier is shown in Equation given below

\[
\begin{align*}
C_0 &= A_0 B_0 \\
C_1 &= A_0 B_0 + A_1 B_1 + (A_0 + A_1)(B_0 + B_1) \\
C_2 &= A_1 B_1
\end{align*}
\]
This requires three LUTs on the FPGA: one for each of the output bits \((C_0, C_1, C_2)\).

**Algorithm 2:** `gkmul` (General Karatsuba Multiplier)

**Input:** \(A, B\) are multiplicands of \(m\) bits

**Output:** \(C\) of length 2\(m\) - 1 bits

- Define: \(M_x \rightarrow A_x B_x\)
- Define: \(M(x,y) \rightarrow (A_x + A_y)(B_x + B_y)\)

```
begin
  for i=0 to m-2 do
    \(C_i = C_{2m-2-i} = 0\)
  for j=0 to i/2 do
    if i=2j then
      \(C_i = C_i + M_j\)
    else
      \(C_i = C_i + M_j + M_{i-j} + M_{j+i-j}\)
    \(C_{2m-2-i} = C_{2m-2-i} + M_{m-1-j}\)
    \(+ M_{m-1-i+j} + M_{(m-1-j,m-1-i+j)}\)
  end
end
C_m-1 = 0
```

In this section we present our proposed multiplier called the hybrid Karatsuba multiplier. We show how we combine techniques to maximize utilization of LUTs resulting in minimum area.

**Algorithm 3:** `hmul` (Hybrid Karatsuba Multiplier)

**Input:** The multiplicands \(A, B\) and their length \(m\)

**Output:** \(C\) of length 2\(m\) - 1 bits

- if \(m < 29\) then return `gkmul(A, B, m)`
- else
  \(l = \text{ln} / 2\)

```
A = A_{[m-1-\ldots-1]} + A_{[l-\ldots-0]}
B = B_{[m-1-\ldots-1]} + B_{[l-\ldots-0]}
C_p2 = \text{hmul}(A, B, l)
C_p3 = \text{hmul}(A_{[m-\ldots-1]}, B_{[m-\ldots-1]}, m - l)
return (C_p2 << 2l) + (C_p3 + C_p2 + C_p3) << l + C_p1
```

In our proposed hybrid Karatsuba multiplier shown in Algorithm 3, the \(m\) bit multiplicands are split into two parts when the number of bits is greater than or equal to the threshold 29. The higher term has \(\text{ln}/2\) bits while the lower term has \(\text{ln}/2\) bits. If the number of bits of the multiplicand is less than 29 the general Karatsuba algorithm is invoked. The general Karatsuba algorithm ensures maximum utilization of the LUTs for the smaller bit multiplications, while the simple Karatsuba algorithm ensures least gate count for the larger bit multiplications. For a 233 bit hybrid Karatsuba multiplier (Figure 3), the multiplicands are split into two terms with \(A_h\) and \(B_h\) of 116 bits and \(A_l\) and \(B_l\) of 117 bits. The 116 bit multiplication is implemented using three 58 bit multipliers, while the 117 bit multiplier is implemented using two 59 bit multipliers and a 58 bit multiplier. The 58 and 59 bit multiplications are implemented with 29 and 30 bit multipliers, the 29 and 30 bit multiplications are done using 14 and 15 bit general Karatsuba multiplier. The number of recursions in the hybrid Karatsuba multiplier is given by

\[ r = \log_2(m) + 1 \]

The delay of the hybrid Karatsuba multiplier (Equation 4.21) is obtained by subtracting the delay of \(\text{hnmul}(m)\) bit simple Karatsuba multiplier from the delay of an \(m\) bit simple Karatsuba multiplier and adding the delay of \(\text{hnmul}(m)\) bit general Karatsuba multiplier.

\[ \text{DELAY}_{H}(m) = \text{DELAY}_{S}(m) - \text{DELAY}_{S}(\text{hnmul}(m)) + \text{DELAY}_{C}(\text{hnmul}(m)) \]

**VII. Problem Definition**

The security of the ECC is based on the apparent intractability of the following elliptic curve discrete logarithm problem (ECDLP): Consider the equation: \(Q = kP\), where \(P\) and \(Q\) are points in the elliptic curve \(E(a,b)\) and \(k < P\). It is relatively easy to calculate \(Q\) given \(k\) and \(P\), but it is relatively hard to determine \(k\) given \(Q\) and \(P\). This is called the discrete logarithm problem for elliptic curves (MENE 97).

The elliptic curve consists of all real numbers for the points \(x, y, a\) and \(b\) in the \((x,y)\) coordinate plane. The \(E(a,b)\) curve plane satisfies the following equation:

\[ y^2 = x^3 + ax + b \pmod{p} \]

The prime number \(p\) sets the upper limits of the equation and is used for modulus arithmetic. \(P\) and \(Q\) are the points on the elliptic curve. When using ECC, there are two types of arithmetic, the cartesian coordinates for resolving the elliptic.
curve and modular arithmetic used for resolving the points along the coordinate system k is a very large integer generated at random which is multiplied with the point.

This system enhances the security of data transfer as well as reduces the size of the cipher text thereby eliminating the drawbacks of Diffie-Hellman and ElGamal algorithms. The working of the proposed system is as follows. In Figure 3, module A is assumed to be the sender and module B to be the receiver. A message, M has to be transmitted from A to B. This message has to be encrypted before transmission and the receiver must be able to obtain the original message after decryption.

![Figure 3 Decryption flow](image)

P is a public key used for encryption. x is a private key known only to module A. Module A calculates xP and makes it public. The values of x and P are chosen such that even with the knowledge of P, it would be nearly impossible to calculate x. Similarly, y is a private key of module B. Module B calculates yP and makes it public. The encryption and decryption steps involved in transmission and reception of a message using ECC is described below.

### VIII. STEPS INVOLVED IN ENCRYPTION AND DECRYPTION

**Encryption:**
- Let x, y be the private keys used by the transmitter and receiver respectively. The transmitter secret key x is multiplied with the public value of the receiver yP i.e., xP.
- The message is encrypted using the formula M + xyp, where M is the plain text.

**Decryption:**
- The receiver’s secret key y is multiplied with the public value of the transmitter xP i.e., yxP
- The message is decrypted by subtracting the value yxP from the received message i.e., M + xyp – yxP = M. The same is illustrated in Figure 4.

The above said encryption and decryption process are used to implement ECC algorithm in the proposed hardware module. Figure 4 shows the flowchart for encryption algorithm and Figure 5 shows the flowchart for decryption algorithm.

![Figure 4 Flowchart for encryption algorithm](image)

![Figure 5 Flowchart for decryption algorithm](image)

### IX. HARDWARE IMPLEMENTATION

The easiest and most efficient language considered for condensing the larger circuits is Verilog HDL. It is easily understandable. It is used for specification, simulation and synthesis of an electronic system. Digital circuits could be described at register transfer level (RTL) by use of HDL.

The program written in Verilog HDL run by Xilinx 14.2ISE tool when synthesised gives the hardware module of the program up to the gate level and then given for implementation using FPGA which does the chiplayout. Before implementation the program is verified by using simulation softwares such as modelsim 5.7 which gives the output as signal waveform and even the testbench program can be written and verified using simulation software before synthesis using xilinx 14.2.

The logic synthesis pushed the HDLs into the forefront to digital design. And there is no need to manually place gates to build digital circuits. They could describe complex circuits at an abstract level in terms of functionality and data flow by designing those circuits in HDLs.
Logic synthesis tools implemented the specified functionality in terms of gates and gate interconnections. Logic synthesis tools have cut design cycle times significantly. Logic synthesis is the process of converting a high level description of the design into an optimised gate level representation, given a standard cell library and certain design constraints.

Simulation is used for verification of HDL based design. They load the verilog HDL code and simulate the behaviour in software. Modelsim 5.7 is one of the software which is used for verification of the program in this thesis. The output will be in the form of the signal waveform.

**X. SIMULATION**

In order to implement ECC cryptographic system three modules namely (i) main controller (ii) point adder and (iii) multiplier are to be simulated. The program for the above three modules are written in verilog HDL.

Verilog simulation requires two main blocks, namely

1. Design Block
2. Stimulus Block

When the encryption/decryption is low (0), then the encryption part is selected. The simulation waveforms of multiplier block, adder block and main controller are obtained. The simulation waveform for multiplier block is given below. When the enable line is ‘00’ the multiplier block is selected. Then the clock is set as high and the reset as low, the system will multiply the constant K with the combination of public and private key and gives the product as KPax and KPay. The resulting output for encryption and decryption of multiplier are shown in the screen shots below in Figure 7.

![Figure 7](image)

Figure 7 Output Screen Shots for Addition Operations

The main controller controls the function of adder and multiplier. It has several internal signals for controlling the operation of the whole system. The clock is set as high and the reset is low. It has the input as the message coordinates and the public and private keys. The encrypted message coordinate is obtained as an output in the encryption process and the original message is obtained back in the decryption process. The screen shots corresponding to the main controller’s output are shown in Figure 8.

![Figure 8](image)

Figure 8 Main controller outputs for Encryption and Decryption
The architecture has been synthesized by using Xilinx ISE 14.2. The Place and Route report from ISE 14.2 shows the number of logic gates consumed. Table 2 shows the device utilization summary of encoder and decoder. Table 3 gives the timing summary.

Table 2 Device Utilization Summary of Encoder / Decoder

<table>
<thead>
<tr>
<th>Sl. No.</th>
<th>Name</th>
<th>Utilization</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Multiplexer</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td>Adder</td>
<td>11</td>
</tr>
<tr>
<td>3</td>
<td>Subtractor</td>
<td>9</td>
</tr>
<tr>
<td>4</td>
<td>Multiplier</td>
<td>69</td>
</tr>
<tr>
<td>5</td>
<td>Comparator</td>
<td>6</td>
</tr>
<tr>
<td>6</td>
<td>Latches</td>
<td>25</td>
</tr>
<tr>
<td>7</td>
<td>LUT’s</td>
<td>52</td>
</tr>
<tr>
<td>8</td>
<td>IO’s</td>
<td>312</td>
</tr>
<tr>
<td>9</td>
<td>GCLK’s</td>
<td>3</td>
</tr>
<tr>
<td>10</td>
<td>Flip flops</td>
<td>254</td>
</tr>
<tr>
<td>11</td>
<td>Registers</td>
<td>8</td>
</tr>
</tbody>
</table>

Table 3 Timing Summary

<table>
<thead>
<tr>
<th>Sl. No.</th>
<th>Name</th>
<th>Utilization</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Minimum period</td>
<td>5.043 ns</td>
</tr>
<tr>
<td>2</td>
<td>Maximum frequency</td>
<td>198.295 MHz</td>
</tr>
</tbody>
</table>

XII. CONCLUSION

In this project, the modified algorithm with the hybrid technique for implementing the Karatsuba multiplier forms the elliptic curve crypto processor. This processor yields a good result in both area and timing aspect on an FPGA when compared to existing crypto processor and using simple and general multiplier. The hybrid Karatsuba multiplier forms the most important module for the elliptic curve crypto processor.

XII. Reference