ISSN : 2278 – 1021

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 1, Issue 6, August 2012

VHDL Implementation using Elliptic Curve Point Multiplication

Ajay Kumar[1], Kunal Lala[2], Amit Kumar[3]

M.Tech (EC) (4th SEM), Dept. of ECE , Krishna Institute of Engg & Tech, Ghaziabad, India[1]
M.Tech (EC) (4th SEM), Dept. of ECE , Krishna Institute of Engg & Tech, Ghaziabad, India[2]
Professor, Dept. of ECE , Krishna Institute of Engg & Tech, Ghaziabad, India[3]

ABSTRACT— ***This paper describes synthesizable VDHL implementation of elliptic curve Point Multiplication. Elliptic curves used for ECC are defined over mathematical structures called Galois fields. Based on the theory of ECC, this paper has carried out Modular addition/subtraction, EC Point doubling/addition, Modular multiplicative inversion, EC point multiplier, projective to affine coordinates conversion. Importantly for cryptography, the elliptic curve point multiplication is the operation on which the security of every elliptic curve cryptosystem relies on.***

Keywords— **Elliptic curve point addition, point doubling, Finite field arithmetic, Point multiplication, FPGA**

I. INTRODUCTION

Since its proposal by Miller [1] and Kobliz [2] in the mi1980s, elliptic curve cryptosystem (ECC) has recently gained much attention in industry and academia. The main reason is that for a properly chosen elliptic curve, no known sub-exponential algorithm can be used to break the system through the solution of the discrete logarithm problem. This means that significantly smaller parameters can be used in ECC than in other competitive systems such as RSA and ElGamal with equivalent levels of security. Some benefits of having smaller key sizes include reductions in processing power, storage space, and bandwidth. Due to these many advantages of ECC, a number of software [3, 4] and hardware [5–13,22–24] implementations have been proposed, and included in Many standards such as IEEE 1363[13] and NIST[14] .

As well known, software implementations can easily be achieved on a general-purpose microprocessor. An operation called point addition is defined on an elliptic curve. The point addition is an operation, where two points on the curve are added and a third point, which is also on the curve, is got. Importantly for cryptography, it is very hard to tell which two points were added. Furthermore, using consecutive point additions, an operation called elliptic curve point multiplication is defined. The most exorbitant finite field operation for point addition and point doubling is the finite field inversion. However, one way to handle finite field inversion can be accomplished by transforming them into less expensive finite field operation, such as finite field addition and multiplication by using projective coordinates. For implementations of ECC, finite fields *GF(p)* and *GF* (2^m) have been used, where p is a prime and m is a positive integer. In particular, *GF*(2^m), which is an m-dimensional extension field of *GF(2)*, is suitable for hardware implementations because there is no carry propagation in arithmetic operations. The most crucial operation in ECC is the computation of point multiplication, i.e., computation of kP for given integer k and point P on elliptic curve. There are many available algorithms for the point multiplication. Depending whether the given finite field is *GF*(2^m) or *GF(p)* ,or whether the given point P is fixed or random, an ideal algorithm for computing kP may vary. However in the case of binary field *GF* (2^m), the López–Dahab algorithm [5] is one of the most popular algorithms. In fact it is a natural extension to binary case of so called Montgomery Ladder Algorithm, which is especially suit-able for hardware implementation because of the data independency of point addition and point doubling. All the elliptic curve cryptographic blocks are synthesized and simulated using Xilinx ISE 13.3 with integrated ISim Simulator.

The reminder of this paper is organized as follows: Section II describes basic Field Arithmetic behind ECC. The considered components, Algorithms and their implementation approaches are described in Section III. Simulation results and performance evaluation are discussed in section IV and section V concludes the paper.

II. Methodology

This approach demonstrates ECC Point Multiplication structure of moderate gate count and high speed and is organized as follows.

Point multiplication is the operation that dominates the execution time of an Elliptic Curve Cryptographic protocol. Implementation operation of point multiplication can be separated into three distinct layers:

A. Point multiplication technique
B. Elliptic curve point addition & doubling
C. Finite Field Arithmetic

Operations involved in point multiplication have the hierarchical formulation as shown in Fig.1 with point multiplication techniques near the top and the fundamental finite field arithmetic at the base. For Example one may decide to implement ECDSA signature generation entirely in hardware so that the only input to the device is the message to be signed, and the only output is the signature for that message.

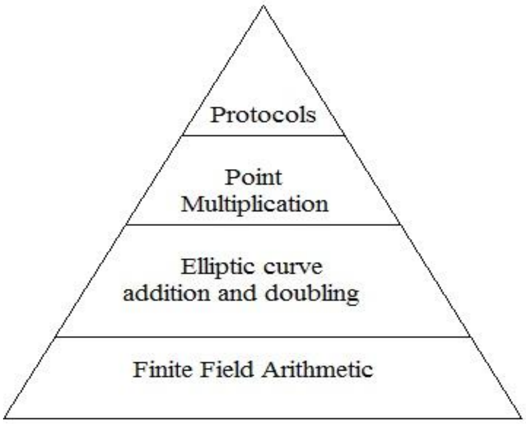


Fig. 1: Hierarchy of operation in ECC

Point Multiplication is the basic computation primitive of elliptic curve cryptography. The definition of corresponding operations depends on a particular field, but they always amount to combinations of arithmetic operation (add, subtract, multiply, square and divide over the chosen field so that software implantation is carried out. The scalar multiplication is computed using Algorithm 1.

A. *Point Multiplication*

A basic operation of any elliptic curve cryptosystem is an elliptic curve point multiplication given as

$$Q = kP = P+P+P+P+\ldots\ldots+P \qquad (1)$$

Where P is a point on an elliptic curve E and k is an integer in a range 1≤ k < order (*P*). Accordingly, the elliptic curve point multiplication means that the point P is added to itself k times. The order of the point P is n_0 if and only if P multiplied with n_0 results in the point at infinity. This is formally described as follows:

$$\text{Order (P)} = n_0 \leftrightarrow n_0P = O_\infty$$

The strength of an elliptic curve cryptosystem lies in the fact that if E, *Q* and *P* are given, it is a very hard task to recover *k*. This is a so called Elliptic Curve Discrete Logarithm Problem (ECDLP).

The integer k is usually very large and, therefore, it would be way too slow to calculate Q just by adding P to itself k times. Thus, efficient elliptic curve point multiplication methods are needed. The simplest and oldest of such methods is the binary method which is also known as the double-and-add-method. The binary method relies on the binary expansion of k. In a binary form, *k* is given as

$$k = \sum_{i=0}^{l-1} k_i 2^i$$

and, therefore, *l* bits are needed to present *k* in the binary form.

The scalar multiplication of the point *P* is computed using the Algorithm 1.

Algorithm 1: Right-to-left binary method for point multiplication

Input: k = (kl−1, k1, k0) in binary, P belongs to $E\ (F(2^m))$.
Output: $k*P$

```
Q←∞
For i from 0 to l −1 do
If  k (i) = 1 then Q←Q + P
P←2P
Return (Q)
```

There are several methods derived for efficient elliptic curve point multiplication, many of which require pre computations before the actual point multiplication. These pre computations include calculations of intermediate points which are then used for speeding up the point multiplication. Certain methods use different representations of the integer k, so that the number of operations during point multiplication can be reduced.

B. *Elliptic curve point addition & doubling*

Consider the Koblitz curve: $y^2 + xy = x^3 + x^2 + 1$ over *GF (2)* and the extension field $L = GF\ (2^{163})$. A polynomial representation based on the irreducible polynomial

$$f(x)=x^{163}+x^7+x^6+x^3+1 \qquad (2)$$

will be used.

Point addition: Let *P* = (x_1, y_1) belongs to E ($F2^m$) and *Q*= (x_2, y_2) belongs to E ($F2^m$), where *P* not equal to *Q*. Then *P* + *Q* = (x_3, y_3), where $x_3 = \lambda^2 + \lambda + x_1 + x_2 + a$ and $y_3 = \lambda(x_1 + x_3) + x_3 + y_1$ with $\lambda = (y_1 + y_2)/(x_1 + x_2)$.

Point doubling: Let *P* = (x_1, y_1) belongs to E ($F2^m$), where *P* = −*P*. Then 2*P* = (x_3, y_3), and $x_3 = \lambda^2 + \lambda + a = x_1 2 + b/x_1 2$ and $y_3 = x_1 2 + \lambda x_3 + x_3$ with $\lambda = x_1 + y_1/x_1$.

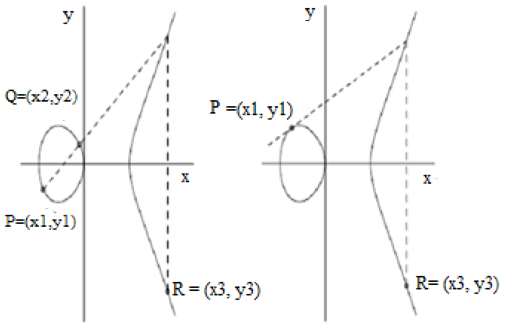


Fig. 2: Point Addition and Point Doubling

C. Finite Field Arithmetic

Fields are abstractions of familiar number systems (such as the rational numbers *Q*, the real numbers *R*, and the complex numbers *C*) and their essential properties. They consist of a set *F* together with two operations, addition (denoted by +) and multiplication (denoted by ·), that satisfy the usual arithmetic properties. If the set F is finite, then the field is said to be finite. A field *F* is equipped with two operations, addition and multiplication. Subtraction of field elements is defined in terms of addition and can be given as $i - j = i + (- j)$ where − j is the unique element in *F* such that $j + (- j) = 0$ (− j is called the negative of j).Similarly, division of field elements is defined in terms of multiplication: with j = 0, $i/j = i \cdot j^{-1}$ where j^{-1} is the unique element in *F* such that $j \cdot j^{-1} = 1$ ($^{j-1}$ is called the inverse of j).Arithmetic unit shown in Fig.3 carries out these finite field operations.

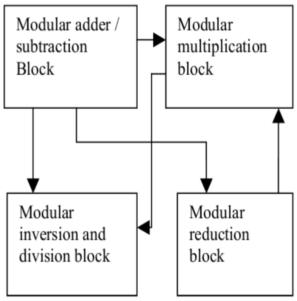

Modular adder / subtraction Block
Modular multiplication block
Modular inversion and division block
Modular reduction block

Fig. 3: Arithmetic unit Block diagram For ECC

Several field operations have been carried out and described in details in next section with necessary circuitry and mathematics.

III. Implementation Approach

The computation primitives for executing the elliptic-curve operations are addition, multiplication, division, inversion and squaring over *GF* (2^m)*.* The first one amounts to the component-by-component addition of the corresponding polynomials. The corresponding circuit is made up of m XOR gates, and its computation time is equal to 1 clock cycle. For multiplying, the generic interleaved multiplier model can be used. For dividing, a simplified version of binary divider, adapted to the case where p = 2, is used.

A. Point Addition

A data path for computing the following equation $\lambda = (y_1 + y_2)/(x_1 + x_2)$, $x_3 = \lambda^2 + \lambda + x_1 + x_2 + 1$ and $y_3 = \lambda(x_1 + x_3) + x_3 + y_1$ is shown below in fig 4.According to the structure of the data path, the computation time is approximately equal to

$$T_{point\text{-}addition} \approx m\,(T_{mod\text{-}\,f\text{-}product} + T_{mof\text{-}f\text{-}division}) \qquad (3)$$

To summarize, doubling has been substituted by squaring, a simple operation over a binary field.

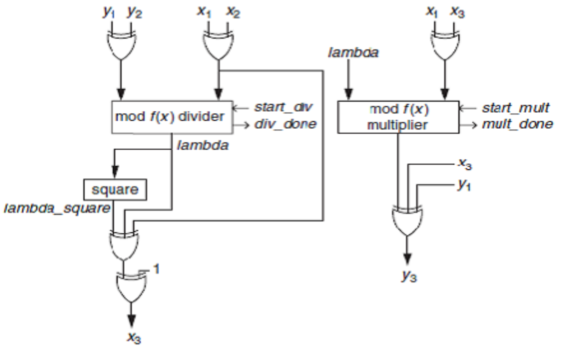


Fig. 4: Point Addition

B. Interleaved Multiplier

The simplest algorithm for *GF* (2^m) multiplication is the shift and adds method with the reduction step interleaved. Multiplication of two elements a(x), b(x) in *GF* (2^m) can be given as:

$$C(x) = a(x)\, b(x) \bmod f(x) = a(x) \sum_{i=0}^{m-1} b_i x^i$$

$$= \left(\sum_{i=0}^{m-1} b_i a(x)\, x^i \right) \bmod f(x) \qquad (4)$$

Therefore, the product c(x) can be computer as

$$C(x) = (b_0 a(x) + b_1 a(x)\, x + b_2 a(x)\, x^2 + \ldots\ldots + b_{m-1}\, a(x)\, x^{m-1}) \bmod f(x) \qquad (5)$$

In this approach Processing of bits of b(x) follows LSB to MSB method. In a least-significant-bit (LSB) multiplier, the coefficients of b(x) are processed starting from the least-significant bit b_0 and continue with the remaining coefficients one at a time in ascending order. We have implemented LSB first method because LSB first scheme is faster than the MSB first scheme since in LSB first approach c(x) and a(x) can be updated in parallel.

Thus multiplication according to this scheme is performed in the following way:

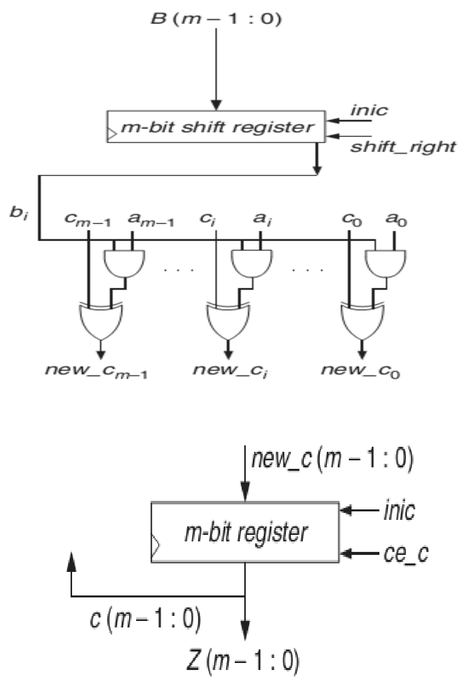
$c(x) = a(x)b(x) \bmod f(x)$

$= (b_0\, a(x) + b_1\, a(x)x + b_2\, a(x)x^2 + \ldots\ b_{m-1}\, a(x)x^{m-1} \bmod f(x)$

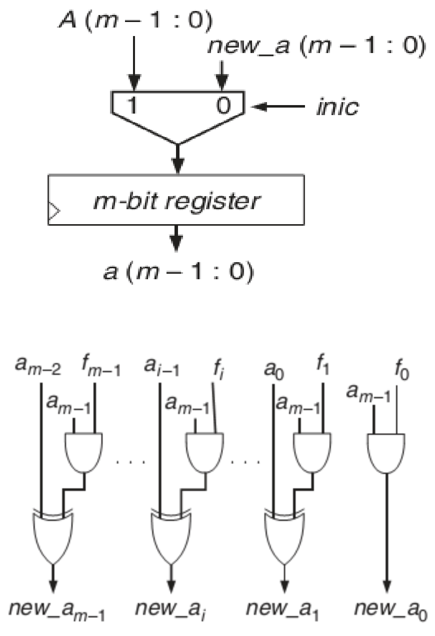
$= (b_0\ a(x) + b_1(a(x)x) + b_2(a(x)x^2\ ) + \ldots . b_{m-1}(a(x)x^{m-1}\ ))\bmod f(x)$

$= (b_0\ a(x) + b_1(a(x)x) + b_2(a(x)x)x + \ldots . b_{m-1}(a(x)x^{m-2}\ )x) \bmod f(x) \qquad (6)$

Fig. 4 (a and b) depicts the data path for the binary version of LSB first multiplier. It is important to note that in the LSB and MSB first multiplication schemes, several coefficients could be processed at each step.



(a)



(b)

Fig. 4: a and b Interleaved LSB-first multiplier

C. Squaring

A straight forward method for implementing field squaring in $GF\ (2^m)$ using the multiplication algorithms with only one input operand in order to perform $c(x) = a(x)a(x) \bmod f(x)$ that is, the operand $b(x)$ is substituted by $a(x)$. MSB-first and LSB-first approaches for squaring can also be given in a similar manner. Fig. 5 depicts the data path for model for squaring which includes the component poly-reducer.

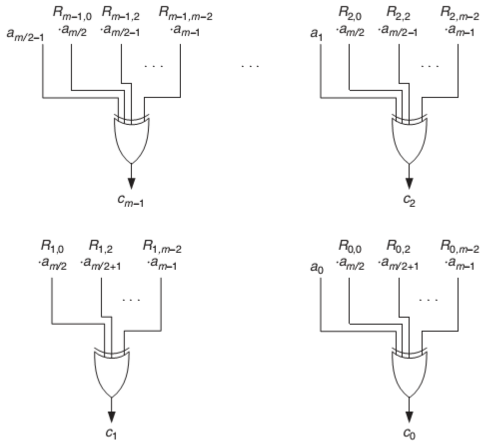


Fig. 5: Classic squaring

D. Binary Division

The quotient of two polynomials in GF (2^m) can be computed using the binary version of the binary algorithm that is used for calculation of gcd from required polynomials.

The binary algorithm for computing $z(x) = g(x)h^{-1}(x)$ mod $f(x)$ has been described as follows If p =2, it can be simplified. Given two polynomials a(x) and b(x), if both are divisible by x, that is, if $a_0 = b_0 = 0$, then gcd (a(x), b(x)) = x. gcd(a(x)/x, b(x)/x); say b(x), is divisible by x($b_0 = 0$) and the other is not ($a_0 \neq 0$), then gcd(a(x), b(x)) = gcd(a(x), b(x)/x); if none of them is divisible by x then define a new polynomial $ab(x) = a(x) - a_0\ b_0^{-1} b(x)$, so that gcd(a(x), b(x)) = gcd(ab(x), b(x)) = gcd(ab(x), a(x)), and ab(x) is divisible by x. The corresponding data path is shown in Fig. 6.

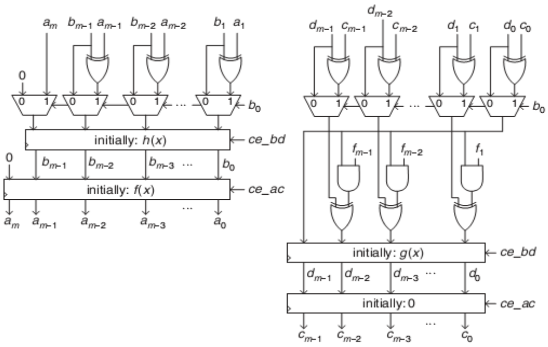


Fig. 6: Binary algorithm: data path

However based on computational similarities algorithm used for binary division can also be used for computing inversion as well. Additionally, the complete circuitry includes additional integral part for accumulation and updating the variables alpha and beta which includes a control unit as well.

E. Point Multiplication

By combining all the above blocks we can implement the point multiplication whose data path is represented in fig 7. The point-doubling operation can be avoided in the case of the two following Koblitz curves over *GF (2^m):*

$$E_0 : y^2 + xy = x^3 + 1 \tag{7}$$

$$E_1 : y^2 + xy = x^3 + x^2 + 1 \tag{8}$$

For that define the Frobenius map τ from $E_c(GF(2^m))$, *with c= 0 or 1:*

$$\tau(\infty) = (\infty) \qquad \tau(x,y)=(x^2,y^2) \tag{9}$$

It can be demonstrated that

$$2P = -\tau^2(P) + \mu\tau(P)$$

With $\mu = 1$ if $c = 1$ and $\mu = -1$ if $c = 0$.

Thus the point-doubling operation amounts to squaring operations in *GF (2^m)* for computing $\tau(P)$ and $\tau^2(P)$ and a point addition.

Algorithm 2: *τ -ary representation of k*

```
$a := k; b := 0; i := 0;$
while $a /= 0$ or $b /= 0$ loop
    if $a \bmod 2 = 0$ then $r(i) := 0;$
    else $r(i) := 2 - ((a - 2*b) \bmod 4);$
    end if;
    $old\_a := a;$
    $a := b + mu*(old\_a - r(i))/2; b := (r(i) - old\_a)/2;$
    $i := i+1;$
end loop;
```

Regarding the maximum value of tin the particular case where *a= k and b= 0,* it has been demonstrated that

$$t \approx 2\log_2 k \tag{10}$$

To summarize, *doubling* has been substituted by *squaring*, a simple operation over a binary field. Furthermore, among two successive coefficients r_i. Thus, according to Eq. (10.66), upper bounds of the number of nonzero coefficients r_i is given by

$$S \approx \log_2 k \approx m \tag{11}$$

Thus, the computation of *kP* includes at most m complex operations (*adding* or *subtracting*), and the total computation time should be roughly half the computation time of that of the basic algorithms.

In order to implement the preceding algorithm, an upper bound of *a* and *b* must be known. It can be demonstrated that

$$-2^m \leq a < 2^m \text{ and } -2^{m-1} \leq b < 2^{m-1} \tag{12}$$

So that *a* is an (*m*+ *1*)-bit 2s complement number and *b* an *m*-bit 2s complement number. A data path for executing Algorithm is shown in Fig. 7.
According to Eqs. (10) And (11), their computation time for point multiplication is approximately equal to

$$T \approx mT_{point\text{-}addition} \approx m^2(T_{mod\text{-}f\text{-}product} + T_{mod\text{-}f\text{-}division}) \tag{13}$$

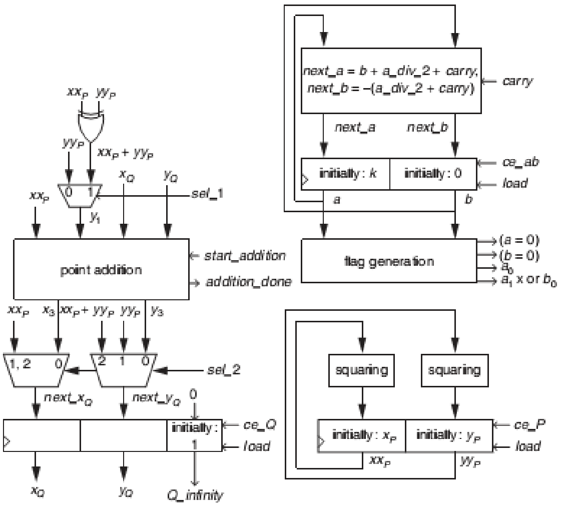


Fig. 7: Point multiplication

IV. Experimental Results

We have synthesized and simulated the architecture for a Xilinx Spartan XC3s400-4pq208 FPGA, using the ISE 13.3(nt) and ISim simulator.

Arithmetic units were synthesized for the Koblitz curve recommended by NIST [14], for the finite field *GF* (2^m) using the irreducible polynomial $f(x) = x^{163}+x^7+x^6+x^3+1$. Synthesis results for *point addition* and *point multiplication* are summarized in table I and table II.

Simulation results by using ISim Simulator for *point addition* and *point multiplication* are shown as waveform in fig. 8 and fig. 9 respectively.

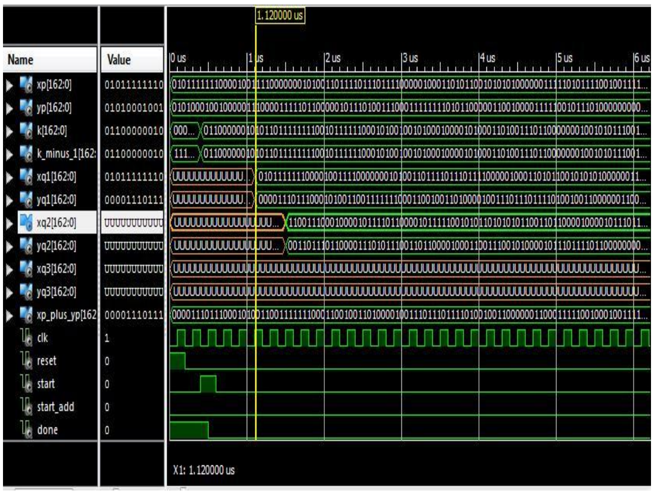


Fig. 8: Point Addition

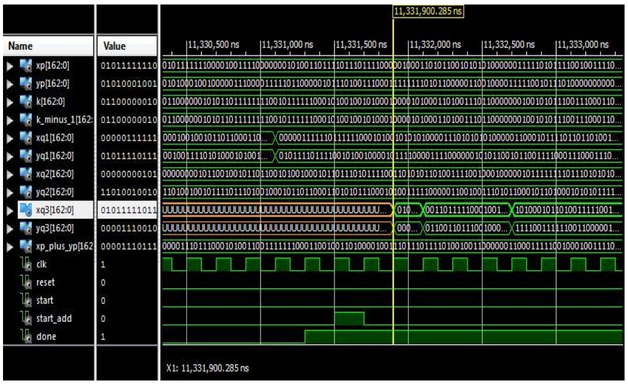


Fig. 9: Point Multiplication

TABLE I
DEVICE UTILIZATION SUMMARY FOR POINT ADDITION

Logic Utilization	Used (2^{163})
No. of slice FF	1176
No. of 4 input LUTs	1772
No. of occupied slices	944
No. of bounded IOBs	982

TABLE II

DEVICE UTILIZATION SUMMARY FOR POINT MULTIPLICATION

Logic Utilization	Used (2^{163})
No. of slice FF	2163
No. of 4 input LUTs	3677
No. of occupied slices	2092
No. of bounded IOBs	819

V. Conclusion

This paper presents an implementation of an Elliptic Curve co-processor components, *Point addition* and *Point Multiplications*. Future work will include considering more efficient algorithms to perform the field arithmetic operations, ITA for field Inversion. For the crypto-graphic work, the time to perform the scalar multiplication can be improved if projective coordinates are used to represent the point of the curve and the Montgomery method is used to compute the scalar multiplication.

REFERENCES

[1] V.S. Miller, "Use of elliptic curves in cryptography," in Proceedings of the Advances in Cryptology, CRYPTO'85, pp. 417–426, 1986.

[2] N. Koblitz, "Elliptic curve cryptosystems," Mathematics of Computation 48, pp. 203–209, 1987.

[3] M. Rosing, "Implementing elliptic curve cryptography," Manning, 1999.

[4] D. Hankerson, J. Hernandez and A. Menezes, "Software implementation of elliptic curve cryptography over binary fields," Proceedings of the CHES 2000, Lecture Notes in Computer Science, vol. 1965, pp. 1–24 , 2000.

[5] Julio López and Ricardo Dahab, "Fast Multiplication on Elliptic Curves over *GF(2^m)* Without Precomputation," In CHES 99, Proceedings of the First International Workshop on Cryptographic Hardware and Embedded Systems, London, UK: Springer-Verlag, pp. 316–327, 1999.

[6] G. Orlando, C. Parr, "A high-performance reconfigurable elliptic curve processor for *GF(2^m)*," in CHES 2000, Lecture Notes in Computer Science, 1965.

[7] Steffen Peter and Peter Langendörfer. "An Efficient Polynomial Multiplier in *GF(2^m)* and its application to ecc designs," In DATE 07, Proceedings of the conference on Design, automation and test in Europe, San Jose, CA, USA, 2007.

[8] Sabel Mercurio Henríquez Rodríguez et. Al, "An Fpga Arithmetic Logic Unit for Computing Scalar Multiplication using the Half-and-Add Method," In ReConFig 2005 Washington, DC, USA.

[9] Francisco Rodríguez-Henríquez and Çetin Kaya Koç, "On Fully Parallel Karatsuba Multipliers for *GF(2^m)*," In Proc. of the International Conference on Computer Science and Technology (CST), pp. 405–410.

[10] Francisco Rodríguez-Henríquez et.al, "Parallel Itoh-Tsujii Multiplicative Inversion Algorithm for a Special Class of Trinomials," Des. Codes Cryptography, pp. 19–37, 2007.

[11] André Weimerskirch and Christof Paar, "Generalizations of the Karatsuba algorithm for Efficient Implementations," Cryptology ePrint Archive, Report 2006/224, 2006.

[12] N.A. Saqib, F. Rodríguez-Henríquez, A. Díaz-Pérezm, "A parallel architecture for fast computation of elliptic curve scalar multiplication over *GF(2^m)*," in Parallel and Distributed Processing Symposium (IPDPS), 2004.

[13] IEEE 1363, Standard Specifications for Public key Cryptography, 2000.

Biography

Ajay Kumar received his B.Tech Degree from B.S.A. College of Engg. & Tech. , Mathura, India in 2009.
Currently he is a research Scholar at Krishna Institute of Engg. & Technology, Ghaziabad, India. His research interests are in Computer Network security, Cryptographic Algorithms (ECC), VLSI Design.

Kunal Lala received is B.Tech Degree from Lord Krishna College of Engineering, Ghaziabad in 2010. Currently he is a research Scholar at Krishna Institute of Engg. & Technology, Ghaziabad, India. His research interests are in Computer Network security, Cryptographic Algorithms (AES), Computer Engineering.

Prof. Amit Kumar currently he is a Professor at Department of Electronics & Communication Engineering, Krishna Institute of Engineering & Technology, Ghaziabad. His research interests are in VLSI design and Testing, Cryptographic Algorithms (IDEA), Network Security.