Area Optimized Low Latency Karatsuba Ofman Multiplier Variant for Elliptical Curve Cryptography

Sunil Devidas Bobade[1*] and Vijay R. Mankar[2]

[1] Research Scholar
S.G.B.Amravati University (Amravati,India)

[2] Deputy Secretary
R.B.T.E. (Pune, India)

[*] *Corresponding author's email: Sunilbobade73 [AT] gmail.com*

ABSTRACT— ***Due to resource constrains, implementation of secure protocols for securing embedded systems has become a challenging task. System designers are advised to design and install area efficient versions of existing, proven security protocols. System designers are finding ways and means to compress existing security protocols without compromising security and without tampering with basic security structure of algorithm. Modular multiplication, point multiplication, point doubling are few critical activities to be carried out in ECC algorithm. By optimizing Modular Multiplier, area efficiency in ECC algorithm can be achieved. In this paper, we propose Area optimized and low latency multiplier that implements the efficient KOA algorithm in altogether novel style to be used in ECC architecture. The proposed algorithm uses a novel technique of splitting input operands based on exponent's parity and it eventually helps in reducing FPGA footprint and offers low latency by avoiding overlapping, prime concern for any embedded system. The complete modular multiplier and the cryptoprocessor module is synthesized and simulated using Xilinx ISE Design suite 14.4 software. We have investigated area occupancy of proposed multiplier and cryptoprocessor and concluded that proposed scheme occupies relatively reduced percentage area of FPGA as compared to the one using traditional KOA multiplier.***

Keywords— ECC, Double Point Multiplication, Karatsuba Ofman multiplication, Area optimization **.**

1. INTRODUCTION

Integrating security protocols in embedded systems is not an easy proposition. Embedded system designers have so far failed to install required levels of security in embedded systems, due to unusual design constraints like storage limitation, restricted processor performance and easy power drain in embedded devices. Regular cryptography algorithms are not suitable for Embedded systems due to wide footprint. Due to resource constrains in the design and implementation of secure protocols, system designers are well advised to use area efficient versions of existing, proven security protocols, rather than developing their own protocols or implementations. This call for refined, area and space optimized; easy to deploy versions of original cryptography algorithms tailor made for resource constrained Embedded system.

Of the available choices, AES is the most powerful, most secured encryption algorithm with a key size ranging from 128 to 256. RSA is another well established and most preferred public key cryptography algorithm. To provide security equivalent to AES, RSA public-key sizes would have to range between 3,072 and 15,000 bits long, too big for embedded implementation. One appealing solution to the key size problem is the promising family of asymmetric algorithms known as Elliptic Curve Cryptography, or ECC.

Victor Miller and Neal Koblitz proposed the concept of elliptic curve cryptography in the mid of 1980's as an advancement in public key cryptographic systems such as DSA and RSA. The main advantage of ECC is the usage of shorter key helping compact implementations, resulting in faster cryptographic operations, running on smaller chips or more compact software. For hardware-based implementations of security functions, the benefits of ECC are more in comparison to RSA and AES. Optimized ECC chip designs have been designed and are as much as 37 times faster than its software counterparts. ECC offers other advantages of small software footprint, low hardware implementation costs, low bandwidth requirements, high device performance. Due to these many advantages of ECC a number of hardware implementations have been proposed, and included in many standards such as IEEE 1363and NIST .

Modular multiplication is the most primitive and critical operation in ECC. The design of Finite field multipliers is the complex design issue in the implementation of the ECC processor. A number of multipliers with different area and time

complexity are reported in the available literatures. The Karatsuba Ofman algorithm is agreed upon as a most efficient multiplication algorithm and is widely adopted in VLSI implementation. Here input operands are processed as the "most significant half" and the "least significant half". In proposed multiplier, instead of splitting input operands into the "most significant half" and the "least significant half", our method split operands according to the parity of multiplicands's exponent. Both the space and time complexities of the resulting multiplier are found to be much better than that of traditional multiplier. Here we have concentrated and investigated on area optimization. This is a significant achievement if we intend to use this multiplier in FPGA implementation of elliptical curve cryptography for embedded systems.

The basic step of Karatsuba Ofman's algorithm is a formula that computes the product of two large numbers using three multiplications of smaller numbers, each with about half as many digits as operands, plus some additions and digit shifts. Karatsuba Ofman method of multiplication is a faster way of multiplying two integers of length n. For the first step of the algorithm, it initially requires the breaking of multi digit integers into parts. These parts can then be used in three multiplications to produce a solution.

The Karatsuba algorithm is an effective multiplication algorithm. It diminishes the multiplication of two n-digit numbers to at most single-digit multiplications in general (and exactly when n is a power of 2). It is along these lines quicker than the traditional algorithms, which requires n^2 single-digit products. If $n = 2^{10} = 1024$, in particular, the precise counts are $3^{10} = 59{,}049$ and $(2^{10})^2 = 1{,}048{,}576$, respectively. Here the operands α and β can be divided into two equal-size parts α_L and α_H , β_L and β_H respectively, which represent the $l/2$ higher and lower order bits of α and β. We can split them in two parts as follows:

$$\alpha(x) = \sum_{i=0}^{l-1} a_i x^i = \sum_{i=\frac{l}{2}}^{l-1} a_i x^i + \sum_{i=0}^{\frac{l}{2}-1} a_i x^i$$

$$= \alpha^{\frac{l}{2}} \sum_{i=0}^{\frac{l}{2}-1} a_{i+\frac{l}{2}} x^i + \sum_{i=0}^{\frac{l}{2}-1} a_i x^i$$

$$\alpha(x) = x^{\frac{l}{2}} \alpha_H + \alpha_L$$

Likewise,

$$\beta(x) = x^{\frac{l}{2}} \beta_H + \beta_L$$

The product $\gamma(x)^*$ can be computed as

$$\gamma(x)^* = \alpha(x).\beta(x)$$

By using above equation output can be represented as,

$$\gamma(x)^* = \alpha_L \beta_L + \alpha_H \beta_H x^l + (\alpha_H \beta_L + \alpha_L \beta_H) x^{\frac{l}{2}}$$

To further improve the computation of the product $\gamma(x)^*$ equation can be modified as,

$$\gamma(x)^* = \alpha_L \beta_L + \alpha_H \beta_H x^l + (\alpha_L \beta_L + \alpha_H \beta_H + (\alpha_H + \beta_L)(\beta_H + \beta_L) x^{\frac{l}{2}}$$

2. RELATED WORK

Several modular multiplication algorithms have been proposed. Of them all, Karatsuba- Ofman algorithm , KOA is widely used for performing modular arithmatic. In [1] a variant of Karatsuba multiplier of the type $GF((2^n)^8$ is presented and is highly area efficient . Kimmo U. Järvinen *et.al* [2] adopted an efficient implementation of point multiplication on Koblitz curves and was designed for extremely-constrained, secure applications. In that approach a new technique was introduced for point addition which required fewer registers and small processor. In [3] Hossein Mahdizadeh and Massoud Masoumi built elliptic curve cryptographic processor by organizing multipliers in parallel.

In [4], hybrid multiplier was proposed that intelligently switches between two variants of multiplier depending on the size of multiplicands. The Karatsuba multiplier is efficient algorithm ensuring fewer LUTs and stable number of Flip-flops for the smaller bit multiplications, while the systolic variant ensures fewer LUTs count for the bigger size multiplicands. This hybrid multiplier does the initial recursion using the systolic algorithm while final small sized multiplications are accomplished using the Karatsuba algorithm. Area analysis report suggested that by using a hybrid multiplier instead of just traditional Karatsuba Multiplier, eventually helps in reducing FPGA footprint**.** This hybrid multiplier exhibits a savior of 7.56 % in terms of Flip flop slices. involves 52 % fewer LUTs and utilizes 47% fewer slices as compared to traditional Karatsuba multiplier for 256 bit multiplication.

In [5], FPGA based area efficient ECC processor was built using Digit multiplier. Instead of processing vector (polynomial) bit by bit or parallelly, operands are process in 16-bit word format. This Modular multiplier exhibits a savior of 23.88% in terms of Flip flop slices. Proposed Multiplier involves 62 % fewer LUTs and utilizes 59% fewer slices as compared to traditional Karatsuba multiplier for 256 bit multiplication. Further reduction was achieved in ECC processor by employing efficient double point multiplication algorithm.

Area and speed efficient 163 bit Scalar multiplier with improved area and speed was designed by Sujoy Sinha Roy *et.al* [6]. Scalar multiplication was performed on Xilinx Virtex V platforms over GF (2^{163}). The implementation used a novel three stage pipelined bit-parallel Karatsuba multiplier with subquadratic complexity. Scalar multiplication algorithm, optimized field primitives, balanced pipeline stages, and enhanced scheduling of point arithmetic all contributed to a high-speed architecture with a significantly small area Hybrid binary-ternary number system for elliptic curve cryptosystems was designed by Jithra Adikari *et.al* [7]. In their newly designed system three novel algorithms for both single and double scalar multiplication were implemented. The first algorithm is w-HBTF and the other two algorithms, namely, HBTJF and RHBTJF.The output results showed that hybrid algorithms are almost always faster than classical w-NAF methods or JSF. Kazuo Sakiyama et.al [8] implemented modular multiplication algorithm that integrates three different existing algorithms, based on Barrett reduction and Montgomery reduction. This modular multiplier is highly speed efficient.

3. PROPOSED AREA OPTIMIZED LOW LATENCY MULTIPLIER

Proposed multiplier is based on Karatsuba Ofman Algorithm. In traditional KOA two operands A and B are splitted as higher and lower significant bits and are represented in polynomial format as

$$A=\sum_{i=0}^{n-1} a_i X^i$$

$$=X^m \sum_{i=0}^{m-1} a_{m+i} X^i + \sum_{i=0}^{m-1} a_i X^i$$

$$= X^m A_H + A_L$$

Similarly,

$$B=\sum_{i=0}^{n-1} b_i X^i$$

$$=X^m \sum_{i=0}^{m-1} b_{m+i} X^i + \sum_{i=0}^{m-1} b_i X^i$$

$$= X^m B_H + B_L$$

Where

$A_H=\sum_{i=0}^{m-1} a_{m+i} X^i$ and $A_L=\sum_{i=0}^{m-1} a_i X^i$

$B_H=\sum_{i=0}^{m-1} b_{m+i} X^i$ and $B_L=\sum_{i=0}^{m-1} b_i X^i$

The product of A and B can be written as

$$AB=A_H B_H X^{2m}+\{[(A_H+A_L)(B_H+B_L)] - [A_H B_H+A_L B_L]\}Xm+A_L B_L$$

In Modulo 2 domain, addition and subtraction operation can be accomplished using XOR Gate and product using AND gate. Total gates involved for implementing above expression will be five AND gates for accomplishing five multiplication activities and four XOR gates for performing modulo 2 addition. Further, it is a three level realization, as input operand has to pass through maximum of three levels of XOR gate before reaching output line because of

overlapping. Therefore, total XOR gate delay for implementing the above expression will be $3T_D$ besides the cost of the recursive computation of the three partial products. Thus basic KOA multiplier utilizes five AND gates and four XOR gates to accomplish the basic multiplication activity.

Instead of splitting input operands into the "most significant half" and the "least significant half", the method split operands according to the parity of X's exponent.
Accordingly, Operands A and B may be rewritten as

$A= \sum_{i=0}^{n-1} a_i X^i$

$=\sum_{i=0}^{m-1} a_{2i} X^{2i} + \sum_{i=0}^{m-1} a_{2i+1} X^{2i+1}$

$=\sum_{i=0}^{m-1} a_{2i} X^{2i} + X \sum_{i=0}^{m-1} a_{2i+1} X^{2i}$

Similarly,

$B= \sum_{i=0}^{n-1} b_i X^i$

$=\sum_{i=0}^{m-1} b_{2i} X^{2i} + \sum_{i=0}^{m-1} b_{2i+1} X^{2i+1}$

$=\sum_{i=0}^{m-1} b_{2i} X^{2i} + X \sum_{i=0}^{m-1} b_{2i+1} X^{2i}$

Now let us define

$A_E= \sum_{i=0}^{m-1} a_{2i} X^{2i}$ $A_O=\sum_{i=0}^{m-1} a_{2i+1} X^{2i}$

$B_E=\sum_{i=0}^{m-1} b_{2i} X^{2i}$ $B_O=\sum_{i=0}^{m-1} b_{2i+1} X^{2i}$

Let i=2 and $X^2=Y$,

Then product AB can be written as

$AB= (A_E(y) + XA_O(y)) (B_E(y) + XB_O(y))$

$=\{A_E(y)B_E(y)+X^2A_O(y)B_O(y)\}+ X\{A_E(y)B_O(y) + A_O(y)B_E(y))\}$

Applying KOA formula to above expression,

$= \{[A_E(y)B_E(y) + YA_O(y)B_O(y)]\} +X\{[(A_E(y) + A_O(y))(B_E(y) + B_O(y))]- [A_E(y)B_E(y) + A_O(y)B_O(y)]\}$

In VLSI implementation of above expression multiplying a polynomial by x or $y = x^2$ is equivalent to shifting its coefficients left, and no gate is required. For implementing this revised expression four AND gates and three XOR gates are required. The expressions in the three square brackets can be computed concurrently, and these addition operations require one XOR gate delay $1T_D$. We conclude that computing product AB needs only a total delay of $2T_D$ besides the cost of the recursive computation of the three partial products.

Compared to the $3T_D$ gate delays required in traditional formula, one XOR gate delay $1T_D$ is saved for each recursive iteration. Thus compared to the four EXOR gates delays required in traditional formula, one XOR gate and also one AND gate is saved for each recursive iteration, which is a significant achievement considering embedded systems are highly resource constrained. . Thus this variant of KOA multiplier utilizes four AND gates and three XOR gates to accomplish the basic multiplication activity.

From the above basic concept, it is observed that by splitting the operand polynomials, based on the parity, results in reduction of hardware (fewer AND and XOR gates) and latency time (fewer levels due to little overlapping). By extending this very concept of splitting the operands based on parity of exponent to FPGA implementation, we have explored and implemented FPGA based finite field multiplier and compared FPGA footprint with that of traditional KOA multiplier. Also we have incorporated the proposed multiplier in ECC processor and investigated its impact on the footprint of entire processor. In this paper, we have only concentrated on area occupancy of proposed multiplier, its

impact on processor footprint but haven't investigated quantum of latency improvement, which forms the future scope of investigation.

4. RESULTS AND DISCUSSION

In this section, we focus on the FPGA implementation of the proposed multiplier. The proposed architecture is coded in verilog HDL and is synthesized using Xilinx ISE version 14.4 design software and is implemented on Xilinx Virtex-4 xc4vlx200ff1513 FPGA. The RTL schematic for the proposed Finite Field multiplier is shown in figure 1

Fig 1. RTL Diagram of Proposed multiplier

Area Report of Proposed Multiplier

Since the area of the complete processor mainly depends on the incorporated GF multiplier, most of the slices in the target device are utilized by it. From table 2, for 16 bit multiplicands, proposed modular Multiplier needs 42 slices out of 89,088 available slices in the target device. Among the 178,176 available four input LUTs only 78 are used. The multiplier also needs only 45 out of 178,176 Flip Flops.

Table 1. RESOURCE UTILIZATION COMPARISON OF TWO MULTIPLIERS

	Karatsuba Ofman Multiplier				Proposed New Version Multiplier			
Multiplicand Size	Flip flops	LUTs	Occupied Slices	IOBs	Flip flops	LUTs	Occupied Slices	IOBs
2 bit	17	25	16	9	15	16	13	9
4 bit	32	75	43	17	21	31	19	13
8 bit	46	150	79	33	29	57	32	21
16 bit	54	162	87	65	45	78	42	37
32 bit	102	315	168	129	81	142	78	69
64 bit	266	665	411	257	145	270	143	133
128 bit	646	1782	1000	513	273	526	272	261
256 bit	1118	13761	7008	913	530	1038	529	517

Table 2. RESOURCE UTILIZATION BY MULTIPLIER FOR L=16

Device Utilization Summary			
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Flip Flops	45	178,176	1%
Number of 4 input LUTs	78	178,176	1%
Number of occupied Slices	42	89,088	1%
Number of Slices containing only related logic	42	42	100%
Number of Slices containing unrelated logic	0	42	0%
Total Number of 4 input LUTs	78	178,176	1%
Number of bonded IOBs	37	960	3%
Number of BUFG/BUFGCTRLs	1	32	3%
Number used as BUFGs	1		
Average Fanout of Non-Clock Nets	2.35		

A bar chart representation shown in figure 2 compares the resource utilization of Proposed multiplier with traditional Karatsuba Ofman Multiplier for L =16. Thus proposed multiplier exhibits a savior of 20 % in terms of Flip flop slices. Proposed Multiplier involves 53 % fewer LUTs and utilizes51.72% fewer slices as compared to traditional Karatsuba Ofman multiplier for 16 bit multiplication. This helps in bringing down the footprint of modular multiplier on FPGA while building cryptography schemes.

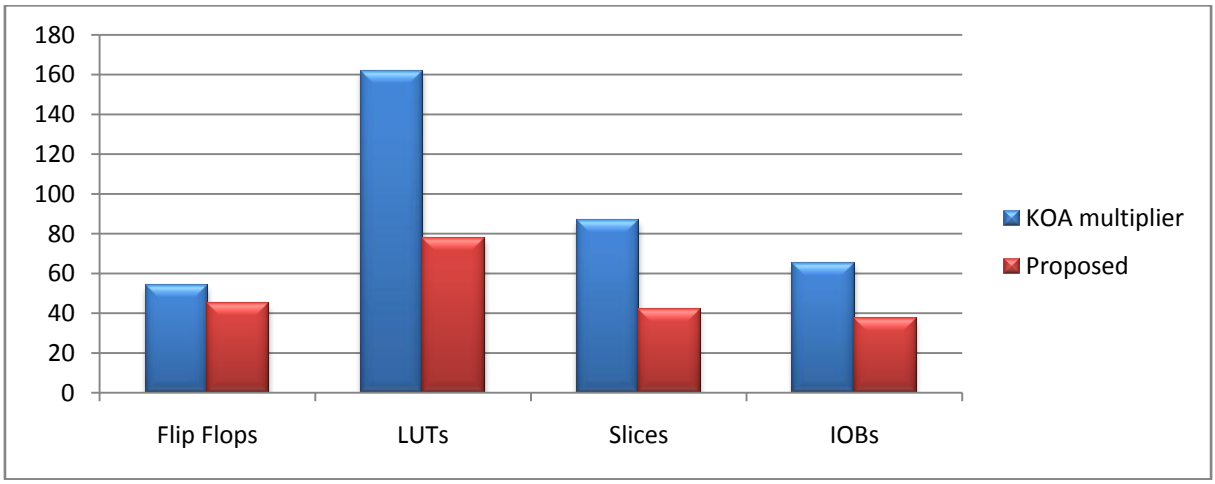

Fig 2. Resource utilization comparison of two KOA variants for L=16

Table 3. RESOURCE UTILIZATION BY MULTIPLIER FOR L=256

Device Utilization Summary			
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Flip Flops	530	178,176	1%
Number of 4 input LUTs	1,038	178,176	1%
Number of occupied Slices	529	89,088	1%
Number of Slices containing only related logic	529	529	100%
Number of Slices containing unrelated logic	0	529	0%
Total Number of 4 input LUTs	1,038	178,176	1%
Number of bonded IOBs	517	960	53%
Number of BUFG/BUFGCTRLs	1	32	3%
Number used as BUFGs	1		
Average Fanout of Non-Clock Nets	2.39		

This savior is more significant for L= 256 as suggested by area report . Proposed multiplier exhibits a savior of 52.49 % in terms of Flip flop slices. Proposed Multiplier involves 92 % fewer LUTs and utilizes 92% fewer slices as compared to traditional Karatsuba multiplier for 256 bit multiplication as indicated in table 3.

Area Report Of Cryptoprocessor

The ECC processor implementation uses a double point multiplication algorithm proposed in [9]. We have adapted revised and area optimized version of Montgomery's PRAC algorithm [10]. The area comparision is carried out for cryptoprocessors using proposed multiplier the one proposed in [11].

Table 4 below shows device utilization summary of Crypto processor, when the above proposed multiplier is employed for performing modular multiplications. Processor needs 376 slices out of 89,088 available slices in the target device. Among the 178,176 available four input LUTs only 573 are used. The multiplier also needs only 451 out of 178,176 Flip Flops.

Table 4. RESOURCE UTILIZATION BY ECC PROCESSOR

Device Utilization Summary			
Logic Utilization	**Used**	**Available**	**Utilization**
Total Number Slice Registers	367	178,176	1%
Number used as Flip Flops	365		
Number used as Latches	2		
Number of 4 input LUTs	573	178,176	1%
Number of occupied Slices	376	89,088	1%
Number of Slices containing only related logic	376	376	100%
Number of Slices containing unrelated logic	0	376	0%
Total Number of 4 input LUTs	574	178,176	1%
Number used as logic	512		
Number used as a route-thru	1		
Number used as Shift registers	61		
Number of bonded IOBs	451	960	46%
Number of BUFG/BUFGCTRLs	1	32	3%
Number used as BUFGs	1		
Number of DSP48s	30	96	31%
Average Fanout of Non-Clock Nets	1.15		

We have investigated area occupancy of cryptoprocessr built using proposed variant of Karatsuba Ofman multiplier and compared it with similar ECC work .Bar chart representation shown in figure 3 compares the resources utilized by proposed ECC with a similar work [11]. The comparison of implemented ECC processor employing proposed modular multiplier and the cryptoprocessor using traditional Karatsuba Ofman Multiplier with respect to the area occupied (Slice registers,Slices, LUTs and IOBs) is tabulated in Table 5 and compared in figure 3. Proposed implementation utilize about 23.93 % reduced slices and 62.65 % fewer LUTs. This is a significant achievement as we intend to put this ECC hardware for providing security services in embedded system which is known to be highly resources thirsty.

Table 5 RESOURCE UTILIZATION COMPARISON OF TWO ECC PROCESSORS

Work	*l*	Slices	LUTs	IOBs
[11]	193	466	932	932
Proposed	233	376	573	451

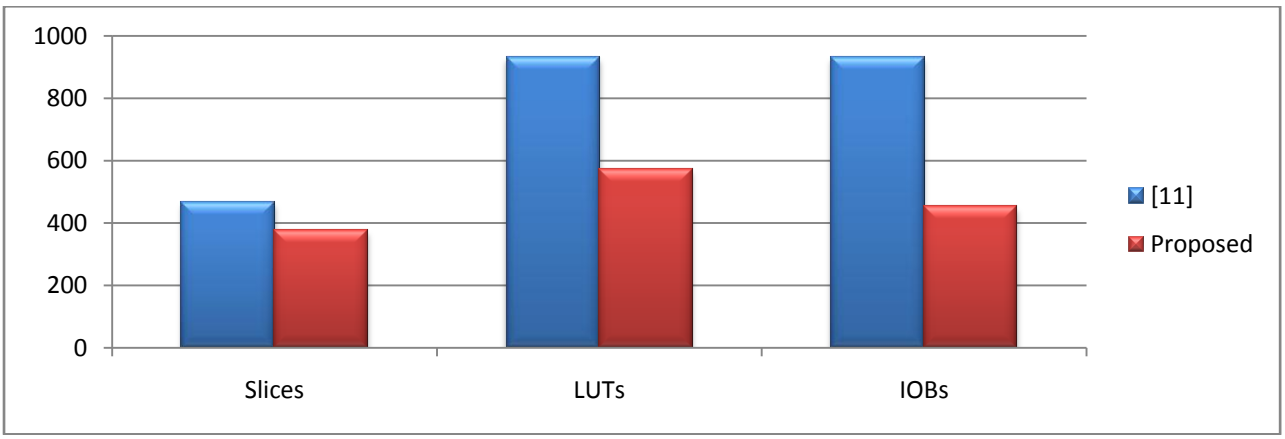

Fig 3. Resource utilization comparison of ECC processor using proposed multiplier and [11]

5. CONCLUSION

We have proposed a novel method to implement the modular multiplier for performing critical multiplication activities in ECC processor. Area occupancy of the resulting multiplier is found to be far superior than that of traditional KOA multiplier. This contributes in bringing down the footprint of entire ECC process. While injecting this modification, nowhere we have tampered with the basic flow and structure of basic ECC protocol. Just by processing the operands in different style, we have achieved area optimization of complete ECC protocol and made it suitable for embedded system with no compromise on security of ECC algorithm.

6. REFERENCES

[1] C. Paar. A new architecture for a parallel finite field multiplier with low complexity based on composite fields. IEEE Transactions on Computers, 45(7):856– 861, July 1996.

[2]Azarderakhsh. R, Jarvinen K.U and Mozaffari-Kermani. M,"Efficient algorithm and architecture for elliptic curve cryptography for extremely constrained secure applications", IEEE Transactions on circuits and systems- I , Vol. 61, No. 4, April 2014.

[3]Hossein Mahdizadeh and Massoud Masoumi, "Novel architecture for efficient FPGA implementation of elliptic curve cryptographic processor over GF(2^{163})", IEEE Transactions on very large scale integration (vlsi) systems, Vol. 21, NO. 12, pp: 2330- 2333, Dec.2013

[4]Sunil Devidas Bobade and Dr. Vijay R.Mankar," Low footprint Hybrid Finite field multiplier for Embedded cryptography", International Journal of Computer Science and Information Security(IJCSIS) , Vol. 13, No. 3, pp: 28- 32,March 2015.

[5] Sunil Devidas Bobade and Dr. Vijay R.Mankar," Space optimized Multiplier Architecture for Embedded cryptography", International Journal of Computer and Applications(IJCA) , Vol. 113, No. 14, pp: 26- 32,March 2015.

[6]Roy. S.S, Rebeiro,C and Mukhopadhyay. D, "Theoretical modeling of elliptic curve scalar multiplier on LUT-based FPGAs for area and speed", IEEE Transactions on Very Large Scale Integration (VLSI) systems, Vol. 21, No. 5, May 2013.

[7]Azarderakhsh. R, Jarvinen K.U and Mozaffari-Kermani. M,"Efficient algorithm and architecture for elliptic curve cryptography for extremely constrained secure applications", IEEE Transactions on circuits and systems- I , Vol. 61, No. 4, April 2014.

[8]Kazuo Sakiyama, Miroslav Knezevica, Junfeng Fana, , Bart Preneela, and Ingrid Verbauwhedea, "Tripartite modular multiplication", Integration, the VLSI Journal, Vol. 44, No.4, pp: 259–269, September 2011.

[9] Roy. S.S, Rebeiro,C and Mukhopadhyay. D, "Theoretical modeling of elliptic curve scalar multiplier on LUT-based FPGAs for area and speed", IEEE Transactions on Very Large Scale Integration (VLSI) systems, Vol. 21, No. 5, May 2013.

[10] R. Azarderakhsh and K. Karabina, " A New Double Point Multiplication Algorithm and its Application to Binary Elliptic Curves with Endomorphisms", IEEE Transactions on Computers, to appear:pp, 2013.

[11] A.Kaleel Rahuman and G.Athisha, "Reconfigurable Architecture for Elliptic Curve Cryptography Using FPGA", Hindawi Publishing Corporation Mathematical Problems in Engineering, 2013.