FPGA implementation of multipliers for ECC

Ravi Kishore Kodali, Prasanth Gomatam, Lakshmi Boppana
Department of Electronics and Communication Engineering
National Institute of Technology, Warangal
WARANGAL 506004,INDIA
E-mail: ravikkodali@gmail.com

***Abstract*—Scalar Multiplication(SM) is the most frequently used operation in Elliptic Curve Cryptography(ECC). The efficiency of an ECC based system depends on the efficient implementation of SM. The type of basis used while designing a cryptosystem determines the space and time complexities. We implemented two multipliers based on Optimal Normal Basis of type II(ONB) and polynomial basis. This work uses *Karatsuba* and *Sunar-Koc* algorithms. The hardware implementations of both the multipliers have been carried out for different key lengths: 243, 251, and 270 bits. The FPGA device used for hardware implementation is XC6VLX240T(Virtex-6). The synthesis results are compared qualitatively in terms of hardware complexities for these key lengths.**

key words-*Sunar-Koc* multiplier, *Karatsuba* multiplier, ECC.

I. INTRODUCTION

The arithmetic operations in Finite field play a crucial role in many Cryptographic computations. The complexity of operations depends on the basis used to represent the Finite field. There are many bases available in literature. Normal basis and Polynomial basis are most widely used among the bases by many applications.

Scalar Multiplication(SM) is the most frequently used finite field operation in Elliptic curve cryptography (ECC). The speed of the multiplier used determines the performance of the cryptosystem. The multipliers are implemented using Polynomial basis and Normal basis and *Karatsuba* and *Sunar-Koc* algorithms have been used.

Sunar-Koc multiplier is a conversion based ONB(type II) multiplier. The operands are initially converted from Normal basis to Canonical basis. The conversion process is called permutation and it requires no additional hardware. After the completion of multiplication the result is converted back to Normal basis. *Karatsuba* multiplier is a polynomial based multiplier. It involves repeated splitting of operands into several sub-operands. The details of both the multipliers are discussed comprehensively in further subsections.

The rest of the paper is organized as follows: Section II provides literature survey, section III gives an overview of *Sunar-Koc* multiplier and *Karatsuba* multiplier, section IV gives scheme of experimentation, section V presents results and section VI concludes the paper.

II. LITERATURE SURVEY

In the original *Karatsuba-Offman* algorithm[1], the operands are split into most significant, least significant halves and the product is calculated. During the summation of partial products overlapping takes place leading to the repetitive use of XOR gates. Instead, the operands are split based on their parity. Such a split-up results in non overlapping of XOR operations yielding improved time complexities [1].

Optimal normal basis (ONB) type II representation [2] is used to achieve reduced time complexities. The multipliers designed using ONB (type II) to achieve minimum propagation delays cannot be implemented using VLSI. Chang proposed a ONB type II multiplier, which makes use of matrix vector representation to compute partial products. This design can be implemented using VLSI. The depicted design finds its application in those systems with limited hardware resources.

Broadly, two types of multipliers making use of ONB [3] are found in literature: (i) lambda matrix based multipliers (ii) conversion based multipliers.

Somani and Amin [3] compared different kinds of lambda based multipliers and conversion based multipliers. They illustrated various ways of splitting the lambda matrix in order to achieve different space and time complexities.

Hasan modified the ONB representation [4] and developed a new architecture for *Massey-Omura* algorithm. The modified representation allows the partial products to be saved into a register resulting in reduced number of XOR gates.

The advantage of normal basis [5] over other bases is that a squaring operation can be carried out by making use of simple shifting operation. The usage of all one polynomial (AOP) makes a part of input independent of shifting operation, thereby attaining lesser time complexities.

The space occupied by ONB multiplier [6] grows exponentially with increased key lengths. Reed suggested the usage of standard basis pipelined multipliers for higher key lengths as the throughput of such multipliers is very high. However, this multiplier takes longer time to generate initial

978-1-4799-6986-9/14/$31.00©2014 IEEE

output and the delay depends on the width of operand.

The author [7] implemented arithmetic field operations using polynomial basis and optimal normal basis respectively. The ONB type II is used where higher performances are required at the cost of hardware and vice-versa. But, Kim [7] predicted that both the bases have identical performance levels by using projective co-ordinate system.

The array multiplier [8] consists of a linear feedback shift register (LFSR), general purpose register and an execution unit (EU). The EU consists of required hardware resources and a latch. The latch is used to synchronize the output and this output is passed onto the feedback register. The contents of the registers are the coefficients of polynomial basis representation of the multiplicand and multiplier operands. The design consumed lesser hardware by using the EU iteratively.

There are two kinds of side channel attacks on cryptographic systems: (i) Simple Power Analysis (SPA) attack (ii) Differential power analysis (DPA) attack.

The SPA attack involves extracting a single power trace of signals involved in computation. It is used to analyse [9] the operation involved in system. The DPA attack involves observation of a set of signals. They are classified into different groups and statistical studies are carried on them. The DPA attack is more dangerous compared to SPA due to its accurate estimate of the arithmetic operations involved. Both the attacks can be tackled by randomizing the scalar point involved in every scalar multiplication used by the crypto primitives. Apart from the above attacks, the timing attacks are also countered by randomizing scalar point yielding varying delays.

Applications like noise generation, ranging code generation, test data generation [10], require random sequence (RS) of longer length. The author [10] added Massy-Omura multiplier circuit to the maximum length shift register architecture in order to generate RS of longer lengths.

Scalar Multiplication can be achieved [11] by a sequence of point addition and point doubling operations. Mishra used a two stage pipeline to perform the same. Mishra used Jacobian co-ordinate system and calculated the time complexities based on the individual blocks involved in the pipeline stages.

In general, elliptic curve scalar multiplication is done by performing point addition (ECAD) and doubling (ECDBL) operations [12] repeatedly. ECAD and ECDBL operations can be performed by a sequence of finite field (FF) operations like addition, squaring, multiplication and inversion. The time complexities for various cases of Montgomery multiplication have been studied [12].

III. MATHEMATICAL BACKGROUND

A. KARATSUBA MULTIPLIER

The operands used in *Karatsuba* Algorithm are the elements in GF(2^m). They are represented using polynomial basis as given by equation (1).

$$\boxed{A = \sum_{j=0}^{m-1} a_j x^j = X^{\frac{m}{2}} A^H + A^L,} \tag{1}$$

where

$$A^H = \sum_{i=0}^{\frac{m}{2}-1} a_{(i+\frac{m}{2})} x^i$$

Similarly, the other operand B can be represented using equation (2).

$$\boxed{B = \sum_{j=0}^{m-1} b_i x^j = X^{\frac{m}{2}} B^H + B^L,} \tag{2}$$

where

$$B^H = \sum_{j=0}^{(\frac{m}{2}-1)} b_{(j+\frac{m}{2})} x^j$$

The resultant product C is computed using equation (3).

$$\begin{aligned} C = X^m A^H B^H + A^L B^L + (A^H B^H + A^L B^L + \\ (A^H + A^L (B^H + B^L))) X^{\frac{m}{2}} \end{aligned} \tag{3}$$

Using recursion, each sub-product is divided further into sub-parts. This process of division continues till the lowest possible level, where multiplication can be easily computed using conventional algorithms, such as shift and add algorithm, Booth's multiplication algorithm, etc.

B. Sunar-Koc Multiplier

The operands in canonical form are shown below:

$$A = a_1\lambda + a_2\lambda^2 + a_3\lambda^{2^2} + ... + a_m\lambda^{2^{m-1}} \tag{4}$$

$$B = b_1\lambda + b_2\lambda^2 + b_3\lambda^{2^2} + ... + b_m\lambda^{2^{m-1}} \tag{5}$$

The three stages involved in *Sunar-Koc* algorithm are: (i) Permutation (ii) Multiplication (iii) Re-permutation.

Permutation:

The operands are initially in canonical basis. They are converted into Normal basis using the below mentioned scheme.

The coefficients in the above equation altogether form a key word.

$$b_i = b_j, \tag{6}$$

$$\text{where } i = \begin{cases} k, & k \in [1, m] \\ (2m+1) - k, & k \in [m+1, 2m] \end{cases}$$

$$\text{and } k = 2^{j-1} \mod (2m+1)$$

A similar scheme is followed for the other operand. The key words after the permutation are shown here.

The key words after the permutation are listed below:

$$A = \sum_{i=1}^{m} (a_i \lambda_i) \tag{7a}$$

$$B = \sum_{i=1}^{m} (b_i \lambda_i) \tag{7b}$$

Multiplication:

The product is computed by summing the following three sub-products.

$$D = \sum_{i=1}^{m} \sum_{j=1}^{m} a_i b_j (\gamma^{(i-j)} + \gamma^{-(i-j)}) \tag{8a}$$

$$E = \sum_{i=1}^{m} \sum_{j=1}^{m-i} a_i b_j (\gamma^{(i+j)} + \gamma^{-(i+j)}) \tag{8b}$$

$$F = \sum_{i=1}^{m} \sum_{j=(m-i+1)}^{m} a_i b_j (\gamma^{(i+j)} + \gamma^{-(i+j)}) \tag{8c}$$

Re-permutation:

Using the same scheme as in permutation the product is converted back to canonical basis.

IV. SCHEME OF EXPERIMENTATION

We have developed VHDL models for both *Karatsuba* and *Sunar-Koc* multipliers respectively. *Karatsuba* and *Sunar-Koc* multipliers have been implemented on the *XILINX* FPGA VIRTEX family device. Hardware complexities were measured for both the algorithms in terms of device utilization. VHDL models have beeb developed for three different key lengths for both the multipliers. The key lengths are 243-bit, 251-bit and 270- bit. The output simulations have also been included in the next section. The algorithms used in the architectures of the multipliers are illustrated here.

A. KARATSUBA MULTIPLIER

Using Algorithm -1, the operands are split into the available multiplier blocks at lower level. Algorithm -2 is made use of in the multiplier block.

B. SUNAR-KOC MULTIPLIER

Sunar algorithm consists of three phases. They are permutation, multiplication and re-permutation. Three separate VHDL modules have been developed for the three phases. The permutation and re-permutation operations are used to convert operands from Normal basis to canonical basis and canonical basis to Normal basis respectively. The algorithm involved in Multiplication phase is illustrated below.

Algorithm 1 METHOD FOR SPLITTING IN *KARATSUBA* ALGORITHM

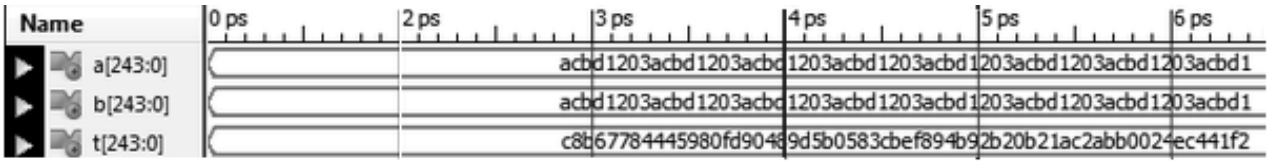
INPUT:Input in polynomial basis
OUTPUT:Output in polynomial basis
BEGIN K
$t = a'length$;
if $t = 2$ or $t = 3$ **then**
  $multi(a, b)$
end if
$z0 := k(a1, b1)$;
$z4 := a1$;
$z5 := b1$;
$z1 := k((a1 \text{ xor } z4), (b1 \text{ xor } z5))$;
$z2 := k(a1, b1)$;
$z6 := (z2 \text{ xor } z1 \text{ xor } z0)$;
$endk$;

Algorithm 2 METHOD FOR MULTIPLICATION IN KARATSUBA ALGORITHM

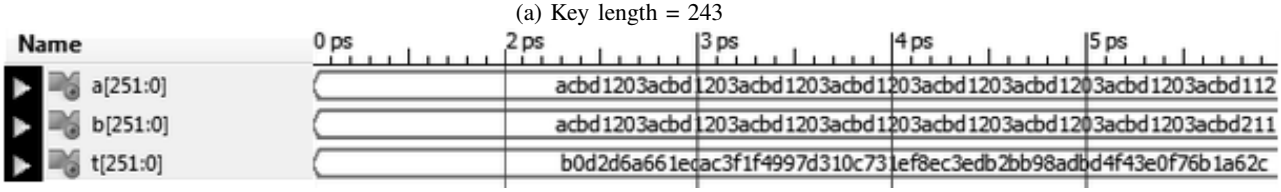
INPUT:Input operands
OUTPUT:Resultant product
BEGIN multi
for $i = 0$ to $b'length - 1$ **do**
  if $b1(i) = 1$ **then**
    $s2 := s2$ XOR $s1$;
  end if
  $s1 := s1((s1'length - 2) downto 0)$
end for
return multi

Algorithm 3 ALGORITHM FOR MULTIPLICATION PROCESS

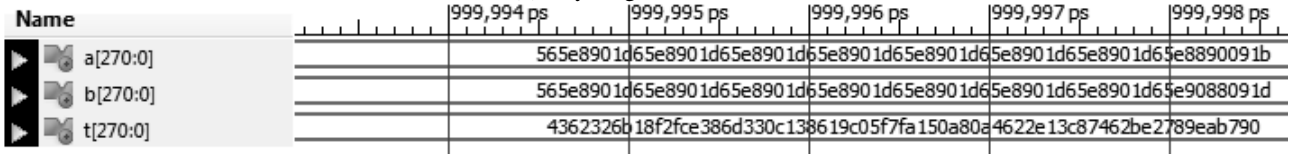
INPUT: Input operands in Canonical basis
OUTPUT: Resultant product in Canonical basis
for $j = 1$ to m **do**
  for $i = 1$ to $m - i$ **do**
    $c(j) := (a(i) and b(j+i)) xor (a(j+i) and b(i)) xor c(j)$
  end for
end for
for $j = 2$ to m **do**
  for $l = 1$ to $k - 1$ **do**
    $d(k) := (a(l)\ and\ b(k-l))\ xor\ d(k)$
  end for
end for
for $j = 1$ to m **do**
  for $i = m - i + 1$ to m **do**
    $e(m) := (a(n)\ and\ \ b(346 - n - m + 1))\ xor\ e(m)$
  end for
end for
for $j = 1$ to m **do**
  $p(u) := c(u)\ xor\ d(u)\ xor\ e(u)$
end for



(a) Key length = 243

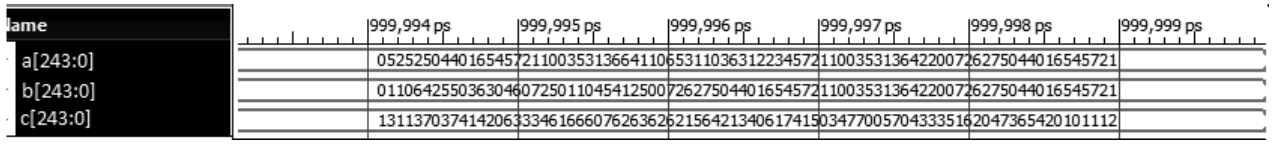


(b) Key length = 251

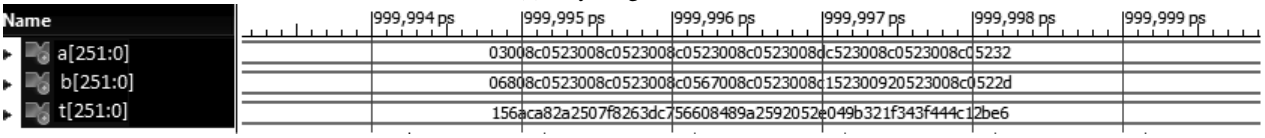


(c) Key length = 270

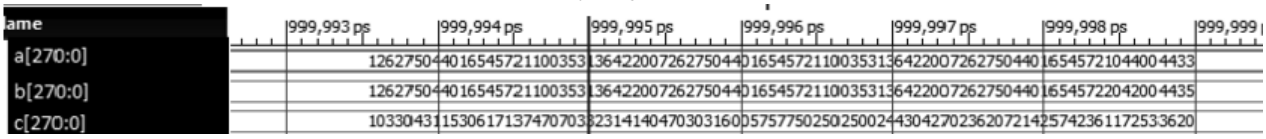
Fig. 1: *Sunar-Koc* algorithm Simulation results for three different key lengths



(a) Key length = 243



(b) Key length = 251



(c) Key length = 270

Fig. 2: *Karatsuba* algorithm Simulation results for three different key lengths

V. SYNTHESIS AND SIMULATION RESULTS

A. SYNTHESIS RESULTS

We have used Xilinx 13.4 to implement the designed multipliers. The simulations for different key lengths have been illustrated here.

B. SYNTHESIS RESULTS FOR SUNAR-KOC MULTIPLIER

The Sunar multiplier has been designed based on the algorithm described in the previous section and the same has been implemented on XC6VLX240T device. The implementation results are illustrated here.

TABLE I: DEVICE UTILIZATION FOR *SUNAR-KOC* MULTIPLIER

Parameter		243-bit	251-bit	270-bit
Logic Utilization	Available	Used	Used	Used
No. of Slice LUT'S	150720	82863	88008	101957
No. of Slice FF-pairs	76055	0	0	0
No. of IOB's	600	730	754	811

C. SYNTHESIS RESULTS FOR KARATSUBA MULTIPLIER

The implementation of *Karatsuba* multiplier has also been carried out on the XC6VLX240T device and the synthesis results are shown here.

A comparison of both the multipliers in terms of hardware complexities is given by the histogram as shown in figure 3.

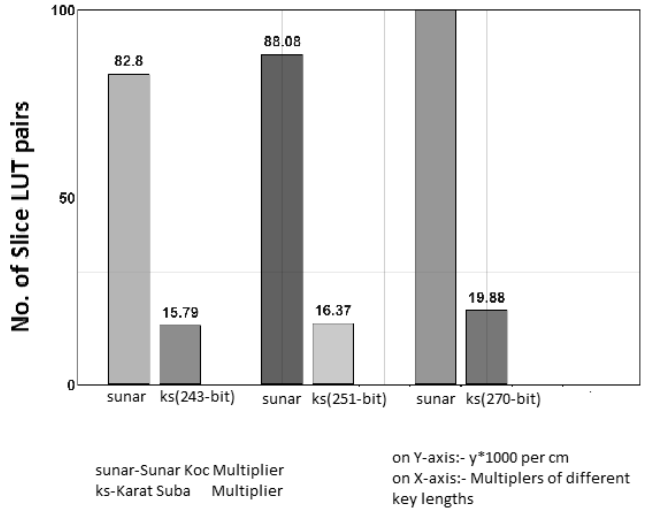


Fig. 3: Comparison of hardware complexities of *Sunar-Koc* and *Karatsuba* multipliers

TABLE II: DEVICE UTILIZATION FOR *KARATSUBA* MULTIPLIER

Parameter		243-bit	251-bit	270-bit
Logic Utilization	Available	Used	Used	Used
No. of Slice LUT'S	150720	15790	16378	19882
No. of Slice FF'Pairs	76055	0	0	0
No. of IOB's	600	656	645	752

It is observed that the *Karatsuba* multiplier consumed lesser resources. *Sunar-Koc* is a parallel multiplication algorithm where as *Karatsuba* makes use of the available resources in an iterative manner. Hence, *Karatsuba* multiplier consumed lesser resources than *Sunar-Koc* multiplier.

VI. CONCLUSIONS

There are different bases available in literature for finite field representation. We have chosen Polynomial basis and Normal basis representation during the implementations of the multipliers. The implementation has been carried out on FPGA device. The device used is XC6VLX240T-ff1156. The multipliers are compared qualitatively in terms of device utilization. Our future work involves an implementation of a Cryptographic processor using these algorithms.

REFERENCES

[1] H. Fan, J. Sun, M. Gu, and K.-Y. Lam, "Overlap-free karatsuba-ofman polynomial multiplication algorithms," *Information Security, IET*, vol. 4, no. 1, pp. 8–14, 2010.

[2] C.-Y. Lee and C.-J. Chang, "Low-complexity linear array multiplier for normal basis of type-ii," in *Multimedia and Expo, 2004. ICME'04. 2004 IEEE International Conference on*, vol. 3. IEEE, 2004, pp. 1515–1518.

[3] T. F. Al-Somani and A. Amin, "Hardware implementations of gf (2m) arithmetic using normal basis," *Journal of Applied Sciences*, vol. 6, no. 6, pp. 1362–1372, 2006.

[4] A. Reyhani-Masoleh and M. A. Hasan, "A new construction of massey-omura parallel multiplier over gf (2¡ sup¿ m¡/sup¿)," *Computers, IEEE Transactions on*, vol. 51, no. 5, pp. 511–520, 2002.

[5] M. A. Hasan, M. Wang, and V. K. Bhargava, "A modified massey-omura parallel multiplier for a class of finite fields," *Computers, IEEE Transactions on*, vol. 42, no. 10, pp. 1278–1280, 1993.

[6] I.-S. Hsu, T.-K. Truong, L. J. Deutsch, and I. S. Reed, "A comparison of vlsi architecture of finite field multipliers using dual, normal, or standard bases," *Computers, IEEE Transactions on*, vol. 37, no. 6, pp. 735–739, 1988.

[7] Y.-J. Choi, M.-S. Kim, H.-R. Lee, and H.-W. Kim, "Implementation and analysis of elliptic curve cryptosystems over polynomial basis and onb," in *Proceedings of World Academy of Science, Engineering and Technology*, vol. 10. Citeseer, 2005.

[8] C. Chiou, L. Lin, F. Chou, and S. Shu, "Low-complexity finite field multiplier using irreducible trinomials," *Electronics Letters*, vol. 39, no. 24, pp. 1709–1711, 2003.

[9] J. C. Ha and S. J. Moon, "Randomized signed-scalar multiplication of ecc to resist power attacks," in *Cryptographic hardware and embedded systems-CHES 2002*. Springer, 2003, pp. 551–563.

[10] C. C. Wang and D. Pei, "A vlsi design for computing exponentiations in gf (2¡ sup¿ m¡/sup¿) and its application to generate pseudorandom number sequences," *Computers, IEEE Transactions on*, vol. 39, no. 2, pp. 258–262, 1990.

[11] P. Mishra, "Pipelined computation of scalar multiplication in elliptic curve cryptosystems (extended version)," *Computers, IEEE Transactions on*, vol. 55, no. 8, pp. 1000–1010, 2006.

[12] B. Ansari and M. A. Hasan, "High-performance architecture of elliptic curve scalar multiplication," *Computers, IEEE Transactions on*, vol. 57, no. 11, pp. 1443–1453, 2008.