# Implementations of *Sunar-Koc* Multiplier using FPGA platform and WSN node

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

***Abstract*—In elliptic curve cryptography (ECC), multiplication operations are used frequently. In order to realize an efficient ECC implementation for large key lengths, it is necessary to choose an algorithm using which it is possible to compute these multiplication operations at higher speeds. This work presents two different implementations of *Sunar-Koc* multiplier, using FPGA device and a WSN node. This work considered the key lengths 173- bit, 194- bit and 233- bit in both the FPGA and WSN node implementations of the multiplier. A MEMSIC IRIS WSN node has been used during the implementation and a resource comparison, comprising of storage requirements, energy consumption and clock cycles for different key lengths, is made. The obtained FPGA synthesis results have also been compared.**

key words- Sunar-Koc multiplier, WSN, ECC, FPGA

## I. Introduction

Elliptic curve cryptography(ECC) requires various elliptic curve (EC) operations, like point addition, scalar and point multiplications, and inversion. Among these operations, scalar and point multiplications play a major role in an efficient implementation of ECC, as this multiplication operation is performed many times. The computational speed of this operation and power consumed are of utmost importance. This work highlights the implementations of *Sunar-Koc* algorithm [1] using both FPGA device and Wireless sensor network (WSN) device. In certain WSN applications requiring security, especially using ECC, efficient multiplication algorithms are needed. However, timing and energy constraints decide the algorithm that is to be used in practical situations [2].

The *Karatsuba* algorithm consumes less hardware and consumes more time, whereas another one, the *Sunar-Koc* algorithm requires more hardware but consumes less computational time. In the *Karatsuba* algorithm, the multiplication of a large key lengths is subdivided till the bottom most level of the key length, at which multipliers are readily available to perform the required operations. Another multiplication algorithm, *Massey-Omura* multiplier, whose time complexities are almost similar to those of the *Sunar-Koc* multiplier. However, the *Massey-Omura* multiplier requires $2(m^2 - m)$ X-or gates, whereas *Sunar-Koc* multiplier requires $1.5(m^2 - m)$ X-or gates. The rest of the paper is organized as follows: Section II provides literature survey, section III presents an overview of *Sunar-Koc* multiplier, section IV gives scheme of experimentation, section V presents simulation and experimental results and section VI concludes the work.

## II. Literature Review

Point addition and point doubling constitute important operations in ECC. These operations can be carried out by a sequence of finite field (FF) operations, like addition, squaring, multiplication and inversion. The multiplication, using Montgomery algorithm [3] over the prime field, $(GF(2^m)$, consumes $(m/w) + c$ clock cycles, where w is the key length. The number of clock cycles, $c$, can be reduced by not pushing multiplier into an idle state. In *Montgomery* algorithm, $(6m + 3a + 5s)$ number of clock cycles are used, where m, a, and s are the clock signals used for multiplication,addition and squaring, respectively. In order to reduce the number of clock cycles, it is assumed that multiplication takes more time in comparison to addition and squaring. It facilitates parallel execution of addition and squaring operations, so that the number of clock cycles required becomes $6m + a$. Also comparison in terms of clock cycles and throughput per LUT's is given in [3].

Traditional double and add algorithm (DAA) requires evaluation of point addition(add) and point doubling(double) operations. Another operation, point quadruple(quad) has been introduced [4] and this operation comprises of various finite field (FF) operations, like multiplication,division squaring and addition.

Three instructions, multiplication and addition (MULAD), multiplication and squaring (MUSQ), and repeated squaring (RESQ) are introduced in [5]. A sequence of nine arithmetic instructions have been developed to perform point addition and point doubling operations, using these instructions. The multiplier block in the data path alone accounted for 95 $\%$ of the area.

The costs of point doubling and point addition operations for various coordinate systems are compared in [6]. Various separate multiplier blocks have been used for point addition and point doubling operations so that complications related to timing, placement and routing can be reduced [7]. Wireless sensor networks(WSN's) consist of a base station (BS) and number of nodes. There are various parameters like encryption, decryption, signature verification in public key cryptography (PKC). In ECC based scheme [8], two scalar multiplications are required for encryption and signature verification, whereas only one multiplication is required for decryption and signature generation [8]. The hybrid key establishment protocol

978-1-4799-2827-9/13/$31.00 ©2013 IEEE

presented in[9] uses ECC based key calculations. The protocol demands exchange of 6 messages between sensor and a security manager. There are two kinds of optimal normal basis [10] multipliers: $\lambda$ based and conversion based multipliers. The $\lambda$ matrices used in these multipliers are constructed for GF($2^m$). In conversion based multipliers, the basis of multipliers is converted from canonical basis to normal basis, which simplifies the computation. The conversion multipliers are more efficient than $\lambda$ based multipliers, since $\lambda$ multipliers need to store the $\lambda$ matrix.

The optimal normal basis of type II is used in *Sunar-Koc* multiplier [11]. The *Sunar-Koc* multiplier [12], is a parallel architecture, whose time complexities are almost similar to those of *Massey-Omura*, consuming less hardware. Various multipliers like hybrid *Karatsuba*, *Sunar-Koc* and *Massey-Omura* multipliers [13] are compared in terms of speed and device utilization. In many applications of ECC, the scalar multiplication [14] along with inversion are used. *Itoh-Tshuji* algorithm for inversion is difficult to implement on hardware.

## III. An Overview of Sunar-Koc Multiplier

The arithmetic field GF($2^m$) is an m- dimensional vector space, in which m linearly independent vectors are chosen to serve the basis notation. There are two kinds of basis:

1) Canonical basis: The ordered set, $(1, \beta, \beta^2, ....., \beta^{m-1})$ in which $\beta \in GF(2^m)$.
2) Normal basis: It is the set, M given in equation 1 and $\beta$ is its normal element.

$$M = \beta, \beta^2, \beta^4, ....., \beta^{2^{m-1}} \tag{1}$$

The optimal normal basis of type II can be constructed using a normal element, $\beta$, such that, $\beta = \gamma + \gamma^{-1}$ [1]. The $\gamma$ is to be selected in such a way that it is a primitive root of unity i.e. $\gamma^{2m+1} = 1$. The construction of optimal normal basis of type II is possible only if $p = 2m + 1$, is a prime number and the following two conditions are met: *a*) 2 = a primitive root mod p; and *b*) The multiplicative order of 2 mod p = m. Figure 1 shows the following three phases of the *Sunar-Koc* algorithm: *i*) permutation (perm); *ii*) multiplication (mult) and *iii*) reverse permutation (reperm)

### A. Phase-1: Permutation of the key words

The operation of Phase-1 in Figure 1 is as follows:

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

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

### B. Phase-2: Multiplication Algorithm

The key words after the permutation are:
$A = \sum_{i=1}^{m} (a_i \beta_i)$ and $B = \sum_{i=1}^{m} (b_i \beta_i)$.
And the product of the two keywords, A and B is $C = A.B$

The product, C, can be further divided into three sub-products: D, E and F , which are given by equation (2).

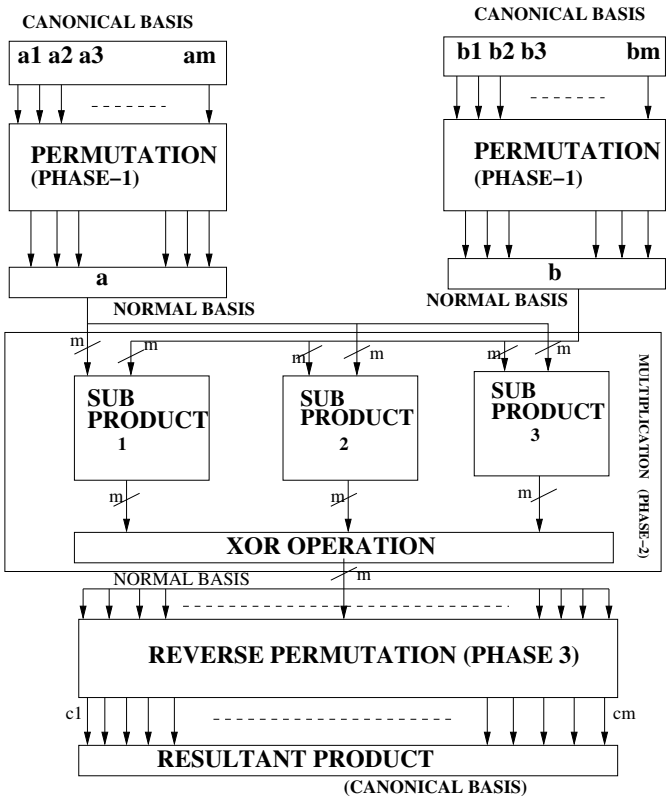



Fig. 1: Block Diagram of Sunar-Koc multiplier

$$D = \sum_{i=1}^{m} \sum_{j=1}^{m} a_i b_j (\gamma^{i-j} + \gamma^{-(i-j)}) \tag{2a}$$

$$E = \sum_{i=1}^{m} \sum_{j=1}^{m-i} a_i b_j (\gamma^{i+j} + \gamma^{-(i+j)}) \tag{2b}$$

$$F = \sum_{i=1}^{m} \sum_{j=m-i+1}^{m} a_i b_j (\gamma^{i+j} + \gamma^{-(i+j)}) \tag{2c}$$

By summing all these three sub-products, C is obtained. In phase-3 as given in Figure 1, the resultant product, C, which is in Normal basis form and it is converted by using the reverse permutation, similar to permutation phase.

## IV. Scheme of Experimentation

The scalar multiplication operation plays a major role in ECC. This work makes use of *Sunar-Koc* algorithm to carry out scalar multiplication. An implementation and performance analysis of the *Sunar-Koc* algorithm has been carried out using WSN node and FPGA device.

### A. Hardware Implementation

In the FPGA prototyping of the *Sunar-Koc* algorithm has been carried out on Xilinx Virtex-6 family device, xc6vlx240t-2ff1156 using Xilinx design suite 13.2. Three modules for permutation, multiplication, reverse permutation phases have

**Algorithm 1** Algorithm for Permutation Phase

**INPUT**:Input in canonical basis before permutation
**OUTPUT**:Output in normal basis after permutation
**for** $i = 1$ to $m$ **do**
  **if** $temp > m$ **then**
    $index(i) := (2m + 1) - temp$
  **else**
    $index(i) := temp$
  **end if**
**end for**

been developed using VHDL. Algorithm 1 gives the steps used during the permutation phase.

Algorithm 2 provides the steps to be carried out during the multiplication phase. The resultant product from the multiplication phase is mapped from normal basis to canonical basis, during the reperm phase of the algorithm.

**Algorithm 2** Multiplication Algorithm

**INPUT**: Input in normal basis
**OUTPUT**: Output in normal basis
**for** $i = 1$ to $m$ **do**
  **for** $j = 1$ to $m - i$ **do**
    $c(i) := (a(j)andb(j+i))xor(a(j+i)andb(j))xorc(i)$
  **end for**
**end for**
**for** $i = 2$ 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** $i = 1$ to $m$ **do**
  **for** $j = m - i + 1$ to $m$ **do**
    $e(m) := (a(n)\ and\ \ b(m - n + 1))\ xor\ e(m)$
  **end for**
**end for**
**for** $i = 1$ to $m$ **do**
  $p(u) := c(u)\ xor\ d(u)\ xor\ e(u)$
**end for**

### B. WSN Implementation

Our aim is to implement scalar multiplication using a WSN node. The nesC code consists of three functions called perm(), mult(), reperm(). During the application development in the implementation of the algorithm using nesC, the constraints of the IRIS node have been considered. The interfaces required for communication between the nodes have been modified. The nodes have been programmed using the *Moteworks* interface. As soon as the node completes execution it starts transmitting the result. The received data by another node is captured using *Xsniffer* software.

In order to improve the performance, m (where m is key length) number of variables are not used, while developing the nesC code. Instead, $m/w$ (where w is the size of variable's data type in terms of bits) number of variables are being used. A separate function, based on the data type of a variable, is used to retrieve bits from the variable. The algorithm 3 illustrates this function.

**Algorithm 3** Algorithm for extracting a bit from a variable

power=1
$temp1 = e\%w$
$temp = e/16$
**if** $temp1! = 1$ **then**
  **for** $i = 1$ to $temp1 - 1$ **do**
    $power = power * 2$
  **end for**
**else**
  power=1
**end if**
result=result and e
return result

Three different data sets for each of the key lengths, 173- bit, 194- bit and 233- bit, have been used to verify the consistency of the result.

## V. Results and Simulation

The simulation results for three key lengths, 173- bit, 194- bit, and 233- bit, are given in Figure 2. The synthesis results for the key lengths have been obtained.

### A. FPGA Implementation Results

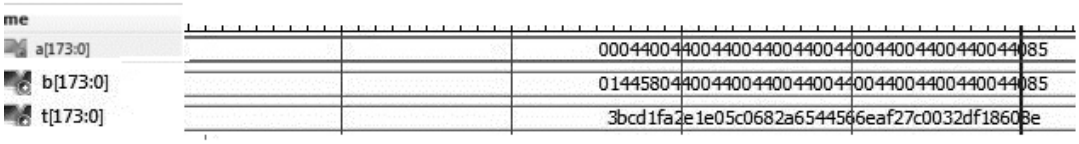
*1) SYNTHESIS RESULTS ON HARDWARE:* The *Sunar-Koc* algorithm is synthesized using Xilinx tool on Virtex-6 device and synthesis results of various key lengths are summarised in Table I. Due to the requirement of higher number of X-or, And, Or gates for increased key lengths, the amount of resources(LUT's) required by the multiplier has also increased. The number of IOB's does not effect the performance of the multiplier. In the case of 233- bit multiplier number of IOB's crossed the resource capability of the device. Nearly 90% of the delay in the three multipliers(173 bit,194-bit and 233 bit)is due to routing within the FPGA device.

TABLE I: FPGA Synthesis results for different Key lengths

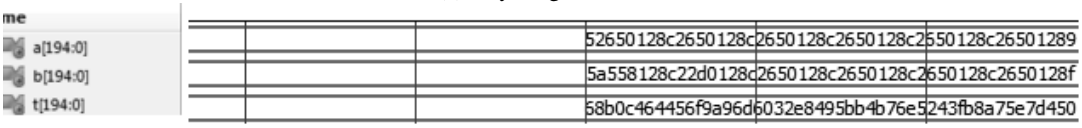
| Parameter | | Key length | | |
|---|---|---|---|---|
| | | 173- bits | 194- bits | 233- bits |
| Logic Utilization | Avail-able | Used | Used | Used |
| No. of Slice-LUT's | 150720 | 42142 | 52829 | 76055 |
| No. of Slices-FF pairs | 76055 | 0 | 0 | 0 |
| No. of IOB's | 600 | 520 | 583 | 700 |

### B. Implementation Results on WSN node

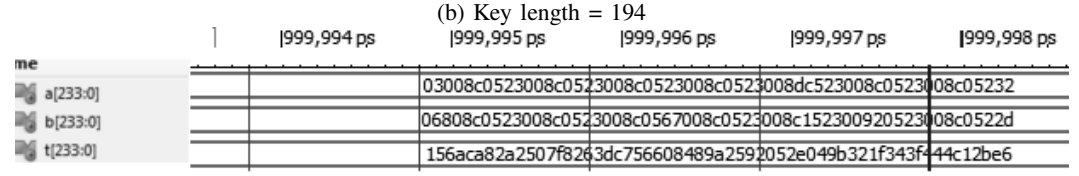
The parameters that have been taken into consideration are memory occupied with interfaces, without interfaces, number of clock cycles completed in executing the multiplication and transmitting it and power consumed by processor. With an




(a) Key length = 174




(b) Key length = 194




(c) Key length = 233

Fig. 2: Simulation results for three different key lengths

increase in key length the memory requirements (both ROM and RAM) of node, the number of clock cycles has also increased. The power consumed is constant due to the mode of the WSN node.

TABLE II: Implementation results on WSN node for different Key lengths

| Parameter | | Key length | | |
|---|---|---|---|---|
| | | 173- bits | 194- bits | 233- bits |
| Memory w/o interfaces | ROM | 2350B | 2390B | 2424B |
| | RAM | 151B | 155B | 163B |
| Memory with interfaces | ROM | 55310B | 55350B | 56714B |
| | RAM | 2015B | 2023B | 1983B |
| Number of Clock cycles | | 163 | 203 | 291 |
| Time Taken | | 0.6367 s | 0.7929 s | 1.136 s |
| Energy (mJ) | | 24 | 24 | 24 |

## VI. Conclusions

The *Sunar-Koc* algorithm is implemented on a WSN device, which has an 8- bit processor. The work has demonstrated the implementation of *Sunar-Koc* multiplier on a resource constrained WSN node for higher key lengths and based on the results this algorithm can be used during the ECC implementations in WSN applications providing security. An FPGA implementation of the algorithm has been carried out for different key lengths.

## References

[1] B. Sunar and C. Koc, "An efficient optimal normal basis type ii multiplier," *Computers, IEEE Transactions on*, vol. 50, no. 1, pp. 83–87, 2001.

[2] N. Gura, A. Patel, A. Wander, H. Eberle, and S. Shantz, "Comparing elliptic curve cryptography and rsa on 8-bit cpus," *Cryptographic Hardware and Embedded Systems-CHES 2004*, pp. 925–943, 2004.

[3] 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.

[4] S. Moon, "A binary redundant scalar point multiplication in secure elliptic curve cryptosystems." *IJ Network Security*, vol. 3, no. 2, pp. 132–137, 2006.

[5] W. N. Chelton and M. Benaissa, "Fast elliptic curve cryptography on fpga," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 16, no. 2, pp. 198–205, 2008.

[6] M. Bednara, M. Daldrup, J. Teich, J. von zur Gathen, and J. Shokrollahi, "Tradeoff analysis of fpga based elliptic curve cryptography," in *Circuits and Systems, 2002. ISCAS 2002. IEEE International Symposium on*, vol. 5. IEEE, 2002, pp. V–797.

[7] C. Shu, K. Gaj, and T. El-Ghazawi, "Low latency elliptic curve cryptography accelerators for nist curves over binary fields," in *Field-Programmable Technology, 2005. Proceedings. 2005 IEEE International Conference on*. IEEE, 2005, pp. 309–310.

[8] G. Gaubatz, J.-P. Kaps, E. Ozturk, and B. Sunar, "State of the art in ultra-low power public key cryptography for wireless sensor networks," in *Pervasive Computing and Communications Workshops, 2005. PerCom 2005 Workshops. Third IEEE International Conference on*. IEEE, 2005, pp. 146–150.

[9] Q. Huang, J. Cukier, H. Kobayashi, B. Liu, and J. Zhang, "Fast authenticated key establishment protocols for self-organizing sensor networks," in *Proceedings of the 2nd ACM international conference on Wireless sensor networks and applications*. ACM, 2003, pp. 141–150.

[10] 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.

[11] 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.

[12] B. Sunar and C. K. Koç, "An efficient optimal normal basis type ii multiplier," *Computers, IEEE Transactions on*, vol. 50, no. 1, pp. 83–87, 2001.

[13] C. Grabbe, M. Bednara, J. Teich, J. von zur Gathen, and J. Shokrollahi, "Fpga designs of parallel high performance gf (2) multipliers [cryptographic applications]," in *Circuits and Systems, 2003. ISCAS'03. Proceedings of the 2003 International Symposium on*, vol. 2. IEEE, 2003, pp. II–268.

[14] Q. Deng, X. Bai, L. Guo, and Y. Wang, "A fast hardware implementation of multiplicative inversion in gf (2)," in *Microelectronics & Electronics, 2009. PrimeAsia 2009. Asia Pacific Conference on Postgraduate Research in*. IEEE, 2009, pp. 472–475.