American Journal of Applied Sciences 6 (8): 1596-1603, 2009
ISSN 1546-9239

Reconfigurable Elliptic Curve Crypto-Hardware Over the Galois Field GF(2^{163})

[1]Mohamed Abdelkader Bencherif, [2]Hamid Bessalah and [1]Abderrezak Guessoum
[1]Department of Electronic, Saad Dahlab University, Road of Soumaa, Blida
[2]Advanced Technologies Development Center, Baba Hassen, Algeria

Abstract: Problem statement: In the last decade, many hardware designs of elliptic curves cryptography have been developed, aiming to accelerate the scalar multiplication process, mainly those based on the Field Programmable Gate Arrays (FPGA), the major issue concerned the ability of embedding this strategic and strong algorithm in a very few hardware. That is, finding an optimal solution to the one to many problem: Portability against power consumption, speed against area and maintaining security at its highest level. Our strategy is to hardware execute the ECC algorithm that reposes on the ability of making the scalar multiplication over the GF(2^{163}) in a restricted number of clock cycles, targeting the acceleration of the basic field operations, mainly the multiplication and the inverse process, under the constraint of hardware optimization. **Approach:** The research was based on using the efficient Montgomery add and double algorithm, the Karatsuba-Offman multiplier and the Itoh-Tsjuii algorithm for the inverse component. The hardware implementation was based upon an optimized Finite State Machine (FSM), with a single cycle 163 bits multiplier and a script generated field squarer. The main characteristics of the design concerned the elimination of the different internal component to component delays, the minimization of the global clocking resources and a strategic separation of the data path from the control part. **Results:** The working frequency of our design attained the 561 MHz, allowing 161786 scalar multiplications per second, outperforming one of the best state of the art implementations (555 MHz); the other contribution concerns the acceleration of the field inverse scheme with a frequency of 777.341 MHz. **Conclusion:** The results indicated that using different optimizations at the hardware level improve efficiently the acceleration of the ECC scalar multiplication and the choice of the target circuit gratefully enhances propagation delays and increases frequency.

Key words: Elliptic curves cryptography, ECC, FPGA, Montgomery, Galois field operations

INTRODUCTION

In the last decade, the approach of hardware implementing Elliptic Curve Cryptography algorithms (ECC) knew a very intensive race, due essentially to the requirements of security, speed and area constraints. In fact, security deals mainly with the ability to face counter-attacks[1], while speed and area which represent the eternal trade-off, that concern the ability to make intensive cryptographic processes, while keeping used hardware as low as possible. In other words, it is the ability of embedding a strategic and strong algorithm in a very few hardware. That is, finding an optimal solution to the one to many problem: Portability against power consumption, speed against area, but the main issue in cryptography is security.

Cryptography has become one of the most important fields in our life, due essentially to two factors, increase in secrecy and increase in breaking code or hackers in the other side. It is no more safe to use its birth date or the name of its child, as a common password in some banking or even mailing accounts.

Organizations tend to increase their benefits by keeping their information system as transparent as possible. On the other hand hackers and code or key breakers are being organized in a kind of unofficial groups; this leads to being a step ahead before getting the codes breakdown.

Scientists are tending to complicate the reverse engineering process of the encryption system, at the same time, keeping encryption keys as low as possible. This issue is being tackled by many mathematics, mainly those working on elliptic curves[2].

The beauty of this new field is potentially related to the simplicity of the operators used in the encryption process, to the non-secure transmission constraints used in the exchange of the keys and to the enhanced complexity that might face hackers when unwanted information goes out of the organization.

Corresponding Author: Mohamed Abdelkader Bencherif, Saad Dahlab University, Blida, Algeria

Table 1: Comparable strengths of different cryptographic issues[3]

Bits of Security	Symmetric key algorithms	IFC (e.g., RSA)	ECC (e.g., ECDSA)
80	2TDEA	k = 1024	f = 160-223
112	3TDEA	k = 2048	f = 224-255
128	AES-128	k = 3072	f = 256-383
192	AES-192	k = 7680	f = 384-511
256	AES-256	k = 15360	f = 512+

f and k: Represent the equivalent key sizes

Why elliptic curve cryptography? : In 1985, Koblitz and Miller introduced the use of elliptic curves in public key cryptography. called Elliptic Curve Cryptography (ECC), Basically, the main operation of elliptic curves consists of multiplying a point by a scalar in order to get a second point, the complexity arises from the fact that given the initial point and the final point, the scalar could not be deduced, leading to a very difficult problem of reversibility, or crypto analysis, called also the elliptic curve discrete logarithm problem.

The ECC algorithms with their small key sizes present nowadays the best challenge for cryptanalysis problems compared to RSA or AES, thus dealing with ECC will lead to smaller area hardware, less bandwidth use and more secure transactions, as shown in (Table 1).

The attractiveness of ECC algorithms is that they operate on a Galois Field (GF), by means of two simple operations, known as the field addition and field multiplication, which define a ring over GF(p^m) where p and m are primes. In the particular case, where we deal with hardware implementations, a binary field is preferred, where the couple (p, m), defines the set of elliptic curves. In our case, p = 2 and m = 163.

In this research, we present an FPGA hardware implementation of the elliptic curve cryptography scheme, using the Montgomery scalar multiplication based on the "add and double" algorithm, targeting as a primary goal an increase in the speed of the hardware implementation and an optimization in the ensuing inverse component.

MATERIALS AND METHODS

Hardware implementation : The strategy of hardware executing the ECC algorithms reposes on the ability of making the scalar multiplication in the GF(2^m) in a very few clock cycles. While increasing m, implementations become very time and resource consuming.

Most of the known architectures concern the acceleration of the multiplication process by modifying the elliptic equations by changing the Z coordinate term[4], or by multiplication scalability[5],or by using many serial and parallel Arithmetic units[6], or using High parallel Karatsuba Multipliers[7], those based on the Massy-Omura multipliers[8], or the work based on a hybrid multipliers approach[9], also some parallel approaches[10], or the New word level structure[11], or through the systolic architecture of[12], or by using the half and add method of[13], or by parallelizing both the add and double Montgomery algorithms[14].

The second problem concerns the inversion which has been tackled by[15], based on the Fermat little theorem of[16], or the almost inverse algorithm based on Kaliski's research[17].

In order to concentrate on one of the problems, some modifications have been done on the ECC equations[18] in order to postpone inversion to the last stage, while dealing only with the multiplication process.

In the next part we present the mathematical background of ECC, while in the material and methods section, we present the FPGA hardware proposed implementation, followed by the simulation results, at last we complete this study by a discussion and a final conclusion.

Elliptic curve mathematical background: ECC is based on the discrete logarithm problem applied to elliptic curves over a finite field. In particular, for an elliptic curve E that relies on the fact that it is computationally easy to find:

$$Q = k \times P \tag{1}$$

Where:

P and Q = Points of the elliptic curve E and their coordinates belong to the underlying GF (2^m)

k = A scalar that belongs to the set of numbers {1…#G-1}, G being the order of the curve E

Nowadays, there is no known algorithm able to compute k given P and Q in a sub exponential time[18].

The equation of a non-super singular elliptic curve with the underlying field GF(2^m) is presented in Eq. 2. It is formed by choosing the elements "a" and "b" within GF(2^m) with:

$$y^2 + x.y = x^3 + a.x^2 + b \tag{2}$$

In the affine-coordinate representation, a finite field point on E(GF(2^m)) is specified by two coordinates x and y both belonging to GF(2^m) satisfying Eq. 2 The point at infinity has no affine coordinates.

In most ECC hardware designs the choice of using three coordinates reposed on avoiding the periodic division of Eq. 3, which consumes a lot of resources in terms of execution cycles, as well as memory and power consumption:

$$x3 = \begin{cases} \left(\frac{y_1 + y_2}{x_1 + x_2}\right)^2 + \left(\frac{y_1 + y_2}{x_1 + x_2}\right) + x_1 + x_2 + a, & P \neq Q \\ x_1^2 + \left(\frac{b}{x_1^2}\right) & P = Q \end{cases} \quad (3)$$

$$y_3 = \begin{cases} \left(\frac{y_1 + y_2}{x_1 + x_2}\right)(x_1 + x_3) + x_3 + y_1 & P \neq Q \\ x_1^2 + \left(x_1 + \frac{y_1}{x_1}\right).x_3 & P = Q \end{cases}$$

A point is converted from a couple of coordinates to a triple system of coordinates using one of the transforms of (Table 2).

In our implementation, the Lopez-Dahab mapping is applied, because the set of operations is reduced compared to the other mappings[13] as presented in (Table 3).

Thus a point P(x, y) is mapped into P(X,Y,Z), that is a third projective coordinate is introduced in order to "flatten" the equations and avoid the division.

The startup transformation required for the implementation is simply done by initializing X, Y and Z as in Eq. 4[20]:

$$\{X = x, \quad Y = y, \quad Z = 1\} \quad (4)$$

Introducing the new tri-coordinates into Eq. 2 becomes:

$$Y^2 \times Z + X \times Y \times Z = X^3 + a \times X^2 + b \times Z^3 \quad (5)$$

The VHDL implementation will be based now on Eq. 5.

Table 2: Types of projective mappings

Representation	Mapping to affine coordinates
Projective	x = X/Z, y = Y/Z
Jacobian	x = X/Z^2, y = Y/Z^3
Lopez-Dahab	x = X/Z, y = Y/Z^2

Table 3: Multiplication and division costs of the add and double Montgomery algorithms[19]

Projective Representation	Point addition*	Point doubling
Affine	2M+1S+8A+1I	3M+2S+4A+1I
Standard	13M+1S+7A	7M+5S+4A
Jacobian	11M+4S+7A	5M+5S+4A
Lopez-Dahab	10M+4S+8A	5M+5S+4A

*: Field Operations: M: Multiplication, S: Squaring, A: Addition, I: Inverse

After completion of the successive operations of addition and multiplication, back to two affine coordinates as follows:

$$\left\{x = X/Z \quad , \quad y = Y/Z^2\right\} \quad (6)$$

In order to make the different computations, the Montgomery Point doubling and Montgomery Point addition algorithms are used, mainly through the ingenious observation of Montgomery, which states that the Y coordinate does not participate into the computations and can be delayed to the final stage[20]. Thus, back to working with only two projective coordinates.

Let us consider the points $P(X_1,Y_1,Z_1)$, $R(X_2,Y_2,Z_2)$, $Q(X_3,Y_3,Z_3)$, belonging to the curve $E(GF(2^{163}))$, where $R = 2 \times P$ and $Q = P+R$, the computations become, through the use of Montgomery method respectively as follows:

- The Montgomery point doubling algorithm: (MontgDouble):

$$\begin{cases} X_2 = X_1^4 + b \times Z_1^4 \\ Z_2 = X_1^2 \times Z_1^2 \end{cases} \quad (7)$$

  Requiring, 4 field squaring operations, 2 field multiplications and one simple field addition
- The Montgomery Point addition algorithm: (MontgAdd):

$$\begin{cases} Z_3 = (X \times_1 Z_2 + X_2 \times Z_1)^2 \\ X_3 = x \times Z_3 + (X_1 \times Z_2) \times (X_2 \times Z_1) \end{cases} \quad (8)$$

  Requiring, 1 field squaring operations, 4 field multiplications and two simple field additions

For the hardware implementation issue, k is represented on an m bits register, as:

$$k = [1, k_{m-1}, k_{m-2}, ..., k_1, k_0] \quad (9)$$

Both Eq. 7 and 8 are used in the Eq. 1 using the scalar Montgomery multiplication algorithm as shown in Fig. 1.

The inversion in $GF(2^{163})$, required at the final stage, could be realized in one of the two known methods, either via the Extended Euclidean algorithm, or by the Fermat's theorem which states that knowing after proof that: $A^{2^m-1} = 1$, leads to consider that $A^{-1} = A^{2^m-2}$ is also factual.

Inputs: k, b, P(x,y) $\in$ E(GF(2^m))

Initialization:

$X_1 = x; Z_1 = 1;\ X_2 = x^4; Z_2 = x^2$

for i = n-2 down to 0 do

  If $k_i = 1$ then

    $(X_1, Z_1) = \text{MontgAdd}(X_1, Z_1, X_2, Z_2)$

    $(X_2, Z_2) = \text{MontgDouble}(X_2, Z_2)$

  else

    $(X_2, Z_2) = \text{MontgAdd}(X_2, Z_2, X_1, Z_1)$

    $(X_1, Z_1) = \text{MontgDouble}(X_1, Z_1)$

  end if

end for

$x_3 = X_1/Z_1$

$$y_3 = \left(x + \frac{X_1}{Z_1}\right) \times d \times (x \times Z_1 \times Z_2)^{-1} + y$$

With:

$$d = \left[(X_1 + x \times Z_1)(X_2 + x \times Z_2) + (x_2 + y)(Z_1 \times Z_2)\right]$$

Output: $Q(x = x_3, y = y_3)$

Fig. 1: Montgomery scalar multiplication algorithm

Thus, in order to compute the inverse of one element in GF(2^{163}), one needs to take the power of this element (2^{163}-2) times.

By Using the Itoh-Tsjuii algorithm based on the add and multiply method leads to realize the inverse as presented in (Table 3)[21].

FPGA implementation: The 163 bits ECC component has been developed using the VHDL language. The different components forming the design are as follows:

- A 163 bits adder which is a simple 163 '2 bits' Xors
- A 163 bits modulo which is a xor-array evaluated through a Matlab script as an input-output matrix, through polynomial reduction using the National Institute of Standards and Technology (NIST) proposed polynomial $P(x) = x^{163}+x^7+x^6+x^3+1$[22]
- A 163 bits squarer that has also been generated from a Matlab script
- A 163 bits modified version of the Karatsuba-Offman multiplier circuit that is a based on splitting the operands into 3 identical operands [High (H), Middle (M) and Low(L) bits], the 'L-M-H' multiplier starts with a basic a 7 bits multiplier, leading to the following tree: $7 \rightarrow 19 \rightarrow 57 \rightarrow 163$ bits multipliers
- A Galois inverter circuit requiring 21 power squaring and 9 field multiplications within only 32 cycles

The ECC block diagram implementation is shown in Fig. 2.

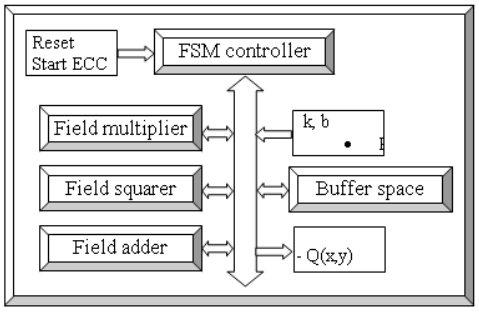

Fig. 2: ECC components implementation

Table 3: Sequence of multiplications and squaring for the inversion component in GF(2^{163}) of the element A**

$b_1 = A$	$= A^{2^1-1}$
$b_2 = b_1^{2^1} \times b_1$	$= A^{2^2-1}$
$b_3 = b_2^{2^2} \times b_2$	$= A^{2^3-1}$
$b_4 = b_3^{2^1} \times b_1$	$= A^{2^5-1}$
$b_5 = b_4^{2^5} \times b_4$	$= A^{2^{10}-1}$
$b_6 = b_5^{2^{10}} \times b_5$	$= A^{2^{20}-1}$
$b_7 = b_6^{2^{20}} \times b_6$	$= A^{2^{40}-1}$
$b_8 = b_7^{2^{40}} \times b_7$	$= A^{2^{80}-1}$
$b_9 = b_8^{2^1} \times b_1$	$= A^{2^{160}-1}$
$b_{10} = b_9^{2^{81}} \times b_9$	$= A^{2^{162}-1}$
$A^{-1} = b_{10}^{\ 2}$	$= A^{-1} = A^{2^{163}-2}$

**A and b_1 to b_{10} are elements of GF(2^{163})

RESULTS

In (Table 4), we present the respective estimated number of cycles, required for each part of the algorithm of Fig. 1, at each stage of the FSM controller.

The occurrences of the different basic operations required in all the FSM stages are listed in (Table 5).

Our main contribution concerned the execution of any basic field operation in just one cycle, taking into account, that lost cycles may occur, when the input of any component is back-propagated into the itself in the next iteration; and the use of non-clocked components, reducing the overall amount of clock driving and registered inputs/outputs.

The total number of cycles is equal to:

{Total # cycles} = {#cycles startup} + {#cycles affine to projective} + {#cycles initial point doubling} + (m-1) × [{#cycles counter Increase} + {#cycles Counter Compare} + {#cycles Point Addition} + {#cycles Point Double}] + {#cycles Affine to Projective}.

= 1+1+2+(162) × [1+1+10+9]+62

=3468 cycles

The total duration = {Total # cycles} × (Minimum Period)
= 3468*1.782ns = 6.1799 µs, equivalent to realize approximately: 161786 ECC scalar multiplications per sec

The speed of the implementation is based on the target device family, mainly those having enough slices and Input/Output pads. Our design is implemented on the xc5vsx95t-3f1136, with the following parameters:

- Goals of optimization set to speed
- Optimization effort set to high
- Global Optimization Goal set to AllClockNets

The partition design summary is presented in (Table 6), while (Table 7) shows the inverse circuit design summary, which shares with the full design the field multiplier, the field squarer and the field adder.

Our architecture has out-performed one of the best architectures[6], as shown in (Table 8), the performance (column 2) represents the required time for one complete scalar multiplication (over all the 163 bits of the scalar k), as per Eq. 1.

Table 4: Estimation of the FSM stages and their respective execution number of cycles

FSM Steps	#* stages	# execution cycles
Startup	1	1
Affine to projective	1	1
Initial point doubling	2	2
Counter increase	1	1
Counter compare	1	1
Montgomery point addition	9	162
Montgomery point doubling	10	162
Project to affine	62	1

*: The symbol #: Stands for: "Number of"

Table 5: Field operations required through the ECC implementation

Field operations	# cycles	# Occurrences in one cycle of the FSM
Field multiplication (163 bits)	1	24
Field squaring A^{2^1}	1	15
Field squaring A^{2^5}	1	7
Field squaring $A^{2^{15}}$	1	8
Field addition	1	11
Field reduction (modulo)	1	24

Table 6: Partition design summary of the ECC circuit

No. of slice registers	6376
No. of slice LUTs	21013
No. of fully used bit slices	4928

Table 7: Partition design summary of the inverse circuit

No. of slice registers	1962
No. of slice LUTs	15302
No. of fully used bit slices	1147

Benchmark tests: Working with 163 bits and 2^{163} order numbers or more, is not a direct way implementation, even checking of the results is very cumbersome, in this matter, different Matlab scripts with similar input/output behavior to the VHDL programming have been written, in order to compare the execution steps, as well the final results, timing is not taken into consideration in this specific stage (Emulation style process).

The benchmark tests have been done with the inputs of (Table 9) (in hexadecimal format)[22].

Fig. 3 and 4 show the intermediate results, obtained from the hardware simulator Modelsim, through different steps of the scalar multiplication as indicated by the "k_counter" value (5th line of the Fig. 3 and 4).

Table 8: Performance comparison of ECC (GF(2^{163})) implementations

Design	Performance [µs]	Frequency [MHz] (Max)
Chelton *et al*.[23]	19.5500	153.900
Smyth *et al*.[24]	3720.0000	166.000
Sozzani *et al*.[25]	30.0000	416.700
Satoh and Takano[26]	190.0000	510.200
Sakiyama *et al*.[6]	12.0000	555.600
Present study (Fastest)	6.1799	561.136

Table 9: Benchmark input test vectors of the NIST proposed curve B-163[18]

Px = '3F0EBA16286A2D57EA0991168D4994637E8343E36'
Py = '0D51FBC6C71A0094FA2CDD545B11C5C0C797324F1'
b = '20A601907B8C953CA1481EB10512F78744A3205FD'

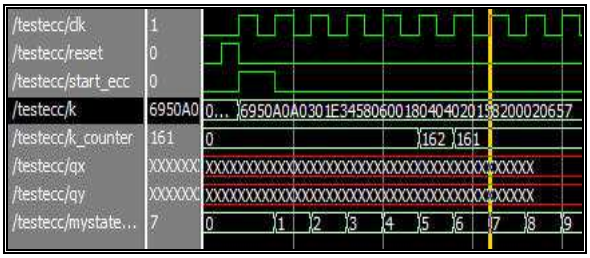

Fig. 3: Startup of the simulation under Modelsim

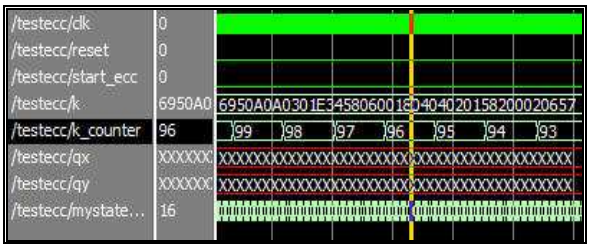

Fig. 4: Stepping through the bits of the scalar k

Fig. 5: Final result of the scalar multiplication k×P

Table 10: The x and y output coordinates of the result point Q=kxP, for an arbitrary value of k

k = ’6950A0A0301E34580600180404020158200020657’
Q.x = ’39FBE940C15F355289DEF938ABF2512FEA36EC7BE’
Q.y = ’356A9D7CDCD59E95546B385E77566EF4B4DD81D3’

Fig. 5 and Table 10 show the output results of the ECC scalar multiplication for a "163 bits" arbitrary value of k.

The implementation was intensively tested for different inputs of k and the obtained results were compared each time with the outputs of the different scripts written within Matlab. Both, hardware implementation and software emulation generated the same results over the 163 bits.

DISCUSSION

The main contribution of present research concerned three major points:

- An optimal Finite State Machine (FSM) controlling the whole components, minimizing empty cycles.
- Optimization of the hardware inversion process, by reducing the number of different squaring from 162-21, leading to an inversion in just 32 cycles
- Separation of the data path routing from the control part, in order to modify only the multiplier, the squarer, the adder as well as the modulo component for the different curves of (Table 1)

The introduction of additional multipliers and squarer’s can speed up the design at the expense of hardware spreading inside the FPGA. In embedded processes, this choice “speed/space” is crucial, depending mainly on the type of application, the targeted space, the possible addition of extra future functions and finally the cost allocated to the project.

The results, we obtained are very encouraging and will impact our decision on the embedding of larger encryption schemes, mainly the extension to the NIST proposed curves (193, 233, 283, 409 and 571) in a single FPGA, taking into account: The use of two or more multipliers (tuned parallel design), the use of internal memories such as Block RAMs (optimized timing memory accesses), the speed up of the FSM, as well as using different ECC hardware algorithms…; these optimization schemes are constrained to minimize the parallel inputs of the design and reduce routing circuitry, that dramatically decrease efficiency, lower speed and increase power consumption.

CONCLUSION

We have presented a fast version of an ECC crypto-hardware based on a finite state machine, implemented on a XILINX FPGA xc5vsx95t-1136 device. We attained a frequency of 561.136 MHz, which allows the execution of 161786 scalar multiplications per sec. Compared to the remarkable research of[6], with its 555.6 MHz, allowing 80000 scalar multiplications.

Our implementation can be still more competitive while introducing more optimization at the level of the multiplier and the squaring components.

The second main optimization, in present research, concerned the modular inverse circuit; which attained the frequency of 777.341 MHz; that is 13 times faster than the implementation of[27], against an increase, from our side, of 1:2 in the number of slices, for the 163 bits operands.

ACKNOWLEDGEMENT

This research is part of the research protocol of the Arithmetic, Architecture, Algorithms Serial and Parallel Team, (A3SP), Systems Architecture and Multimedia Department. (Centre de Développement des Technologies Avancées), CDTA, Algeria, for the period 2007-2009. (www.cdta.dz).

REFERENCES

1. Wollinger, T., J. Guajardo and C. Paar, 2004. Security on FPGAs: State of the art implementations and attacks. ACM. Trans. Embedd. Comput. Syst., 3: 534-574. http://portal.acm.org/citation.cfm?id=1015052
2. Husemöller, D., 2002. Elliptic Curves. 2nd Edn., Springer-Verlag, New York, ISBN: 0-387-95490-2, pp: 510.
3. Barker, E., W. Barker, W. Burr, W. Polk and M. Smid, 2007. NIST SP 800-57: Recommendation for key management-part 1: General. http://www.citeulike.org/user/dhein1030/article/3730728

4. Lee, Y.K., L. Batina, K. Sakiyama and I. Verbauwhede, 2008. Elliptic curve based security processor for RFID. IEEE. Trans. Comput., 57: 1514-1527. DOI: 10.1109/TC.2008.148.
5. Chelton, W.N. and M. Benaissa, 2004. A scalable GF(2^m) arithmetic unit for application in an ECC processor. Proceeding of the IEEE Workshop on Signal Processing Systems, Oct. 13-15, Crowne Plaza Hotel, Austin Texas, USA., pp: 355-360. DOI: 10.1109/SIPS.2004.1363076
6. Sakiyama, K., L. Batina, B. Preneel and I. Verbauwhede, 2007. High-performance public-key cryptoprocessor for wireless mobile applications. Mobile Network Appli., 12: 245-258. DOI 10.1007/s11036-007-0020-6.
7. Grabbe, C., M. Bednara, J. Tech, G.J. Von Zur and J. Shokrollahi, 2003. FPGA designs of parallel high performance GF(2^{233}) multipliers [cryptographic applications]. Proceedings of the International Symposium on Circuits and Systems, May 25-28, IEEE Xplore Press, Washington DC., USA., pp: 268-271. DOI: 10.1109/ISCAS.2003.1205958
8. Sutikno, S. and A. Surya, 2000. An architecture of F2^2n multiplier for elliptic. Proceedings of the 2000 IEEE International Symposium on Circuits and Systems, IEEE Xplore Press, Geneva, pp: 279-282. DOI: 10.1109/ISCAS.2000.857084
9. Quan, G., J.P. Davis, S. Devarkal and D.A. Buell, 2005. High-level synthesis for large bit-width multipliers on FPGAs: A case study. Proceedings of the 3rd IEEE/ACM/IFIP International Conference on Hardware/Software Co-Design and System Synthesis, Sept. 2005, IEEE Xplore Press, Jersey City, New Jersey, USA., pp: 213-218. DOI: 10.1145/1084834.1084890
10. Hinkelmann, H., P. Zipf, J. Li, G. Liu and M. Glesner, 2009. On the design of reconfigurable multipliers for integer and Galois field multiplication. Microproc. Microsyst., 33: 2-12. DOI: 10.1016/j.micpro.2008.08.003
11. Benaissa, M. and W.M. Lim, 2006. Design of flexible GF (2^m) elliptic curve cryptography processors. IEEE. Trans. Very Large Scale Integrat. Syst., 14: 659-662. DOI: 10.1109/TVLSI.2006.878235
12. Tsai, W.C. and S.J. Wang, 2002. A systolic architecture for elliptic curve cryptosystems. Proceedings of the 5th International Conference on Signal Processing, Aug. 21-25, IEEE Xplore Press, Beijing, China, pp: 591-597. DOI: 10.1109/ICOSP.2000.894560
13. Rodriguez, S.M.H. and F.R. Henriquez, 2005. An FPGA arithmetic logic unit for computing scalar multiplication using the half-and-add method. Proceedings of the International Conference on Reconfigurable Computing and FPGAs, Sept. 21-28, IEEE Xplore Press, Washington DC., USA., pp: 1-7. DOI: 10.1109/RECONFIG.2005.8
14. Cheung, R.C.C., N.J. Telle, W. Luk and P.Y.K. Cheung, 2005. Customizable elliptic curve cryptosystems. IEEE. Trans. Very Large Scale Integrat. Syst., 13: 1048-1059. DOI: 10.1109/TVLSI.2005.857179
15. Crowe, F., A. Daly and W. Marnane, 2005. Optimized Montgomery domain inversion on FPGA. Proceedings of the European Conference on Circuit Theory and Design, Aug. 28-Sept. 2, IEEE Xplore Press, Washington DC., USA., pp: I/277-I/280. DOI: 10.1109/ECCTD.2005.1522964
16. McIvor, C.J., M. McLoone and J.V. McCanny, 2006. Hardware Elliptic Curve Cryptographic Processor Over GF(p). IEEE. Trans. Circ. Syst., 53: 1946-1957. DOI: 10.1109/TCSI.2006.880184
17. Daly, A., W. Marnane, T. Kerins and E. Popovici, 2006. An FPGA implementation of a GF(p) ALU for encryption processors. Appli. Des., 28: 253-260. DOI: 10.1016/J.MICPRO.2004.03.006
18. Henkerson, D., A. Menezes and S. Vanstone, 2004. Guide to Elliptic Curve Cryptography. Springer-Verlag, ISBN: 038795273X, pp: 102,264.
19. Shokrollahi, J., 2006. Efficient Implementation of Elliptic Curve Cryptography on FPGAs. Doctorate thesis, Bohn University. http://deposit.ddb.de/cgi-bin/dokserv?idn=983089183&dok_var=d1&dok_ext=pdf&filename=983089183.pdf
20. Rodriguez-Henriquez, F., N.A. Saqib, A. Diaz-Perez and C. Kaya Koc, 2006. Cryptographic Algorithms on Reconfigurable Hardware. Springer, ISBN: 0387338837, pp: 299.
21. Guajardo, J. and C. Paar, 2002. Itoh-tsujii inversion in standard basis and its application in cryptography and codes. Des. Code. Cryptograph., 25: 207-216. DOI: 10.1023/A:1013860532636
22. National Institute of Standards and Technology, 2000. FIPS PUB 186-2: Digital Signature Standard (DSS). http://xml.coverpages.org/FIPS-186-2.pdf
23. Chelton, W.N. and M. Benaissa, 2008. Fast elliptic curve cryptography on FPGA. IEEE. Trans. Very Large Scale Integrat. Syst., 16: 198-205. DOI: 10.1109/TVLSI.2007.912228

24. Smyth, N., M. McLoone and J.V. McCanny, 2006. An adaptable and scalable asymmetric cryptographic processor. Proceedings of the IEEE International Conference on Application-Specific Systems, Architectures and Processors (ASAP), Sept. 2006, IEEE Computer Society, Washington DC., USA., pp: 341-346. DOI: 10.1109/ASAP.2006.8
25. Sozzani, F., G. Bertoni, S. Turcato, L. Breveglieri, 2005. A parallelized design for an elliptic curve cryptosystem coprocessor. Proceedings of the International Symposium on Information Technology: Coding and Computing (ITCC), Oct. 4-6, IEEE Xplore Press, USA., pp: 626-630. DOI: 10.1109/ITCC.2005.25
26. Satoh, A. and K. Takano, 2003. A scalable dual-field elliptic curve cryptographic processor. IEEE Trans. Comput., 52: 449-460. DOI: 10.1109/TC.2003.1190586
27. De Dormale, G.M., P. Bulens and J.J. Quisquater, 2004. An improved montgomery modular inversion targeted for efficient implementation on FPGA. Proceedings of the IEEE International Conference on Field-Programmable Technology, Oct. 6-8, IEEE Computer Society, Washington DC., USA., pp: 441-444. DOI: 10.1109/FPT.2004.1393320