DESIGN OF GAUSSIAN NORMAL AND POLYNOMIAL BASIS MULTIPLIERS OVER GF(2^{163})

*Vladimir Trujillo-Olaya, Jaime Velasco-Medina, Julio C. López-Hernández**

Grupo de Bionanoelectrónica, Escuela EIEE, Universidad del Valle, Cali, Colombia
* Instituto de computacao, UNICAMP, Campinas, Brasil
E-mail: vlatruo@univalle.edu.co, jvelasco@univalle.edu.co, jlopez@ic.unicamp.br

ABSTRACT

This article address efficient hardware implementations for multiplication over GF(2^{163}). Hardware implementations of multiplication algorithms are suitable for elliptic curve cryptoprocessor designs, which allow that elliptic curve based cryptosystems implemented in hardware provide more physical security and higher speed than software implementations. In this case, the multipliers were designed using conventional, modified and fast multiplication algorithms, the synthesis and simulation were carried out using Quartus II v.5.0 of Altera, and the designs were synthesized on the Stratix II EP2S60F1020C3. The simulation results show that the multipliers designed present a very good performance using small area.

DESIGN OF GAUSSIAN NORMAL AND POLYNOMIAL BASIS MULTIPLIERS OVER GF(2^{163})

*Vladimir Trujillo-Olaya, Jaime Velasco-Medina, Julio C. López-Hernández**

Grupo de Bionanoelectrónica, Escuela EIEE, Universidad del Valle, Cali, Colombia
* Instituto de computacao, UNICAMP, Campinas, Brasil
E-mail: vlatruo@univalle.edu.co, jvelasco@univalle.edu.co, jlopez@ic.unicamp.br

ABSTRACT

This article address efficient hardware implementations for multiplication over GF(2^{163}). Hardware implementations of multiplication algorithms are suitable for elliptic curve cryptoprocessor designs, which allow that elliptic curve based cryptosystems implemented in hardware provide more physical security and higher speed than software implementations. In this case, the multipliers were designed using conventional, modified and fast multiplication algorithms, the synthesis and simulation were carried out using Quartus II v.5.0 of Altera, and the designs were synthesized on the Stratix II EP2S60F1020C3. The simulation results show that the multipliers designed present a very good performance using small area.

1. INTRODUCTION

In order to protect or exchange confidential data, the cryptography plays an important role in the security of the information. Therefore, it is necessary to implement efficient cryptosystems, which can support applications economically feasible. In this context, public key cryptography based on elliptic curves is widely used because it presents higher security per key bit, and their two main applications are the private key exchange and the digital signature. Additionally, the Elliptic Curve Cryptosystems (ECC) can be used in applications where the computation resources are limited such as smart cards and cellular telephones. The ECC systems are included in the NIST and ANSI standards, and the principal advantage over other systems of public key like RSA is the size of the parameters, which are very small, however the ECC systems provide the same level of computational security.

On the other hand, the algorithms used for elliptic curve cryptosystems are divided hierarchically into three levels. The arithmetic level, the group operation level and the encryption level. Therefore, in order to achieve efficient hardware implementations it is evident to reach the best algorithm optimization for each level. However, it is import to mention that the most expensive operation applied in elliptic curve based cryptosystems is the "scalar multiplication" of a large natural number with a point on an elliptic curve [1]. In this case, the performance of an elliptic curve cryptoprocessor depends on the multiplication over GF(2^m). Therefore, the multiplier is the most important functional block for elliptic curve cryptoprocessor design.

In the literature are presented few hardware designs for the gaussian normal basis multiplication over GF(2^m). In [2], V. Trujillo, J. Velasco and J. C. López present the hardware design for the Gaussian normal multiplier over GF(2^{163}). In [3], Elia and Leone present a theoretic study about a general algorithm to design fast parallel multipliers in any basis over GF(2^m). The algorithm has been applied to gaussian normal basis multiplication over small galois field (GF(2^{73})). However, for m > 73 is not considered. Additionally, the total number of gates (XOR and AND) is not guaranteed to be the minimum. In this case, the algorithm is aimed at minimizing the number of XOR gates. In [4], Hua Li and Chang N. Zhang present an algorithm to design a low complexity programmable cellular automata based versatile modular multiplier over GF(2^m).

This work address efficient hardware implementations for gaussian normal basis and polynomial basis multiplication over GF(2^{163}). In this case, the multipliers designed present a good speed/area ratio, which is very suitable for elliptic curve cryptoprocessor design. Therefore, elliptic curve based cryptosystems can be used in applications that require small area, good speed and low consumption power, such as smart cards and cellular telephones.

This article is organized as follows. Initially, sections 2, 3 and 4 present the basic concepts on GF(2^m) arithmetic and algorithms for gaussian normal basis and polynomial basis multiplication over GF(2^m). In section 5, the hardware architectures for the multipliers are described. In section 6, the simulation results are presented. Finally, section 7 presents the conclusions and the future work.

2. GF(2^m) ARITHMETIC

2.1 Concepts on gaussian normal basis

Normal Basis representations of an element in the finite field GF(2^{163}) have the computational advantage that squaring an element can be done very efficiently. However multiplying distinct elements, can be cumbersome in general for this reason, ANSI X9.62 specifies that *Gaussian Normal Basis* be used, for which multiplication is both simpler and more efficient [5].

A normal basis for GF(2^m) is as follows:

$\{\beta, \beta^2, \beta^{2^2}, ..., \beta^{2^{m-1}}\}$, where $\beta \in$ GF(2^m)

where, any element $\alpha \in$ GF(2^m) can be written as follows:

$$a = \sum_{i=0}^{m-1} a_i \beta^{2^i} \text{ where } a_i \in \{0,1\}.$$

The *type T* of a Gaussian Normal Basis (GNB) is a positive integer, measuring the complexity of the multiplication operation with respect to that basis. Generally, the type *T* of smaller value allows to make a more efficient multiplication. For a given *m* and *T*, the field GF(2^m) can have at most one GNB of type *T*.

A GNB exists whenever *m* is not divisible by 8. Let *m* be a positive integer and let *T* be a positive integer. Then the type *T* of a GNB for GF(2^m) exists if and only if $p = Tm + 1$ is prime.

If $\{\beta, \beta^2, \beta^{2^2}, ..., \beta^{2^{m-1}}\}$ is a Gaussian Normal Basis in the finite field GF(2^m), then the element $a = \sum_{i=0}^{m-1} a_i \beta^{2^i}$ is represented by the binary string $(a_0 a_1 a_2 ..... a_{m-1})$ where $a_i \in \{0,1\}$

In this case, the multiplicative identity element is represented by the bit string of all 1's. While the additive identity element is represented by the bit string of all 0's.

An important result for the arithmetic of the Gaussian Normal Basis is the Fermat's Theorem. For all $\beta \in$ GF(2^m) so that:

$$\beta^{2^m} = \beta$$

This theorem is important to carry out the squaring of an element in the finite field GF(2^m).

2.2 GF(2^m) arithmetic for gaussian normal basis

The following arithmetic operations are defined on the elements of GF(2^m) when using a GNB of type *T*:

- *Addition*:

If $a = (a_0 a_1 a_2 ..... a_{m-1})$ and $b = (b_0 b_1 b_2 ... b_{m-1})$ are elements of GF(2^m), then $a + b = c = (c_0 c_1 c_2 ... c_{m-1})$ where $c_i = (a_i + b_i) \bmod 2$.

- *Squaring*:

Let $a = (a_0 a_1 a_2 ..... a_{m-1}) \in$ GF(2^m), then

$$a^2 = \left(\sum_{i=0}^{m-1} a_i \beta^{2^i} \right)^2$$

$$= \sum_{i=0}^{m-1} a_i \beta^{2^{i+1}} = \sum_{i=0}^{m-1} a_{i-1} \beta^{2^i}$$ due to Fermat's Theorem; $\beta^{2^m} = \beta$, then

$$a^2 = (a_{m-1} a_0 a_1 a_2 ..... a_{m-2})$$

In this case, squaring is a simple rotation of the vector representation.

- *Multiplication*:

Let $p = Tm + 1$ and let $u \in$ GF(p) be an element of order *T*. Define the sequence *J(1), J(2), ..., J(p-1)* by: $J(2^i u^j \bmod p) = i$, for *i* from *0* until *m-1 and j* from *0* until *T-1*.

If $a = (a_0 a_1 a_2 ..... a_{m-1})$ and $b = (b_0 b_1 b_2 ... b_{m-1})$ are elements of GF(2^m), then $a \bullet b = c = (c_0 c_1 c_2 ... c_{m-1})$, where

$$c_i = \sum_{k=1}^{p-2} a_{J(k+1)} \bullet b_{J(p-k)}$$

- *Inversion*:

If $a \neq 0$ and $a \in$ GF(2^m), the inverse of *a*, is the unique element $c \in$ GF(2^m) for which $a \bullet c = 1$ ($c = a^{-1}$).

The algorithm used for inversion is based on the identity:

$$a^{-1} = a^{2^m - 2} = \left(a^{2^{m-1} - 1} \right)^2$$

In [6], Itoh and Tsujii proposed a method that minimizes the number of multiplications to calculate the inversion, which is based on the following identities:

$$a^{2^{m-1}-1} = \begin{cases} (a^{2^{\frac{m-1}{2}}-1})^{2^{\frac{m-1}{2}}} \bullet a^{2^{\frac{m-1}{2}}-1} & m \text{ odd} \\ a \bullet \left(a^{2^{m-2}-1} \right)^2 & m \text{ even} \end{cases}$$

2.3 Concepts on polynomial basis

This basis is also known as the canonical or standard basis, is defined as the set $\{1, \alpha, \alpha^2, \ldots, \alpha^{m-1}\}$ where α is a root of the irreducible polynomial *f(x)*. Where $f(x)=x^m+f_{m-1}x^{m-1}+\ldots f_2x^2+f_1x+f_0$ ($f_i \in \{0,1\}$) is an irreducible polynomial of degree *m* over GF(2^m) [4].

2.4 GF(2^m) arithmetic for polynomial basis

The following arithmetic operations are defined on the elements of GF(2^m), using a polynomial basis representation with reduction polynomial *f(x)*.

- *Addition*:

If $a = (a_0 a_1 a_2 \ldots . a_{m-1})$ and $b = (b_0 b_1 b_2 \ldots b_{m-1})$ are elements of GF(2^m), then $a + b = c = (c_0 c_1 c_2 \ldots c_{m-1})$ where $c_i = (a_i + b_i) \bmod 2$

- *Multiplication*:

If $a = (a_0 a_1 a_2 \ldots . a_{m-1})$ and $b = (b_0 b_1 b_2 \ldots b_{m-1})$ are elements of GF(2^m), then $a*b=r=(r_{m-1} r_{m-2} \ldots r_1 r_0)$ where the polynomial *r(x)* is the remainder when the polynomial *a(x)*b(x)* is divided by *f(x)* [4].

- *Inversion*:

If $a \neq 0$ and $a \in$ GF(2^m), the inverse of *a*, is the unique element $c \in$ GF(2^m) for which $a \bullet c = 1$ ($c = a^{-1}$).

3. ALGORITHMS FOR GAUSSIAN NORMAL BASIS MULTIPLICATION OVER GF(2^m)

In [5], NIST presents a conventional algorithm for the gaussian normal basis multiplication over GF(2^m), which is shown in Figure 1. In this case, an algorithm is used to generate the J(k) subindexes for the F(U,V) array, which is shown in Figure 2.

In [7], Lopez presents a modified conventional algorithm for the gaussian normal basis multiplication over GF(2^m), which is implemented in software and shown in Figure 3, and the values generated for the function p1 are shown in Figure 4.

The parameters recommended by NIST for GF(2^{163}) are:

m	= 163,	number of bits
T	= 4,	number recommended by NIST for GF(2^{163})
p	= 653,	prime number p = Tm + 1
U	= 149,	number that satisfies the relation $U^4 \bmod p = 1$

CONVENTIONAL ALGORITHM:
GNB MULTIPLICATION OVER GF(2^m)

Input: a, b $\in$ GF(2^m) ***Output:*** c = a.b $\in$ GF(2^m)

1. U $\leftarrow$ a = (a_0, a_1, ...a_{m-1})
2. V $\leftarrow$ b = (b_0, b_1, ...b_{m-1})
3. ***For*** k = 0 ***to*** m-1 ***do***
4. $c_{(k)}$ = F(U,V)
5. U = left rotation of U
6. V = left rotation of V
7. ***End***
8. c $\leftarrow$ = (c_0, c_1,... c_{m-1}) = a.b

where:

$$F(U,V) = \sum_{k=1}^{p-2} U_{J(k+1)} V_{J(k)}$$

Figure 1: *Conventional algorithm for the gaussian normal basis multiplication over GF(2^m)*

ALGORITHM: J(k) GENERATION FOR F(U, V)

Input: m, T, U, p ***Output:*** J(1), J(2), …, .J(p-1)

1. w $\leftarrow$ 1
2. ***For*** j=0 ***to*** T-1 ***do***
3. n $\leftarrow$ w
4. ***For*** i=0 ***to*** m-1 ***do***
5. j(n) $\leftarrow$ i
6. n $\leftarrow$ 2n mod p
7. ***End***
8. w $\leftarrow$ UW mod p
9. ***End***

Figure 2: *J(k) generation for F(U, V)*

MODIFIED ALGORITHM:
GNB MULTIPLICATION OVER GF(2^m)

Let A and B finite field elements in GF(2^{163})
Input: A, B in GF(2^m)
Output: C = A.B
1. T = B^2 (Rot_right(B,1))
2. C = 0
3. For i = m-1 to 0 do
   3.1 C = C^2 (C = Rot_right(C,1))
   3.2 if a_i = 1 then C = C xor mbeta(T)
   3.3 T = T^2 (T = Rot_right (T,1))
   End
4. Return C = A. B

$T(p1) = [0, T_{p1}(1), T_{p1}(2), T_{p1}(3), \ldots, T_{p1}(162)]$
$T(p2) = [0, T_{p2}(1), T_{p2}(2), T_{p2}(3), \ldots, T_{p2}(162)]$
$T(p3) = [T_1, T_{p3}(1), T_{p3}(2), T_{p3}(3), \ldots, T_{p3}(162)]$
$T(p4) = [0, T_{p4}(1), T_{p4}(2), T_{p4}(3), \ldots, T_{p4}(162)]$
Then
mbeta(T) = T(p1) xor T(p2) xor T(p3) xor T(p4)

Figure 3: *Modified conventional algorithm for the gaussian normal basis multiplication over GF(2^m)*

The functions p1, p2, p3, p4 are: pi[j], j= 1,2,3,...162

p1[] ={0,117,89,99,121,160,90,139,*93,35,154,131,82,111,121,64,*
55,136,72,9,7,45,43,32,*153,84,33,76,135,56,114,31*,
85,134,29,101,61,41,141,4,*105,57,147,113,148,97,28,72,*
80,81,13,130,153,11,144,75,*106,137,16,112,27,98,58,68,*
104,108,44,22,150,79,3,40,*71,19,145,62,50,54,17,73,*
94,132,159,154,25,30,115,65,*118,152,25,133,34,74,92,140,*
42,23,90,110,14,129,78,69,*5,49,77,103,123,91,2,125,*
148,142,161,158,1,145,151,143,*100,84,67,37,3,87,120,15,*
59,138,155,95,26,6,21,86,*107,60,102,20,46,149,152,53,*
2,71,48,106,40,38,146,119,*127,157,116,51,128,100,36,122,*
109,12}

Figure 4: *Pi(j) generation for p1 function*

In [3], Elia and Leone present an algorithm to design fast parallel multipliers , This work present the modified algorithm which consists of two steps which are shown in Figures 5 and 6.

FAST MULTIPLIER ALGORITHM: STEP 1

COMPUTATION OF TENSOR τ

Input: a normal polynomial $p(t) = c_0 + c_1 t + \ldots + c_{m-1} t^{m-1} + c_m t^m$ over $GF(2^m)$.

Output: associated tensor τ.

1. Compute:

$t = a_{0,0} + a_{0,1} t + \ldots + a_{0,m-1} t^{m-1}$

$t^2 = a_{1,0} + a_{1,1} t + \ldots + a_{1,m-1} t^{m-1}$

$t^4 = a_{2,0} + a_{2,1} t + \ldots + a_{2,m-1} t^{m-1}$

…

$t^{2^{m-1}} = a_{m-1,0} + a_{m-1,1} t + \ldots + a_{m-1,m-1} t^{m-1}$

2. Set:

$$A = \begin{pmatrix} a_{0,0} & a_{0,1} & \ldots & a_{0,m-1} \\ a_{1,0} & a_{1,1} & \ldots & a_{1,m-1} \\ \ldots & \ldots & \ldots & \ldots \\ a_{m-1,0} & a_{m-1,1} & \ldots & a_{m-1,m-1} \end{pmatrix}$$

3. Compute:

$$\mathbf{B} = \mathbf{A}^{-1}$$

4. Set:

$$C = \begin{pmatrix} 0 & 1 & 0 & \ldots & 0 \\ 0 & 0 & 1 & \ldots & 0 \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ 0 & 0 & 0 & \ldots & 1 \\ c_0 & c_1 & c_2 & \ldots & c_{m-1} \end{pmatrix}$$

5. Compute:

$$\mathbf{D} = \mathbf{ACB}$$

6. Set:

$$\mathbf{E} = \mathbf{D}^{T}$$

7. Set:

$$\tau = \left[E, E^{(1,1)}, E^{(2,2)}, \ldots, E^{(m-1,m-1)} \right]$$

8. Output τ.

Figure 5: *Computation of tensor* τ

FAST MULTIPLIER ALGORITHM: STEP 2 REDUCTION

Input: tensor τ, vector **Z**.

Output: reduced tensor τ and modified vector **Z**.

1. For i = 1, 2, …, m-1
2. For j = 0, 1, …, i-1
3. if $\mathbf{g}_{f(i,j)} = \mathbf{g}_{f(i,j)}$
4. Set $\mathbf{g}_{f(j,j)} = \mathbf{g}_{f(j,j)} + \mathbf{g}_{f(i,j)}$
5. Set $\mathbf{g}_{f(i,i)} = \mathbf{g}_{f(i,i)} + \mathbf{g}_{f(i,j)}$
6. Set $Z_{f(i,j)} = (a_i + a_j)(b_i + b_j)$
7. Set $\mathbf{g}_{f(j,j)} = 0$
8. Set $Z_{f(i,j)} = 0$
9. For i = 0, 1, …, m^2-2
10. if $\mathbf{g}_i = 0$ then continue
11. For j = i+1, …,m^2 –1 do
12. if $\mathbf{g}_i = \mathbf{g}_j$ then
13. Set $Z_i = Z_i + Z_j$
14. Set $\mathbf{g}_j = 0$
15. Set $Z_j = 0$
16. For i = 0, 1, …, m^2-1
17. if $\mathbf{g}_i = 0$ then continue
18. For j = 0, 1, …, m^2-1 and i ≠ j do
19. if $H_j < H_i$ then continue
20. else if $\mathbf{g}_i$ is include in $\mathbf{g}_j$ then
21. Set $\mathbf{g}_j = \mathbf{g}_j + \mathbf{g}_i$
22. Set $Z_i = Z_i + Z_j$
23. For j = 0, 1, …, m^2-1
24. if $\mathbf{g}_i = 0$ or $H_i < 3$ then continue
25. For j = i+1, …,m^2 –1 do
26. select superimposed columns
27. For j = i, i_1, i_2, … i_k
28. Set to zero superimsed entries of $\mathbf{g}_j$
29. Add a new column to τ
30. Add a new entry to Z
31. Output τ and Z

Figure 6: *Reduction of tensor* τ *and* Z

4. ALGORITHMS FOR POLYNOMIAL BASIS MULTIPLICATION OVER GF(2^m)

In [7], Hua Li and Chang N. Zhang present an algorithm to design a low complexity programmable cellular automata based versatile modular multiplier over $GF(2^{163})$.

This algorithm is shown in Figure 7. where $C(x)=A(x)B(x) \bmod P(x)$ can be written as follow:

$$C(x)=[(\ldots((0+a_{m-1}B(x))x+a_{m-2}B(x))x+\ldots+a_1B(x))x + a_0B(x)] \bmod P(x)$$

ALGORITHM:
POLYNOMIAL BASIS MULTIPLICATION OVER GF(2^m)

1. $C^{-1}(X) = 0$
2. for I=0 to m-1 do

$$C^i(x)=[\ C^{i-1}(x)x + a_{m-1-i}B/(x)]\ mod\ P(x)$$

where:

$$C^i(x) = \sum_{j=0}^{m-1} c_j^{\ i} * x^j$$

Figure 7: *Algorithm for Polynomial basis multiplication*

5. HARDWARE ARCHITECTURES FOR GF(2^{163}) MULTIPLIER

In this section are presented the hardware architectures for gaussian normal basis and polynomial basis multiplication over GF(2^{163}). In this case, conventional and modified algorithms for the gaussian normal basis multiplication and polynomial multiplication over GF(2^{163}) and fast parallel multiplier algorithm over GF(2^{97}) are implemented.

5.1. Conventional algorithm based multipliers

In order to achieve a good performance for hardware multipliers based on the conventional algorithm [2], several F(U,V) arrays can be used, which allow to speedup the multiplication. The hardware architecture for the gaussian normal basis multiplier based on using 4 F(U,V) arrays is shown in Figure 7.

In this architecture, the F(U,V) functional block allows to generate the subindexes that determine the order of combination of each bit of the two input data of the multiplier and it is implemented using XOR and AND functions. In this case, the F(U,V) array is designed using a combinational circuit, which is implemented using the following equation:

$$C_0 = (U_0 V_1) \oplus (U_1 V_0) \oplus (U_1 V_{117}) \oplus (U_1 V_{13}) \oplus (U_{102} V_{110}) \oplus \ldots\ldots\ldots$$
$$(U_{97} V_{149}) \oplus (U_{97} V_{42}) \oplus \ldots\ldots\ldots\ldots\ldots\ldots\ldots$$
$$(U_{99} V_{115}) \oplus (U_{99} V_4) \oplus (U_{99} V_{58}) \oplus (U_{99} V_{90})$$

The barrel shifter functional blocks allow to rotate and shift the input data U and V for the multiplier, and the serial-in/parallel-out register allows to storage the bit C_i, which is generated from the F(U,V) array, thus, this register stores the product C = A.B. The control unit is implemented using a FSM, which allows to control the I/O registers, generate the control sequences and initialize the multiplication.

5.2. Modified conventional algorithm based multipliers

The hardware architecture for the gaussian normal basis multiplier based on modified conventional algorithm (presented in [5]) is shown in Figure 8.

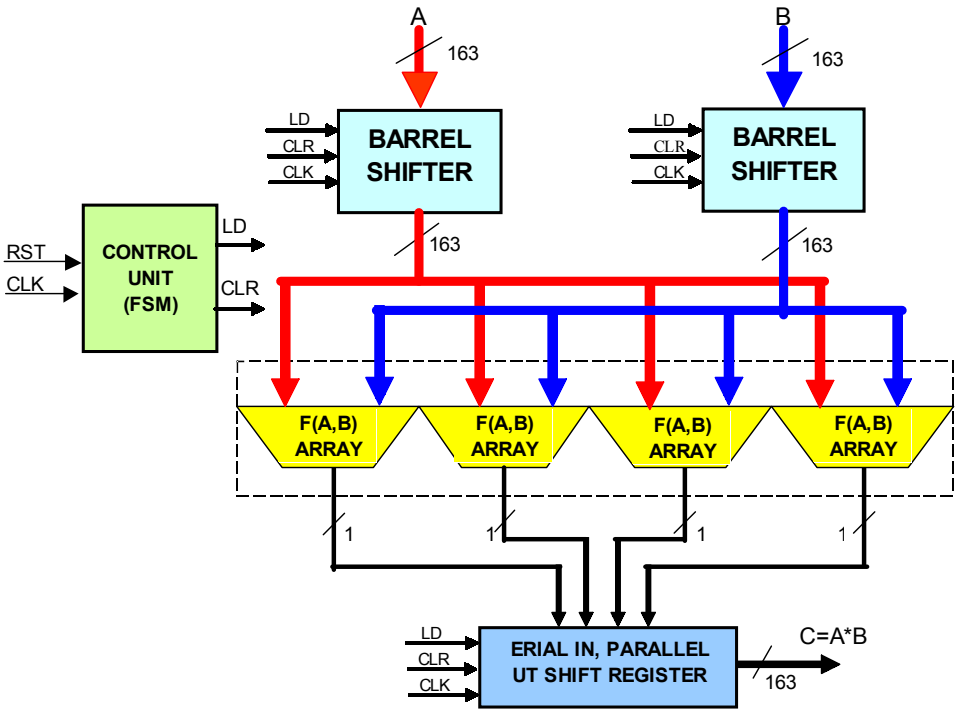

Figure 7*: Hardware architecture for GF(2^{163}) multiplier based on conventional algorithm using 4 F(U, V)*

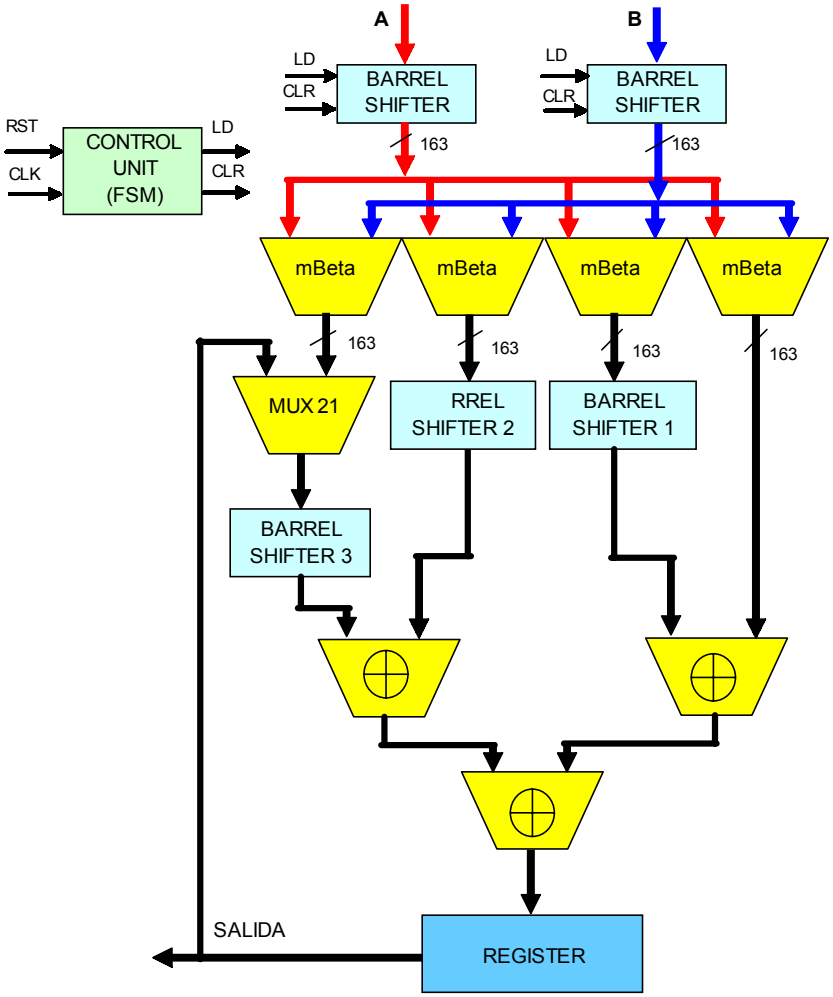

***Figure 8**: Hardware architecture for GF(2^{163}) multiplier based on modified conventional algorithm*

5.3. Fast parallel multiplier

The hardware architecture for the modified fast parallel multiplier algorithm for GF(2^{97}) is presented in Figure 9. In this case, the hardware architecture uses a new array which uses XOR and AND functions. By the moment, simulations results are obtained for the finite field multiplication over GF(2^{101}), however for the case GF(2^{163}), the Quartus II presents problems of fitting.

5.4. Polynomial basis multiplier

The architecture of a PCA cell is shown in Figure 10. Where Cm is configured as the coefficients of B(x) and Cr is configured as the coefficients of P(x), Xs is configured as coefficients of A(x), Xl and Xs are partial results of neighborhood PCA.

This work present an architecture modular multiplier based on PCA (Programmable Cellular Automata) and the polynomial basis representation, the basic architecture of the multiplier is suitable for both parallel and serial multiplier.

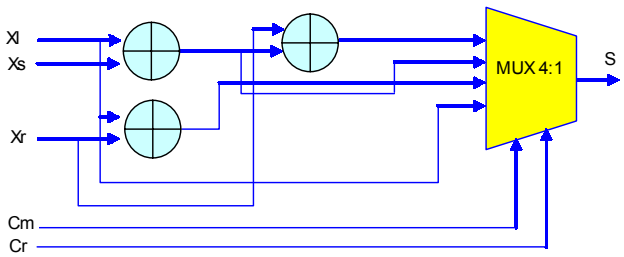

***Figure 10:** PCA cell*

An array of PCA cells determine the architecture of serial multiplier GF(2^4) shown in Figure 11 or parallel multiplier shown in Figure 12.

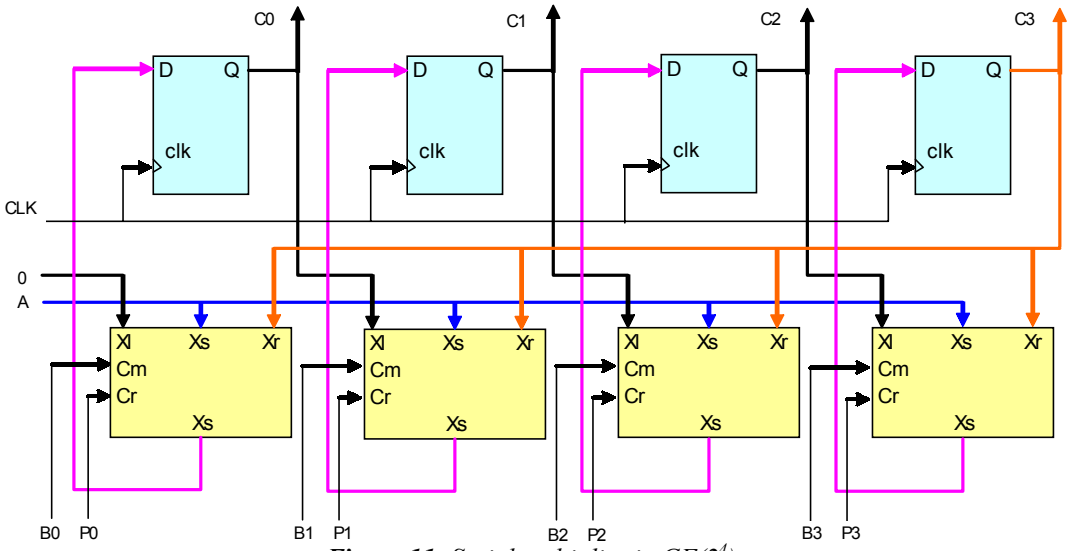

Figure 11: *Serial multiplier in GF(2^4)*

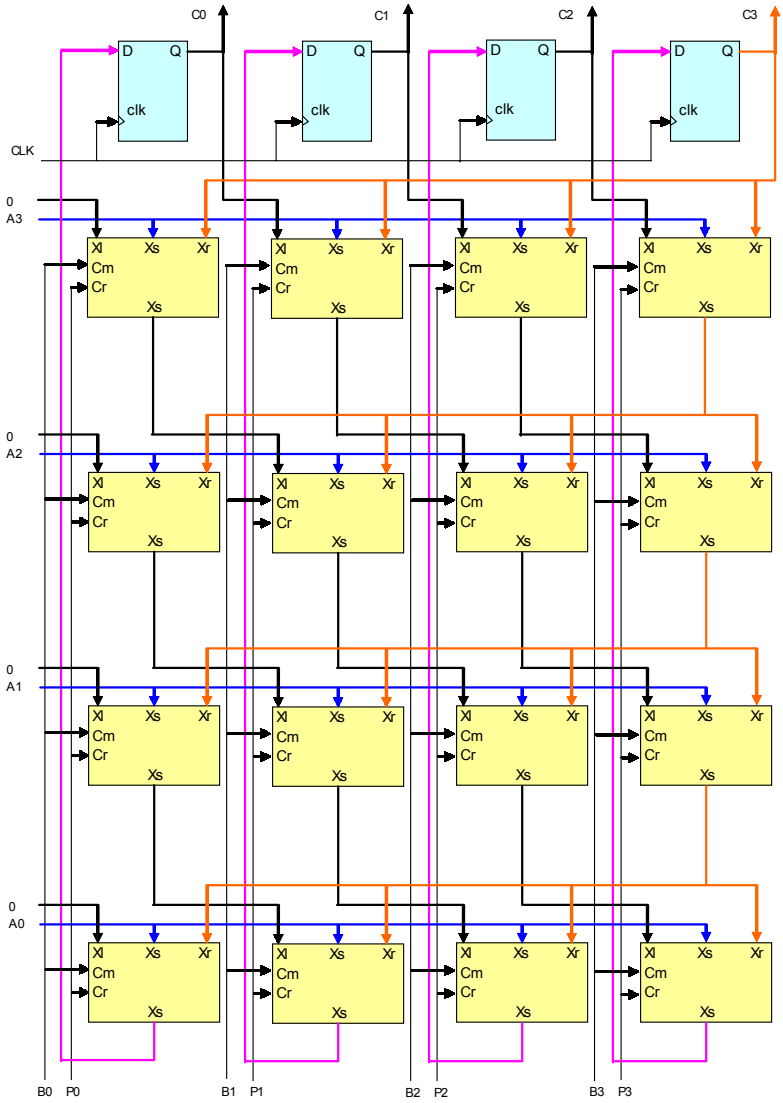

Figure 12: *Parallel multiplier in GF(2^4)*

6. SIMULATION RESULTS

In order to verify the performance of the multipliers, several simulations were carried out. The simulation results for hardware implementations are shown in Tables 1, 2, 3 and 4. The multipliers are implemented on the FPGA EP2S60F1020C3, and the simulation and synthesis were carried out using Quartus II version 4.0.

F(U,V)	Logic elements	F_{MAX}(MHz)	Tend (μs) using CLK = 10ns
1	974	128.34	1.704
4	2760	123.24	0.500
8	5144	113.00	0.300
16	9912	108.44	0.200

Table 1: *Simulation results for the conventional algorithm based multiplier*

mbeta(T)	Logic elements	F_{MAX}(MHz)	T (μs) CLK = 10ns
1	1249	178.13	1.67
4	1295	232.4	0.532

Table 2: *Simulation results for the modified conventional algorithm based multiplier*

As could be observed from Tables 1 and 2, the multipliers designed present a good performance using small area, which is very suitable for elliptic curve cryptoprocessor design. In Table 3 are presented the simulation results for the conventional algorithm based multiplier using pipeline architecture. In table 4 are presented the simulation results for polynomial multipliers.

F(U,V) Pipeline 2	Logic elements	F_{MAX}(MHz)	Tend (μs) using CLK = 10ns
1	1292	276.55	1.72
4	3209	292.74	0.495
8	6671	189.57	0.296
16	12973	163.21	0.196

F(U,V) Pipeline 4	Logic elements	F_{MAX}(MHz)	Tend (μs) using CLK = 10ns
1	1269	386.12	1.74
4	3437	284.98	0.517
8	6465	220.51	0.316
16	12575	184.09	0.217

Table 3: *Simulation results for the conventional algorithm based multiplier using pipeline architecture*

Multiplier	Logic elements	F_{MAX}(MHz)	T (μs) CLK = 10ns
Serial	163	215.8	1.67
Parallel	26569	4.6	0.02

Table 4: *Simulation results for the serial and parallel multiplier*

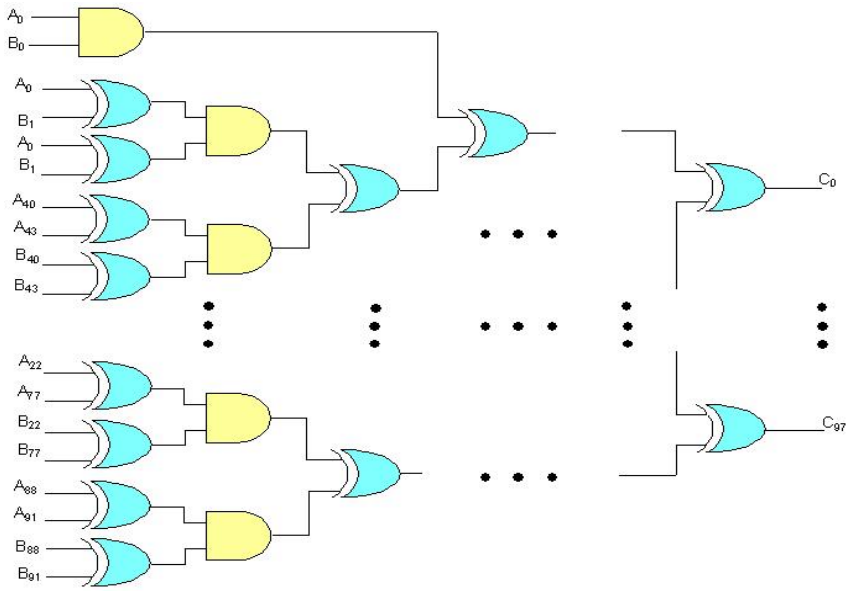

Figure 9*: Hardware architecture for GF(2^{97}) multiplier based on parallel multiplier algorithm*

7. CONCLUSIONS AND FUTURE WORK

This article presents the design of efficient hardware implementations for the gaussian normal and polynomial basis multiplication over GF(2^{163}). In this case, the multipliers are designed using conventional, modified and parallel multiplier algorithms for the multiplication over GF(2^m).

The multipliers present a good speed/area ratio, which is very suitable for elliptic curve cryptoprocessor design, this allows that elliptic curve based cryptosystems can support applications economically feasible such as smart cards and cellular telephones.

The multipliers were simulated using Quartus II of Altera and synthesized on the FPGA EP2S60F1020C3.

The future work, will be oriented to design hardware multipliers for the gaussian normal basis multiplication over GF(2^{233}), design a fast parallel multiplier over GF(2^{163}) and to implement new multiplication algorithms.

8. ACKNOWLEDGMENT

This work was sponsored by Altera Corporation through the University Program. The authors give a special thanks to Mrs Ralene Marcoccia of Altera Corporation.

8. BIBLIOGRAPHY

[1] M. Jung, "FPGA Based Implementation Of An Elliptic Curve Coprocessor Utilizing Synthesizable VHDL code", Darmstadt University of Technology. Available at http:// www.vlsi.informatik.tu-darmstadt.de/staff/mjung/publications/ comprehensive.pdf

[2] V. Trujillo, J. Velasco and J. López, "Multiplicador en el cuerpo finito GF(2^{163}) usando bases normales gaussianas", IX Workshop Ibercip Cuba 2003. www.iberchip.org/IX/Pages/Sesion11c.html

[3] M. Elia and M. Leone, "A basis-independent algorithm to design fast parallel multipliers over GF(2^m)", http://doi.ieeecomputersociety.org/10.1109/ITCC.2004.1286712, 2004.

[4] H. Li and C. N. Zhang, "Efficient Cellular Automata Based Versatile Multiplier for GF(2^m)", Journal of information Science and Engineering 18.479-488. 2002.

[5] National Institute of Standards and Technology, "Digital Signature Standard", FIPS Publication 186-2, February 2000. Available at http://csrc.nist.gov/fips.

[6] T. Itoh and S. Tsujii, "A fast algorithm for computing multiplicative inverses in GF(2^m) using normal bases", Information and Computation, 1988.

[7] J. Lopez Hernández, "Modified conventional algorithm for gaussian normal basis multiplication over GF(2^m)", internal report.