Karatsuba-Ofman Multiplier with Integrated Modular Reduction for $\mathbb{GF}(2^m)$

Eduardo CUEVAS-FARFAN[1], Miguel MORALES-SANDOVAL[2], Alicia MORALES-REYES[1], Claudia FEREGRINO-URIBE[1], Ignacio ALGREDO-BADILLO[3], Paris KITSOS[4], René CUMPLIDO[1]
[1]*Instituto Nacional de Astrofisica, Optica y Electronica, 72840, Puebla, Mexico*
[2]*Laboratorio de Tecnologias de la Informacion-CINVESTAV,87130, Victoria, Tamaulipas, Mexico*
[3]*Universidad del Istmo Tehuantepec,70760, Oaxaca, Mexico*
[4]*Hellenic Open University, GR-26222, Patras Greece*
cuevas.farfan@ccc.inaoep.mx

***Abstract*—In this paper a novel $\mathbb{GF}(2^m)$ multiplier based on Karatsuba-Ofman Algorithm is presented. A binary field multiplication in polynomial basis is typically viewed as a two steps process, a polynomial multiplication followed by a modular reduction step. This research proposes a modification to the original Karatsuba-Ofman Algorithm in order to integrate the modular reduction inside the polynomial multiplication step. Modular reduction is achieved by using parallel linear feedback registers. The new algorithm is described in detail and results from a hardware implementation on FPGA technology are discussed. The hardware architecture is described in VHDL and synthesized for a Virtex-6 device. Although the proposed field multiplier can be implemented for arbitrary finite fields, the targeted finite fields are recommended for Elliptic Curve Cryptography. Comparing other KOA multipliers, our proposed multiplier uses 36% less area resources and improves the maximum delay in 10%.**

***Index Terms* — Data security, Cryptography, Public key, Algorithm design and analysis, Field programmable gate arrays.**

I. INTRODUCTION

Nowadays binary field arithmetic has achieved great importance thanks to different applications like cryptography and error-correcting code. Several algorithms used by these applications are based on this kind of arithmetic [1]. Among them, one of the most relevant is Elliptic Curve Cryptography, which provides the same security levels as RSA but uses shorter key lengths, which is desirable for wireless and mobile environments.

Among binary field arithmetic operations, multiplication is one of the most expensive. Typically a multiplication on $\mathbb{GF}(2^m)$ is a two steps process: 1) a polynomial multiplication, and 2) a modular reduction step. The Karatsuba-Ofman Algorithm (KOA) [2] performs the first step. Techniques, such as Barret reduction [3] or Lazy reduction [4], can be used for modular reduction. Improving multiplication performance is tackled in [5]-[18].

There are different algorithms to perform binary field multiplications, such as the Montgomery [19], the FFT [20] and the Cantor [21] multipliers. The Karatsuba-Ofman algorithm [2] was the first to achieve below $O(n^2)$ complexity and, additionally it is well suited for hardware implementation because its structure is highly parallel.

In this paper, a novel $\mathbb{GF}(2^m)$ multiplier based on the original KOA algorithm that integrates the modular reduction step is introduced. Usually, the reduction step is performed independently and is not considered in the original KOA. The reduction step is executed by parallel linear feedback shift registers. An analysis on the theoretical cost in terms of area and maximum delay is carried out for the proposed multiplier and the classical KOA with a separate reduction step. This analysis considers an irreducible polynomial defining the finite field $\mathbb{GF}(2^m)$ as trinomials. The new KOA algorithm is developed on FPGA technology, using VHDL for hardware description and the Xilinx ISE tools for implementation. Results in terms of area and time are presented for finite fields recommended in Elliptic Curve Cryptography. The proposed multiplier improves resources usage and processing time when compared to the KOA algorithm with Classical Reduction.

The rest of this document is organized as follows: Section 2 explains LFSRs and their use in binary field arithmetic and realization in hardware. Section 3 explains the KOA algorithm for multiplication in $\mathbb{GF}(2^m)$ including the Classical Reduction step. Section 4 describes the proposed modification for the KOA algorithm and the hardware architecture is presented providing a comparison with the classical approach. Details of the architecture implementation and results are discussed in Section 5. Finally, conclusions of this research are drawn in Section 6.

II. LFSR AND PARALLEL LFSR (PLFSR) IN $\mathbb{GF}(2^m)$

A LFSR is a n-bit shift register that pseudo-randomly scrolls among 2^n-1 states at high speed [22]. It requires minimal logic to generate binary sequences. After reaching all states, the output sequence is repeated cyclically.

A LFSR of length n has n memory cells which together form the initial state $(s_0, s_1, \cdots, s_{n-1})$ of the shift register. The input bit for the LFSR is a linear function of its current or previous state. Several bits of the shift register value are driven by the XOR function, because this and the inverse-XOR are the only 1-bit linear functions. The selection of those bits is represented by a polynomial or characteristic polynomial over $\{0,1\}$. If the input bit for the LFSR is a linear function of bits s_0, s_1 and s_{n-1}, then the LFSR

Digital Object Identifier 10.4316/AECE.2013.02001

characteristic polynomial is $f(x)=1+x+x^{n-1}$; thus any LFSR can be represented as a x polynomial variable. In finite fields, this polynomial must be irreducible which means it cannot be split in the product of two polynomials.

An element $A(x)$ in the field $\mathbb{GF}(2^m)$ can be represented as a $(m-1)$-order polynomial as follows:

$$A(x)=\alpha_{m-1}x^{m-1}+\alpha_{m-2}x^{m-2}+\cdots+\alpha_1 x+\alpha_0 = \sum_{i=0}^{m-1}\alpha_i x^i \quad (1)$$

where $A(x)$ is normally represented as a m-bit vector containing all coefficients defining its corresponding polynomial, that is, $A(x)=(\alpha_{m-1},\alpha_{m-2},\cdots,\alpha_1,\alpha_0)$.

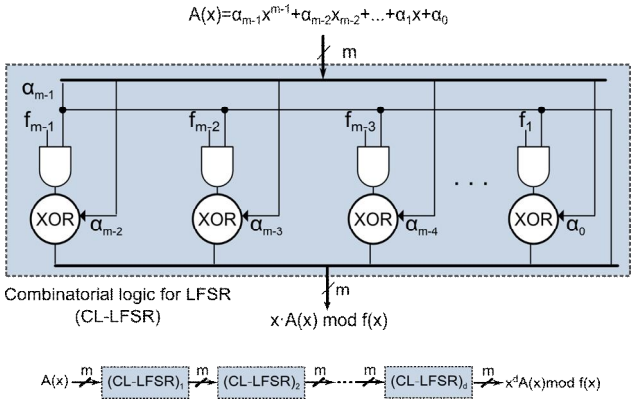
Addition (and subtraction) in $\mathbb{GF}(2^m)$ is a bitwise XOR operation between operands bit vectors resulting in a trivial operation in software or hardware. On the contrary, $\mathbb{GF}(2^m)$ multiplication and division are more expensive operations. These operations usually require the modular reduction step $A(x) \bmod f(x)$, where $f(x)$ is an irreducible m-order polynomial that generates $\mathbb{GF}(2^m)$. $f(x)$ is expressed by Equation (2), considering $f_m=f_0=1$.

$$f(x)=x^m+\sum_{i=1}^{m-1}f_i x^i+1=(1,f_{m-1},f_{m-2},\cdots,f_1,1) \quad (2)$$

Thus, $xA(x)$ becomes a shift to the left operation on $A(x)$ leading to a $(m+1)$-bit vector, $xA(x)=(\alpha_{m-1},\alpha_{m-2},\cdots.\ ,\alpha_1,\alpha_0,0)$. The resulting bit vector is the same with an extra '0' at the least significant position. If $\alpha_{m-1}=0$, a reduction is not necessary. However, if $a_{m-1}=1$, the resulting polynomial is reduced by $f(x)$, following $xA(x)\oplus f(x)$. Equation (3) defines $xA(x)$ considering $f_m=f_0=1$:

$$\begin{aligned} &xA(x) \bmod f(x) = \\ &(\alpha_{m-2}\oplus[f_{m-1}\odot\alpha_{m-1}],\alpha_{m-3}\oplus[f_{m-2}\odot\alpha_{m-1}],\cdots, \\ &\alpha_0\oplus[f_1\odot a_{m-1}],\alpha_{m-1}) \end{aligned} \quad (3)$$

where $\oplus$ represents a bitwise XOR operation and $\odot$ represents a bitwise AND operation. This expression is well modeled by the LFSR shown in Fig. 1a. The combinatorial logic (CL-LFSR) shown in Fig. 1a performs the required arithmetic to compute $xA(x) \bmod f(x)$. Therefore, d CL-LFSR blocks could be connected in a cascade fashion to implement a parallel LFSR (PLSFR) and to obtain $x^d A(x) \bmod f(x)$ in just one iteration, as it is shown in Fig. 1b. More details on the LFSR and the PLFSR are described in [19].



a.

b.

Figure 1. Linear feedback shift register structures. a. LFSR basic structure. b. Parallel LFSR (PLFSR) for computing $x^d A(x) \bmod f(x)$

III. Karatsuba-Ofman Algorithm for Multiplication in $\mathbb{GF}(2^m)$

A finite field $\mathbb{GF}(2^m)$ is defined by an irreducible m-order polynomial $f(x)$. Considering $A(x)$ and $B(x)$ being $(m-1)$-order polynomials in $\mathbb{GF}(2^m)$ with coefficients $\alpha_i,\beta_i\in\{0,1\}$. A field multiplication $A(x)\cdot B(x)$ in $\mathbb{GF}(2^m)$ results in another field element $C(x)$ that is computed in two steps:

1. $C'(x)=A(x)\cdot B(x)$, where $C'(x)$ is a $2m-2$-order polynomial.
2. Modular reduction, $C(x)=C'(x) \bmod f(x)$.

There are several algorithms to compute $C'(x)$, among them and widely known is the classical or Schoolbook method consisting of a shift-and-add scheme. Most of the proposed field multiplication algorithms are based on this method whose complexity is $O(n^2)$. In 1962, a multiplication algorithm was published by Karatsuba and Ofman [2] with $O(n^{\log_2 3})$ complexity. The KOA algorithm computes the first step of a field multiplication by using the *divide and conquer* technique. The multiplication $C'(x)$ is computed recursively using three field multiplications with low order operands.

For simplicity, lets consider that $m=2^t, t\geq 0$. This means that all field elements in $\mathbb{GF}(2^m)$ are power of 2 bit-vectors, and that $m/2$ is always a power of 2.

KOA splits the multiplier and multiplicand as it is shown in the following equation:

$$A(x)=(\underbrace{\alpha_{m-1},\alpha_{m-2},\cdots,\alpha_{m/2}}_{A^H},\underbrace{\alpha_{m/2-1},\cdots,\alpha_0}_{A^L}) \quad (4)$$

Thus, the next equations are sustained:

$$A(x)=A^H x^{m/2}+A^L \quad (5)$$

$$B(x)=B^H x^{m/2}+B^L \quad (6)$$

where A^H, A^L, B^H and B^L are $m/2$-order polynomials, and $C'(x)=A(x)\cdot B(x)$ is calculated as:

$$\begin{aligned} C(x) &= (A^H x^{m/2}+A^L)\cdot(B^H x^{m/2}+B^L) \\ &= z_2 x^m + z_1 x^{m/2} + z_0 \end{aligned} \quad (7)$$

where,

$$\begin{aligned} z_2 &= A^H\cdot B^H \\ z_1 &= A^H\cdot B^L + A^L\cdot B^H \\ z_0 &= A^L\cdot B^L \end{aligned} \quad (8)$$

At this point, $A(x)\cdot B(x)$ requires four multiplications with operands that are half the size the initial ones. KOA can be used recursively to compute these new multiplications and it reduces the number of multiplications to three at the cost of some more additions by redefining z_1,

as shown in Equation (9).

$$z_1 = (A^H + A^L)\cdot(B^H + B^L) - z_2 - z_0 \quad (9)$$

In $\mathbb{GF}(2^m)$, additions and subtractions are the same and are performed as bitwise XOR operations, thus redefining z_1 has no substantial cost. The recursive Karatsuba-Ofman method for multiplying two polynomials $A(x), B(x)$ in $\mathbb{GF}(2^m)$ is shown in Algorithm 1.

The KOA algorithm receives as input the multiplier and multiplicand as well as their bit-length (n). In the first call $n = m$. At each recursive call, operands are divided resulting in $n/2$-bit vectors. The recursive KOA finishes when $n = 1$, returning as a result the bitwise AND of $A(x)$ and $B(x)$.

Steps 7-8 in Algorithm 1 perform a recursive call to KOA and the resulting polynomials z_0, z_1 and z_2 are $(n-1)$-bit vectors. In step 9, the final multiplication $C'(x) = A(x)\cdot B(x)$ is calculated, resulting in a $(2n-1)$-order polynomial. When all recursive calls are finished, the final result is a $(2m-1)$-order polynomial. A graphical representation of $C'(x)$ operation is depicted in Fig. 2.

Up to this point, it is assumed that $m = 2^k$, but in many applications, such as cryptography, m is not a power of 2. One strategy is padding with 0's the bit vector representation of the input operands until reaching a power of 2 length, but with this strategy many gates remain unused. Thus, a modification to KOA called Binary Karatsuba Multiplier. (BKM) was proposed in [17]. More details on this technique are provided next.

A. Reduction step

Algorithm 1 calculates $C'(x) = z_2 x^m + z_1 x^{m/2} + z0$, where $C'(x)$ is a $(2m-1)$-bit vector that does not belong to $\mathbb{GF}(2^m)$ and needs to be reduced $\bmod f(x)$.

For general irreducible polynomials $f(x)$, specialized reduction methods must be applied, such as the Barret reduction method [23] or the Montgomery method [24].

Algorithm 1 KOA[n , $A(x)$, $B(x)$]: Recursive Karatsuba-Ofman Algorithm

Input: $n = 2^t, t \geq 0, n \leq m$;
$A(x)$, $B(x)$ n-bit vectors

Output: $C'(x) = A(x)\cdot B(x)$

1: **IF** $n = 1$
2: **RETURN** $A \odot B$
3: **ENDIF**
4: $A(x) \leftarrow A^H(x)x^{n/2} + A^L(x)$
5: $B(x) \leftarrow B^H(x)x^{n/2} + B^L(x)$
6: $z_2 \leftarrow$ **KOA**$[n/2, A^H, B^H]$
7: $z_0 \leftarrow$ **KOA**$[n/2, A^L, B^L]$
8: $z_1 \leftarrow$ **KOA**$[n/2, (A^L + A^H), (B^L + B^H)] + z_2 + z_0$
9: $C'(x) \leftarrow z_2 x^n + z_1 x^{n/2} + z_0$
10: **RETURN** $C'(x)$

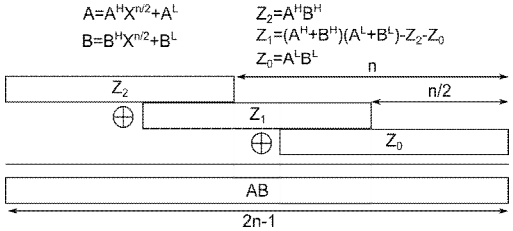


Figure 2. C(x) computation at step 9 of KOA algorithm

For special $f(x)$ classes, such as trinomials and pentanomials, the reduction step of KOA algorithm can be performed using a matrix of XOR gates [25]. This technique has been used in KOA hardware implementations [5], [16], [17]. The reduction technique is based on the fact that if $f(x) = x^m + g(x)$, where $g(x)$ is a low order $(< m)$ polynomial, the equivalence $x^m \equiv g(x) \mod f(x)$ is sustained. Therefore, a trinomial $f(x)$ with form $f(x) = x^m + x^a + 1$ expresses $C'(x)$ polynomial in the following way:

$$\begin{aligned} C'(x) &= \sum_{i=0}^{2m-2} c_i x^i = \sum_{i=0}^{m-1} c_i x^i + \sum_{i=m}^{2m-2} c_i (x^{a+i-m} + x^{i-m}) \\ &= \underbrace{\sum_{i=0}^{m-1} c_i x^i}_{(1)} + \underbrace{\sum_{i=0}^{m-1-a} c_{i+m} x^{a+i}}_{(2)} + \underbrace{\sum_{i=0}^{a-1} c_{2m-a+i} x^{a+i}}_{(3)} + \\ &\quad \underbrace{\sum_{i=0}^{a-1} c_{2m-a+i} x^i}_{(4)} + \underbrace{\sum_{i=0}^{m-1} c_{m+i} x^i}_{(5)} \end{aligned} \quad (10)$$

The last expression in Equation (10) states that $C(x)$ can be formulated as a m-bit vector that results from adding five terms obtained from $C'(x)$, achieving the desired reduction $\bmod f(x)$. Graphically, this reduction is shown in Fig. 3.

B. Theoretical cost analysis for KOA with Classical Reduction

Let S be the cost in area of a KOA hardware implementation. If $m = 1$, the total cost is only one 1-bit AND gate. If $m > 1$, the total cost is given by three KOA recursive calls with half size operands: $3 \cdot S_{m/2}$. In addition, the following XOR gates are also needed:

- Two $(n/2)$ XOR gates to perform $(A^L + A^H)$ and $(B^L + B^H)$, Algorithm 1, step 8.
- Two $(n-1)$ XOR gates to add three $(n-1)$-bit numbers, Algorithm 1, step 8.
- One $(n-1)$-bit XOR to concatenate z_2 and z_1, Algorithm 1, step 9.

The total number of XOR gates required is $4n-3$. The reduction step cost is given by the number of XOR gates necessary to add five terms of Equation (10), which is $2m + a$, where a corresponds to the power of the second term in the irreducible polynomial $f(x)$.

The total area cost for the KOA algorithm considering the Classical Reduction technique for trinomials, is given by the

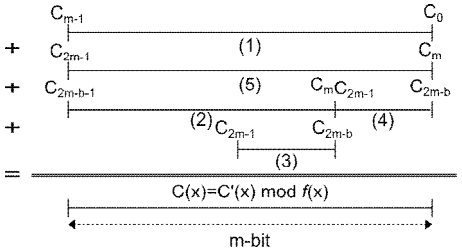


Figure 3. $C(x) = C'(x) \bmod f(x)$ computation, where $f(x)$ is a trinomial.

recurrence in the next equation:

$$S_n = \begin{cases} 1 \cdot AND & \text{if } n = 1 \\ 3S_{n/2} + 4n - 3 \cdot XOR & \text{if } n \neq m \\ 3S_{n/2} + 6n + b - 3 \cdot XOR & \text{if } n = m \end{cases} \quad (11)$$

where $m = 2^t$, for any $t \geq 0$.

Considering T as the time delay required for hardware implementation. Thus, T_A and T_X are the maximum delays for AND and XOR gates respectively. If $m = 1$, the total KOA delay is given by the AND gate in Algorithm 1, line 2. If $m > 1$, the three KOA calls in Algorithm 1, lines 7-8 can be performed in parallel, each with $T_{m/2}$ time cost. However, KOA call at step 8 requires two additions, similar to the previous step, ($A^H + A^L$) and ($B^H + B^L$). These two operations can be performed in parallel with T_X delay. Once step 8 is completed, another two additions are required to calculate the cost of $2T_X$. Finally, one addition is required at step 9, to add one T_X delay that leads to a total cost of $T_{m/2} + 4T_X$. Thus, the delay for the reduction step is $3T_X$. Time complexity for KOA algorithm is given by the recurrence in the following equation:

$$T_n = \begin{cases} T_A & \text{if } n = 1 \\ T_{n/2} + 4T_X & \text{if } n \neq m \\ T_{n/2} + 7T_X & \text{if } n = m \end{cases} \quad (12)$$

IV. KARATSUBA-OFMAN WITH MODULAR REDUCTION INTEGRATED

In this section the novel KOA algorithm proposed in this research, which integrates the modular reduction step within the algorithmic procedure, is presented. Considering Algorithm 1, in step 9, z_2 is multiplied by x^n and z_1 by $x^{n/2}$. The proposed approach takes advantage of the module operation and integrates the modular reduction step within KOA algorithm through Equation (13).

$$\begin{aligned} C(x) &= C'(x) \bmod f(x) \\ &= (z_2 x^n + z_1 x^{n/2} + z0) \bmod f(x) \\ &= z_2 x^n \bmod f(x) + z_1 x^{n/2} \bmod f(x) + z_0 \end{aligned} \quad (13)$$

In the previous section, it was demonstrated that Equation (13) can be solved using LFSR. Following this approach, two PLFSR are required to compute $z_2 x^n \bmod f(x)$ and $z_1 x^{n/2} \bmod f(x)$. Therefore, the total number of shifts required are: n shifts for $z_2 x^n \bmod f(x)$ and $n/2$ shifts for $z_1 x^{n/2} \bmod f(x)$.

In order to identify the exact KOA's algorithmic stage where this modification needs to take place, because most applications do not use $m = 2^t$, the field size is generalized to be of any size. Thus, the BKM technique is used as a starting point [17]. BKM considers that $m = 2^k + d$, where 2^k is the largest power of 2 that is smaller than m, and d are the remainder bits. Then, instead of splitting the input polynomial in two equal size bit-vectors, both input polynomials are split according to the next equation:

$$A(x) = (\underbrace{\alpha_{m-1}, \cdots, \alpha_{2^k}}_{A^H}, \underbrace{\alpha_{2^k-1}, \cdots, \alpha_0}_{A^L}) \quad (14)$$

where A^H is a d-bit vector and A^L is a 2^k-bit vector.

The BKM strategy is used to analyze where the PLFSRs are needed. In the first call $n = m$, the resulting polynomial's order is $2m-1$, thus a reduction is needed. For computing z_0 and z_1, input operands size is 2^k, and results size is $2^{k+1}-1$. Because 2^k is the highest power of 2 that is smaller than m; $2^{k+1}-1$ is consequently higher than m, therefore a reduction is needed for z_0 and z_1. To compute z_0, if $d > m/2$, the result is $2d-1 > m$ and a reduction is also necessary.

Before analyzing subsequent recursive calls in the BKM strategy, it is observed that several PLFSRs are required resulting in an expensive hardware architecture. Hence, a different strategy to optimize the number of PLFSRs is approached. The proposed strategy is similar to that used in [7] and [12]. It consists in splitting the input bit vectors by half using the function ceiling $\lceil \cdot \rceil$ to ensure an integer result, since m could be an odd number, see next equation:

$$A(x) = (\underbrace{\alpha_{m-1}, \alpha_{m-2}, \cdots, \alpha_{\lceil m/2 \rceil}}_{A^H}, \underbrace{\alpha_{\lceil m/2 \rceil - 1}, \cdots, \alpha_0}_{A^L}) \quad (15)$$

where A^L size is $\lceil m/2 \rceil$ and A^H size is $m - \lceil m/2 \rceil$. This approach is used in every algorithmic recursive call. To determine if a reduction is necessary, the first call is assessed with $n = m$, the input operands size is m and the result size is $2m-1$, thus a reduction is necessary. For z_0 and z_1, inputs size is $\lceil m/2 \rceil$; if m is even result size is $m-1$, if m is odd result size is m. Reduction is not necessary in both cases. For z_2, inputs size is $m - \lceil m/2 \rceil$; if m is even result size is $m-1$, if m is odd result size is $m-2$. Thus, reduction is not required. In recursive calls, operands are smaller and no more reductions are needed. In total, only one reduction at the first call is necessary following the proposed approach.

Algorithm 2 presents the proposed novel Karatsuba-Ofman algorithm based on LFSR. It is worth noticing that $C(x)$ result is already reduced mod $f(x)$. Steps 4 and 5 use the splitting strategy explained before in Equation (15) whereas steps 6-8 perform the recursive calls. Step 9 evaluates $n = m$ which is true only for the first call when using PLFSRs. For the rest of the calls $n \neq m$ and partial results sizes are smaller than m therefore a reduction is not needed.

As an example, consider the binary field $\mathbb{GF}(2^{163})$, see Fig. 4. During the first call, KOA-LFSR splits the input operands following Equation (15) and makes three recursive

Algorithm 2 KOA-LFSR[n , $A(x)$, $B(x)$]:
Recursive Karatsuba-Ofman Algorithm with reduction step integrated using LFSR for field multiplication in $\mathbb{GF}(2^m)$

Input: n an integer smaller or equal to m ; $A(x)$, $B(x) \in \mathbb{GF}(2^m)$

Output: $C(x) = A(x) \cdot B(x) \mod f(x)$

1: **IF** $n = 1$
2: **RETURN** $A \odot B$
3: **ENDIF**
4: $A(x) \leftarrow A^H(x)x^{\lceil n/2 \rceil} + A^L(x)$
5: $B(x) \leftarrow B^H(x)x^{\lceil n/2 \rceil} + B^L(x)$
6: $z_2 \leftarrow$ **KOA-LFSR**$[n - \lceil n/2 \rceil, A^H, B^H]$
7: $z_0 \leftarrow$ **KOA-LFSR**$[\lceil n/2 \rceil, A^L, B^L]$
8: $z_1 \leftarrow$ **KOA-LFSR**$[\lceil n/2 \rceil, (A^L + A^H), (B^L + B^H)] + z_2 + z_0$
9: **IF** $n = m$
10: $C(x) \leftarrow z_2 x^{2\lceil n/2 \rceil} \bmod f(x) + z_1 x^{\lceil n/2 \rceil} \bmod f(x) + z_0$
11: **ELSE**
12: $C(x) \leftarrow z_2 x^{2\lceil n/2 \rceil} + z_1 x^{\lceil n/2 \rceil} + z_0$
13: **ENDIF**
14: **RETURN** $C(x)$

calls (steps 6-8) with $n \in \{95, 96, 96\}$ at a 2nd recursion level. In the second call $n = 95$ and KOA-LFSR splits the inputs again and invokes three recursive calls with $n \in \{47, 48, 48\}$. For $n = 96$, recursive calls are with $n \in \{48, 48, 48\}$, this call is at the third recursion level. Splits and recursive calls continue until the basic case where $n = 1$ is performed and multiplication is carried out with a simple AND gate.

Fig. 5 shows a diagram with the fully parallel KOA algorithm based on the LFSR multiplier. In Fig. 5a, the first call case using PLFSRs is shown. In Fig. 5b the recursive calls case is drawn, where simple shifts are used instead of PLFSR.

A. Theoretical cost analysis for KOA-LFSR multiplier

This novel approach leads to the next space and time complexity analysis. To simplify this analysis, only the special case of having an even m is considered, that is: $\lceil m/2 \rceil = m/2$. It has been stated that PLFSRs are used only in the first algorithmic call. Thus, all recursive calls, where $n \neq m$, have the same cost according to Equation(11).

In the first KOA-LFSR call, two $(m/2)$ XOR gates to perform $(A^L + A^H)$ and $(B^L + B^H)$ are needed, see Algorithm 2, step 8. Also, two $(m-1)$ XOR gates to add z_0, z_2 and $(A^L + A^H)(B^L + B^H)$ are required. Finally, two $(m-1)$ XOR gates to compute the addition of step 10 in Algorithm 2 are needed. To calculate the PLFSR cost, a special case with $f(x)$ as a trinomial of the form $f(x) = x^m + x^a + 1$ is considered. It is observed that only $f_m = f_{m-a} = f_0 = 1$, as a consequence, most AND gates have

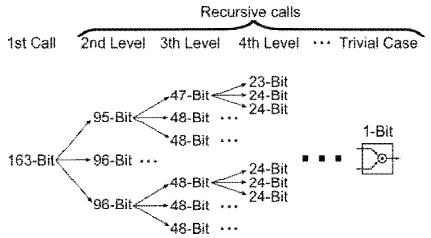


Figure 4. Recursive calls tree for KOA-LFSR algorithm for multiplication in $\mathbb{GF}(2^{163})$.

0 outputs, and most XOR gates perform a simple right shift ($\alpha_{m-i} \leftarrow \alpha_{m-i-1}$), see Equation (3). Simple shifts have no cost in hardware; therefore from all gates in Fig. 1a, only one XOR gate is necessary to calculate $xA(x)$ when $f(x)$ is a trinomial. For pentanomials, three XOR gates would be required. Because the number of shifts is fixed ($2\lceil m/2 \rceil$ for $z_2 x^{2\lceil m/2 \rceil} \bmod f(x)$ and $\lceil m/2 \rceil$ for $z_1 x^{\lceil m/2 \rceil} \bmod f(x)$), the number of XOR needed for these reductions is the number of shifts multiplied by the number of XORs per shift. A total of $3m/2$ XOR gates for trinomials and $9m/2$ for pentanomials are required which is an improvement over the traditional approach. Finally, Equation (16) summarizes trinomials cost:

$$S_n = \begin{cases} 1 \cdot AND & \text{if } n = 1 \\ 3S_{n/2} + 4n - 3 \cdot XOR & \text{if } n \neq m \\ 3S_{n/2} + 11n/2 - 3 \cdot XOR & \text{if } n = m \end{cases} \quad (16)$$

Time delay remains as formulated by Equation (12) for all recursive calls with $n \neq m$, that is $T_{n/2} + 4T_X$. For the first call, PLFSR add some extra delay. Having PLFSR imply several LFSR connected in cascade, however corresponding time delay is not equivalent to the number of LFSRs. A XOR is also shifted and requires m shifts to return to the

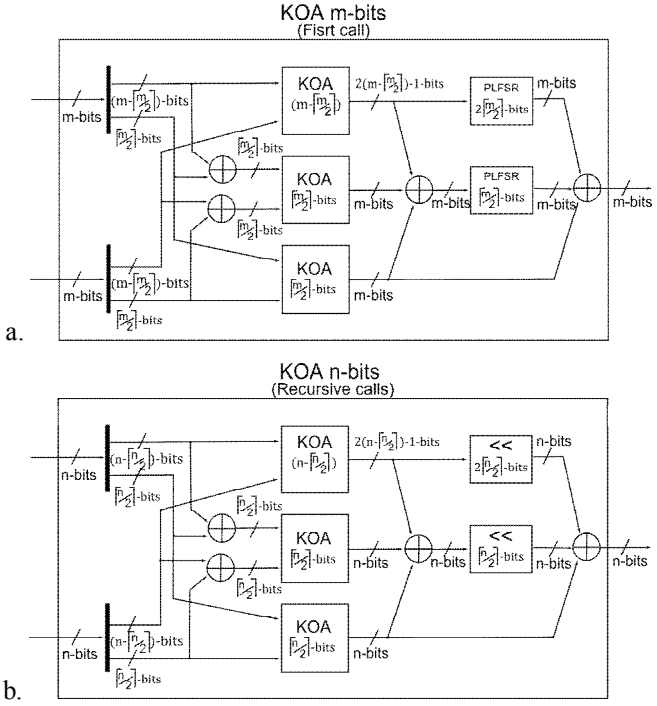


Figure 5. Fully Parallel Karatsuba-Ofman Multiplier based on LFSR for $\mathbb{GF}(2^m)$. a. First call ($n = m$). This call uses Parallel Linear Feedback Shift Registers. b. Recursive calls only use simple shift.

TABLE I. AREA COST (S_n) AND TIME DELAY (T_n) FOR KOA WITH CLASSICAL REDUCTION AND THE PROPOSED KOA-LFSR CONSIDERING THE TRINOMIAL CASE.

Parameter	Classical KOA	KOA-LFSR	Case
S_n	$1 \cdot AND$	$1 \cdot AND$	if $n=1$
	$3S_{n/2}+4n-3 \cdot XOR$	$3S_{n/2}+4n-3 \cdot XOR$	if $n \neq m$
	$3S_{n/2}+6n+b-3 \cdot XOR$	$3S_{n/2}+11n/2-3 \cdot XOR$	if $n=m$
T_n	T_A	T_A	if $n=1$
	$T_{n/2}+4T_X$	$T_{n/2}+4T_X$	if $n \neq m$
	$T_{n/2}+7T_X$	$T_{n/2}+6T_X$	if $n=m$

original position. Most gates are simple shifts when using trinomials and pentanomials, such as $f(x)$, with no hardware cost. Because at most there are $2\lceil m/2 \rceil$ LFSRs connected, only 2 XOR gates are actually in cascade. For trinomials, which are the worst case, the PLFSR time delay is $2T_X$. Hence, the time delay for the first call is $T_{n/2}+4T_X+2T_X$. For pentanomials, PLFSR's delay is $4T_X$, because when a XOR returns to the original position, it also goes through other 2 XOR gates. In this case, the first call delay is $T_{n/2}+4T_X+4T_X$. The overall time delay for the proposed multiplier considering trinomials is expressed by the next equation:

$$T_n = \begin{cases} T_A & \text{if } n=1 \\ T_{n/2}+4T_X & \text{if } n \neq m \\ T_{n/2}+6T_X & \text{if } n=m \end{cases} \quad (17)$$

In Table I, a theoretical cost comparison for the KOA algorithm with Classical Reduction and the proposed KOA-LFSR is presented considering the trinomial case. It is observed that the proposed KOA-LFSR algorithm achieves a reduction in hardware cost and in time delay required to implement the multiplier on a hardware platform.

V. Architecture Implementation and Results Analysis

To validate the proposed modification of the KOA algorithm, a fully parallel Karatsuba-Ofman Multiplier has been designed, simulated and synthesized. Different fields with irreducible polynomials considering both, trinomials and pentanomials are assessed, see Table II. These polynomials define finite fields recommended by the NIST for cryptographic applications [26], while the others are proposed by CERTICOM as a challenge[1]. For comparative purposes, results for a fully parallel Binary Karatsuba Multiplier using the Classical Reduction are presented.

The proposed architecture was implemented using VHDL as a description language. For design validation, a C routine to generate test data vectors was created and ModelSim PE Student Edition 10.1c was used as simulation environment. For synthesis, Xilinx ISE 13.2 was used targeting a Xilinx Virtex-6 (xc6vlx240t) device.

Fig 6 shows the synthesis results for trinomials and pentanomials. These graphs show the tendency of used LUTs and the minimum clock period achieved by each architecture. These results include the total hardware usage necessary for the multiplication and the reduction steps.

In Fig. 6, the improvement for the proposed KOA-LFSR algorithm in time and area is presented when compared to the BKM technique with Classical Reduction. Theoretical cost for the BKM technique with Classical Reduction is not provided.

These results not only confirm the theoretical improvement shown in Table I, but also demonstrate that the proposed multiplier helps the synthesis tool to optimize the FPGA's resources usage. The KOA-LFSR algorithm has a very regular structure from which the synthesis tool takes advantage and optimizes the result. In Fig. 6, the area and time tendency, when the field size increases, are observed showing a better performance for the proposed KOA-LFSR algorithm.

In Table III, the proposed multiplier is compared to a different Karatsuba-Ofman Multiplier using the same device. A direct comparison with other works is difficult because to the best of our knowledge, other works do not consider the cost of the KOA multiplication and the reduction step together. Some authors only work on the polynomial multiplier; others focus on the reduction for general polynomials. Moreover, in order to compare different hardware architectures, the same FPGA devices should be considered, because it would not be fair to compare the required area on a 4-in LUT FPGA versus a 6-in LUT FPGA.

In [5], a multiplier based on the BKM technique is presented. It truncates the recursion at a predefined number of bits and then uses a more efficient multiplier. The idea in this work is that for small multipliers there are better multipliers than the KOA approach. Thus, in this work the

TABLE II. Irreducible polynomials used to validate KOA-LFSR.

$f(x)=$	Recommended
$x^{167}+x^6+1$	
$x^{191}+x^9+1$	Certicom
$x^{233}+x^{74}+1$	NIST
$x^{239}+x^{36}+1$	Certicom
$x^{359}+x^{68}+1$	NIST
$x^{409}+x^{87}+1$	NIST
$x^{131}+x^{13}+x^2+x+1$	Certicom
$x^{163}+x^7+x^6+x^3+1$	NIST
$x^{277}+x^{12}+x^6+x^3+1$	
$x^{283}+x^{12}+x^7+x^5+1$	NIST
$x^{571}+x^{10}+x^5+x^2+1$	NIST

[1] http://www.certicom.com/index.php/curves-list, August 2012

recursion is truncated at different levels. Experimental tests with $n \in \{4,8,16\}$ are presented with best results are achieved when $n = 8$. This work reports the number of slices used and the time required. As a reduction step, the classical method explained in Section III.A is used.

In [9], a KOA based multiplier with pipelining is presented. This multiplier truncates KOA's recursive calls after some steps and thereafter the Classic Method is used. Pipeline registers are placed between every KOA recursive call. The proposed approach is compared to its more similar experimental case. This design is assessed considering several pipeline stages in order to find the best compromise between area and time. The used modular reduction strategy is not explicitly mentioned.

In [7], authors perform a detailed analysis of several KOA-based multipliers implemented in FPGAs and ASICs. This work considers multipliers that are a mix of the KOA and the Classic algorithms. First, it analyzes separately both approaches and realizes that the classic method is better for small fields. Then, it implements a KOA multiplier that truncates recursive calls and executes small multipliers with the classic method. The KOA multiplier used on that work uses a splitting strategy very similar to the one used in this research. In [7], experiments with several multipliers were carried out. In order to provide a fair comparison for the research herein presented, those approaches that do not consider the modular reduction but which are closely related to this study were chosen. That work also presents place-and-route results, however a direct comparison is not possible because the classic method implementation is carried out manually. Their results show the number of LUTs required in their design.

In [8], several combinations of parallel and sequential multipliers are provided. Results for a sequential 240-bit multiplier are presented, for comparison with the proposed KOA-LFSR approach a 239-bit multiplier is selected.

In [10], the number of slices used by the architecture is reported, for comparison the same parameter has been used. This paper explores different architectures of Karatsuba multipliers, some of them are fully parallel while others are a hybrid of parallel and sequential multipliers. The fastest (fully parallel) and the smallest architectures are shown. The reduction step is not considered in this research. Because, m is only considered as a power of 2, fields closer to 128 and 256 are chosen for comparison. Exact 128 and 256 fields are not selected because to the best of our knowledge, there are not irreducible polynomials reported for these fields.

VI. Conclusions

In this paper, a novel multiplier called KOA-LFSR has been presented. The proposed approach is a modification of the original Karatsuba-Ofman algorithm (KOA) to perform modular multiplication in $\mathbb{GF}(2^m)$. Contrary to the original Karastuba-Ofman multiplier that performs only the multiplication step, the KOA-LFSR performs both multiplication and modular reduction. An array of Linear Feedback Shift Registers connected in cascade to carry out the reduction is used, this array is computed during KOA recursive calls. The proposed multiplier performs better than the original KOA with Classical Reduction, saving area resources and achieving better timing. It is important to notice that the way of splitting the input operands is crucial for achieving an optimal performance. The splitting of input operands as shown in Equation (15) resulted in the best way to integrate the reduction step in the KOA algorithm. Because the LFSR is a regular and compact module, the synthesis tool optimally mapped this module leading to a better usage of hardware resources. For future work a hybrid multiplier will be tackled, where recursive calls can be truncated at a specific value and simpler multipliers would be used such as the Scholarbook one or multipliers embedded in the same FPGA device.

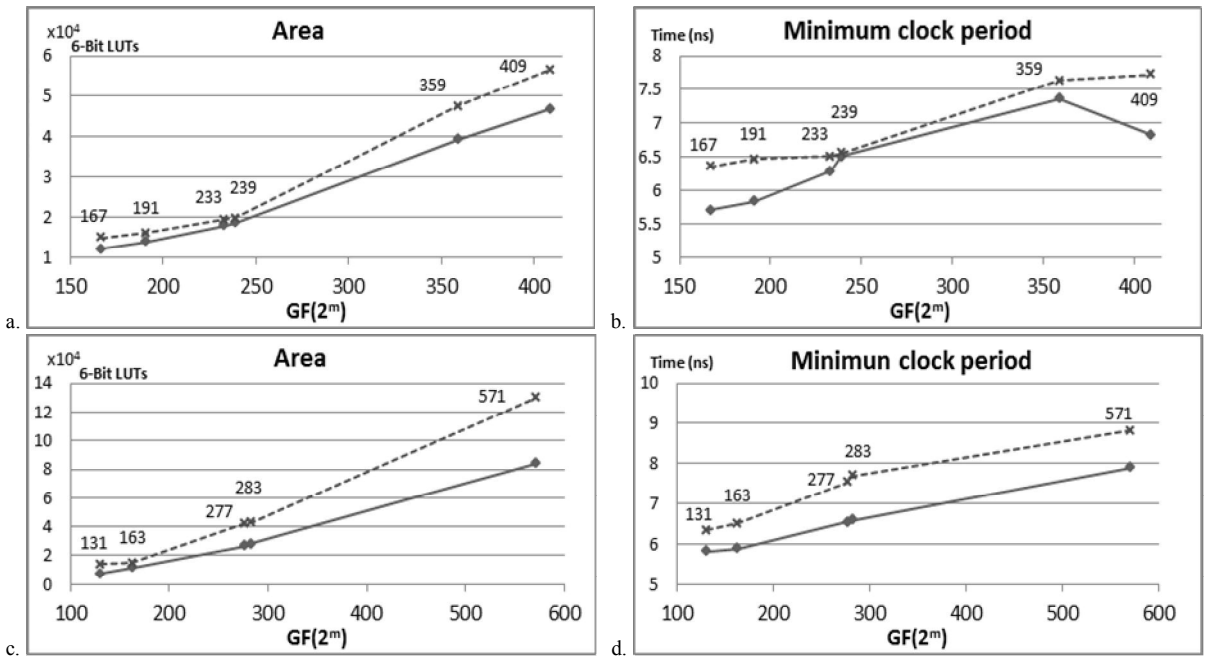


Figure 6. Area and delay for Fully Parallel KOA-LFSR Multiplier for $\mathbb{GF}(2^m)$. - - - BKM with Classical Reduction. —— Proposed Multiplier
a. Trinomials area (LUTs). b. Trinomials minimum clock period (ns). c. Pentanomials area (LUTs). d. Pentanomials minimum clock period (ns).

TABLE III. COMPARATIVE BETWEEN THIS ARCHITECTURE AND OTHER JOBS.

Ref.	Device	*m*	Fully Parallel	Includes reduction	Area	Time (ns)
[5]	Virtex E	191	YES	YES	6265 Slices	45.889
This	Virtex E	191	YES	YES	7093 Slices	19.308
[7]	Virtex 5	163	YES	NO	7488 LUTs	--
This	Virtex 5	163	YES	YES	7786 LUTs	5.468
[8]	Virtex II	128	NO	NO	2473 Slices	378
[8]	Virtex II	128	NO	NO	3978 Slices	153
This	Virtex II	131	YES	YES	4147 Slices	10.122
[8]	Virtex II	240	NO	NO	4839 Slices	290
This	Virtex II	239	YES	YES	10510 Slices	10.710
[9]	Virtex 5	128	YES	YES	6941 LUTs	5.487
This	Virtex 5	131	YES	YES	6162 LUTs	5.753
[10]	Spartan 3	128	YES	NO	10172 Slices	59.52
[10]	Spartan 3	128	NO	NO	2528 Slices	515.64
This	Spartan 3	131	YES	YES	4205 Slices	13.454
[10]	Spartan 3	256	YES	NO	--	69.77
[10]	Spartan 3	256	NO	NO	8276 Slices	569.04
This	Spartan 3	239	YES	YES	13620 Slices	15.046

REFERENCES

[1] B. Schneier, Applied Cryptography, 2nd edition. Wiley, 1996, p. 758.

[2] A. Karatsuba and Y. Ofman, "Multiplication of Multidigit Numbers on Automata," Soviet Physics-Doklady, vol. 7, no. 7, pp. 595–596, 1963.

[3] M. Knezevic, F. Vercauteren, and I. Verbauwhede, "Faster Interleaved Modular Multiplication Based on Barrett and Montgomery Reduction Methods," IEEE Transactions on Computers, vol. 59, no. 12, pp. 1715–1721, Dec. 2010.

[4] G. X. Yao, J. Fan, R. C. C. Cheung, and I. Verbauwhede, "A High Speed Pairing Coprocessor Using RNS and Lazy Reduction," IACR Cryptology ePrint Archive, vol. 2011, p. 258, 2011.

[5] A. B. El-sisi, S. M. Shohdy, and N. Ismail, "Reconfigurable Implementation of Karatsuba Multiplier for Galois Field in Elliptic Curves," Novel Algorithms and Techniques in Telecommunications and Networking, pp. 97–92, 2010.

[6] H. Fan, J. Sun, M. Gu, and K.-Y. Lam, "Overlap-free Karatsuba-Ofman polynomial multiplication algorithms," IET Information Security, vol. 4, no. 1, p. 8, 2010.

[7] G. Zhou, H. Michalik, and L. Hinsenkamp, "Complexity Analysis and Efficient Implementations of Bit Parallel Finite Field Multipliers Based on Karatsuba-Ofman Algorithm on FPGAs," IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 18, no. 7, pp. 1057–1066, Jul. 2010.

[8] M. Machhout, M. Zeghid, W. El Hadj Youssef, B. Bouallegue, A. Baganne, and R. Tourki, "Efficient Large Numbers Karatsuba-Ofman Multiplier Designs for Embedded Systems," in Conference of the World Academy of Science Engineering and Technology 28, 2009, pp. 992–1001.

[9] G. Zhou, H. Michalik, and L. Hinsenkamp, "Improving Throughput of AES-GCM with Pipelined Karatsuba Multipliers on FPGAs," Reconfigurable Computing: Architectures, Tools and Applications, vol. 5453, pp. 193–203, 2009.

[10] W. El hadj youssef, M. Machhout, M. Zeghid, B. Bouallegue, and R. Tourki, "Efficient hardware architecture of recursive Karatsuba-Ofman multiplier," in 2008 3rd International Conference on Design and Technology of Integrated Systems in Nanoscale Era, 2008, pp. 1–6.

[11] Y. L. Zhang, G. C. Shou, Y. H. Hu, and Z. G. Guo, "Low Complexity GF(2m) Multiplier Based on Iterative Karatsuba Algorithm," Advanced Materials Research, vol. 546–547, pp. 1409–1414, Jul. 2012.

[12] A. Weimerskirch and C. Paar, "Generalizations of the Karatsuba Algorithm for Efficient Implementations," Cryptology ePrint Archive, vol. 2006/224, 2006.

[13] J. von zur Gathen and J. Shokrollahi, "Efficient FPGA-Based Karatsuba Multipliers for Polynomials over F2," Selected Areas in Cryptography, vol. 3897, pp. 359–369, 2006.

[14] N. S. Chang, C. H. Kim, Y.-H. Park, and J. Lim, "A Non-redundant and Efficient Architecture for Karatsuba-Ofman Algorithm," Information Security, vol. 3650, pp. 288–299, 2005.

[15] N. A. Saqib, F. Rodríguez-Henríquez, and A. Díaz-Pérez, "A parallel architecture for fast computation of elliptic curve scalar multiplication over GF(2m)," in 18th International Parallel and Distributed Processing Symposium, 2004. Proceedings., 2004, vol. 00, no. C, pp. 144–151.

[16] M. Ernst, M. Jung, F. Madlener, S. A. Huss, and R. Bl, "A Reconfigurable System on Chip Implementation for Elliptic Curve Cryptography over GF(2m)," in Cryptographic Hardware and Embedded Systems - CHES 2002, vol. 2523, B. Kaliski, C. Koc, and C. Paar, Eds. Springer Berlin / Heidelberg, 2003, pp. 381–399.

[17] F. Rodríguez-Henríquez and C. K. Koc, "On Fully Parallel Karatsuba Multipliers for GF(2m)," in Computer Science and Technology 2003, 2003.

[18] M. Jung, F. Madlener, M. Ernst, and S. A. Huss, "A Reconfigurable Coprocessor for Finite Field Multiplication in GF(2m)," in IEEE Workshop on Heterogeneous Reconfigurable Systems on Chip (HRSoc'02), 2002.

[19] M. Morales-Sandoval, C. Feregrino-Uribe, and P. Kitsos, "Bit-serial and digit-serial GF(2m) Montgomery multipliers using linear feedback shift registers," IET Computers & Digital Techniques, vol. 5, no. 2, p. 86, 2010.

[20] J. von zur Gathen and J. Gerhard, "Arithmetic and factorization of polynomial over (extended abstract)," in Proceedings of the 1996 international symposium on Symbolic and algebraic computation - ISSAC '96, 1996, pp. 1–9.

[21] D. G. Cantor, "On arithmetical algorithms over finite fields," Journal of Combinatorial Theory, vol. 50, no. 2, pp. 285 – 300, 1989.

[22] M. Abramovici, M. A. Breuer, and A. D. Friedman, Digital Systems Testing and Testable Design, 1st ed. WILEY-IEEE PRESS, 1994.

[23] M. M. Knezevic, K. Sakiyama, J. Fan, and I. Verbauwhede, "Modular Reduction in GF(2m) without Precomputational Phase," in International Workshop on the Arithmetic of Finite Fields (WAIFI 2008), 2008, vol. 5130, pp. 77–87.

[24] C. K. Koc, "Montgomery reduction with even modulus," IEE Proceedings of Computers and Digital Techniques, vol. 141, no. 2, pp. 314–316, 2010.

[25] F. Rodríguez-Henríquez, A. Díaz-Pérez, N. A. Saqib, and C. K. Koc, Cryptographic Algorithms on Reconfigurable Hardware. Boston, MA: Springer US, 2006.

[26] "FIPS PUB 186-3 Digital Signature Standard (DSS)," NIST - Federal Information Processing Standars Publication, 2009.