A New Low Complexity Parallel Multiplier for a Class of Finite Fields

Manuel Leone

Telecom Italia Lab,
via G. Reiss Romoli 274, 10148 Torino, Italy
manuel.leone@tilab.com

Abstract. In this paper a new low complexity parallel multiplier for characteristic two finite fields $GF(2^m)$ is proposed. In particular our multiplier works with field elements represented through both Canonical Basis and Type I Optimal Normal Basis (ONB), provided that the irreducible polynomial generating the field is an All One Polynomial (AOP). The main advantage of the scheme is the resulting space complexity, significantly lower than the one provided by the other fast parallel multipliers currently available in the open literature and belonging to the same class.

1 Introduction

Finite fields have recently attracted a lot of attention due to the increasing number of cryptography and coding theory applications that require high performance finite field capabilities ([9]). Several new architectures have been proposed in order to fulfill the constraints imposed by specific purposes ([2,8,10]). Although different solutions can be compared from several points of view, *time complexity* and *space complexity* are, usually, the two most important parameters. The former is defined as the elapsed time between input and output of the circuit implementing the multiplier, and it is usually expressed as a function of the field degree m, the delay of an AND gate T_A and the delay of an XOR gate T_X. The latter, on the contrary, is defined as the pair of numbers Σ_A and Σ_X, of AND and XOR gates used respectively. Although a manifest improvement in space complexity over the best known algorithm is still possible, because of an achievable asymptotic space complexity given by $O(m \log_2 m \log_2 \log_2 m)$ ([1]), these two parameters are characterized by an evident trade off. In fact, reducing the number of gates causes, in general, a corresponding increase in the execution time. So, if performance is the most critical parameter, we can accept a greater space complexity, in exchange for a reduction of the corresponding time delay. Conversely, in other applications such as those based on smart cards, mobile phones, or other portable devices, a reduced space complexity is often the most important design aspect.

Because of these reasons we will focus on a special class of fast multipliers, characterized by a generator of type AOP, which can take advantage of the trade off between time and space complexity to achieve a space complexity significantly lower than those offered by the traditional bit-parallel multipliers of the same class ([3,4,5,6,7]), with a small increase in the corresponding time delay. In other words, a limited rise in the time complexity is accepted in order to obtain a more consistent reduction in the corresponding circuit area.

Therefore the paper is organized as follows: section two introduces some useful preliminaries; section three provides an architectural description of the multiplier when the field elements are represented through a Canonical Basis, while section four focuses on Type I ONB representations. The last section summarizes the results obtained and draws some conclusions.

2 Preliminaries

Characteristic two finite fields $GF(2^m)$ provide a plethora of methods to represent field elements according to their particular application. Specifically, the two most classical schemes reported in literature are *Canonical Basis* (also called Standard Basis) and *Optimal Normal Basis*, though other strategies have recently been proposed ([2]). The former represents the generic field element $a \in GF(2^m)$ through the m-bit vector $(a_0, a_1, \ldots, a_{m-1})$ with respect to the set $(1, \alpha, \alpha^2, \ldots, \alpha^{m-1})$, where α is the root of an irreducible polynomial of degree m over $GF(2)$, which corresponds to the expansion $a(\alpha) = \sum_{i=0}^{m-1} a_i \alpha^i$. On the contrary, the latter specializes the set as $(\gamma, \gamma^2, \ldots, \gamma^{2^{m-1}})$, where γ is now the root of an *N-polynomial* of degree m over $GF(2)$. In this case the expansion is therefore given by $a(\gamma) = \sum_{i=0}^{m-1} a_i \gamma^{2^i}$ (for more information see [9]).

In order to reduce the complexity of the field multiplication special classes of irreducible polynomials have been suggested ([7], [10]). Among them, the AOP generators have been shown to be particularly interesting. An AOP is a polynomial characterized by the form $p(x) = 1 + x + x^2 + \ldots + x^m$, which is irreducible if and only if $m + 1$ is prime and 2 is primitive modulo $m + 1$ ([9]). For instance, for $m \leq 100$ there are thirteen useful values: 2, 4, 10, 12, 18, 28, 36, 52, 58, 60, 66, 82, and 100. Moreover, each *N-polynomial* generating a Type I ONB is also an AOP ([9]). For this reason in the following we will focus on AOPs, discussing the advantages of this class in the context of both Canonical Basis and Type I ONB representations.

3 Canonical Basis

Let $p(x) = 1 + x + x^2 + \ldots + x^m$ an irreducible polynomial over $GF(2^m)$, and let $a(\alpha)$ and $b(\alpha)$ be two elements of $GF(2^m)$, represented through the m-tuples $(a_0, a_1, \ldots, a_{m-1})$ and $(b_0, b_1, \ldots, b_{m-1})$, with respect to the root α of $p(x)$. Our goal is the computation of the field element $(c_0, c_1, \ldots, c_{m-1})$ given by the product $c(\alpha) = a(\alpha) \cdot b(\alpha) \in GF(2^m)$. This product can be computed in two different phases:

1. computation of the ordinary product of two polynomials $\ell(x) = a(x) \cdot b(x)$ over $GF(2)$
2. computation of the field product $c(x) \in GF(2^m)$ as $c(x) = \ell(x) \quad mod \quad p(x)$

3.1 Multiplication of Polynomials over GF(2)

First, we observe that the degrees of the polynomials $a(x)$ and $b(x)$ are both $\leq m-1$, therefore the degree of the polynomial $\ell(x)$ will be, in turn, $\leq 2m-2$. Formally we have:

$$\ell(x) = a(x) \cdot b(x) = \ell_0 + \ell_1 x + \ell_2 x^2 + \ldots + \ell_{2m-2} x^{2m-2} \tag{1}$$

This polynomial can be computed by means of a divide-and-conquer approach originally proposed to increase the speed of integer multiplications ([11]). Actually this strategy, which we will slightly improve and extend respect to the results obtained in [8], in turn reminiscent of the Karatsuba-Ofman algorithm, has been also successfully applied in case of trinomial generators ([12]).

More precisely, let us to observe that in this context m is surely even, thanks to the sufficient conditions that make $p(x)$ irreducible. Therefore we can assume $m = 2N$. As a consequence the polynomials $a(x)$ and $b(x)$ can be rewritten as $a(x) = A(x) + x^N B(x)$ and $b(x) = C(x) + x^N D(x)$ respectively, where

$$\begin{aligned} A(x) &= a_0 + a_1 x + a_2 x^2 + \ldots + a_{N-1} x^{N-1} \\ B(x) &= a_N + a_{N+1} x + a_{N+2} x^2 + \ldots + a_{2N-1} x^{N-1} \end{aligned}$$

and analogously

$$\begin{aligned} C(x) &= b_0 + b_1 x + b_2 x^2 + \ldots + b_{N-1} x^{N-1} \\ D(x) &= b_N + b_{N+1} x + b_{N+2} x^2 + \ldots + b_{2N-1} x^{N-1} \end{aligned}$$

Therefore, the product $\ell(x)$ can be computed as

$$\ell(x) = a(x) \cdot b(x) = A(x)C(x) + x^N [B(x)C(x) + A(x)D(x)] + x^{2N} B(x)D(x) \tag{2}$$

which, introducing the following auxiliary polynomials

$$\begin{aligned} P_{AC}(x) &= A(x) \cdot C(x) \\ P_{BD}(x) &= B(x) \cdot D(x) \\ P_{A+B}(x) &= A(x) + B(x) \\ P_{C+D}(x) &= C(x) + D(x) \end{aligned}$$

we can also express as

$$\ell(x) = P_{AC}(x) + x^N [P_{A+B}(x) \cdot P_{C+D}(x) + P_{AC}(x) + P_{BD}(x)] + x^{2N} P_{BD}(x) \tag{3}$$

Eq.(3) compute the product $a(x) \cdot b(x)$ by means of three multiplications of polynomials of degree $N-1$, together with shifts and "lettings-down" of α powers. Specifically, the architectural structure of the multiplier can be organized as follows:

– two circuits, composed of N XOR gates each, for the parallel computation of $A(x) + B(x)$ and $C(x) + D(x)$

– three circuits, composed by N^2 AND and $(N-1)^2$ XOR gates each, for the parallel computation of $A(x)\cdot C(x)$, $B(x)\cdot D(x)$, and $[A(x)+B(x)]\cdot[C(x)+D(x)]$; the XOR tree depth is $\lceil \log_2(N-1)\rceil$, provided that the polynomials involved have at most degree N
– one circuit, composed of $2N-1$ XOR gates, for the computation of $A(x)C(x) + B(x)D(x)$
– one circuit, composed of $2N-1$ XOR gates, for the computation of $[A(x)C(x) + B(x)D(x)] + [A(x)+B(x)][C(x)+D(x)]$
– one circuit, composed of $2N-2$ XOR gates, for the computation of $\ell(x)$ by means of the eq.(3), where each term, at this point, has been already pre-computed

As far as the time complexity is concerned, it should be noted that the overall circuit is able to produce the output $\ell(x)$ according to a time delay of $T_A + T_X(\lceil \log_2(N-1)\rceil + 3)$. In fact, after a period of time equal to T_X, the intermediate values $A(x)+B(x)$ and $C(x)+D(x)$ will be available; therefore, when other $T_A + T_X(\lceil \log_2(N-1)\rceil + 1)$ seconds have elapsed, the circuit will have also computed $[A(x)+B(x)]\cdot[C(x)+D(x)]$, $B(x)D(x)$, $A(x)C(x)$ and $A(x)C(x)+B(x)D(x)$, while waiting for other T_X seconds, also the computation of the term $A(x)C(x)+B(x)D(x)+[A(x)+B(x)]\cdot[C(x)+D(x)]$ will be completed. Therefore the result $\ell(x)$, which now needs other T_X seconds to be reached, just requires a time complexity equal to $T_A + T_X(\lceil \log_2(N-1)\rceil + 3)$.

The overall characteristics of the algorithm, whose details have been presented in Table 1, are respectively:

$$\begin{array}{lll} \Sigma'_A &= 3N^2 &= \frac{3}{4}m^2 \\ \Sigma'_X &= 3N^2+2N-1 &= \frac{3}{4}m^2+m-1 \\ \Theta' &= T_A+T_X(\lceil \log_2(N-1)\rceil+3) &= T_A+T_X(\lceil \log_2(m-2)\rceil+2) \end{array} \tag{4}$$

which can be compared with those provided by a direct parallel multiplication

$$\begin{array}{ll} \Sigma'_A &= m^2 \\ \Sigma'_X &= m^2-2m+1 \\ \Theta' &= T_A+T_X(\lceil \log_2(m-1)\rceil) \end{array} \tag{5}$$

It is evident how the former strategy exchanges a part of its time complexity in order to gain a $\frac{3}{4}$ factor in the corresponding number of gates.

Anyway, the values in (4) can be also further manipulated and expressed as (see also Table 1)

$$(\Sigma'_A)_m = 3(\Sigma'_A)_{m/2} \tag{6}$$

$$(\Sigma'_X)_m = 3(\Sigma'_X)_{m/2} + 4m - 4 \tag{7}$$

$$(\Theta')_m = (\Theta')_{m/2} + 3T_X \tag{8}$$

where $(C)_d$ represents the complexity C of the multiplier, i.e. Σ'_A, Σ'_X and Θ', when the polynomials in input have degree at most $d-1$, that is d coefficients.

Eq.(6), (7), and (8) show that the product of two polynomials of degree $\leq m-1$ can be performed by means of three multiplications of two polynomials

Table 1. Time and space complexity to multiply polynomials over GF(2).

Operation	Σ_A	Σ_X	Θ	Register Size
$A(x)+B(x)$		N	T_X	N
$C(x)+D(x)$		N		N
$A(x)C(x)$	N^2	$(N-1)^2$	$T_A+T_X\lceil\log_2(N-1)\rceil$	$2N-1$
$B(x)D(x)$	N^2	$(N-1)^2$		$2N-1$
$(A(x)+B(x))(C(x)+D(x))$	N^2	$(N-1)^2$		$2N-1$
$A(x)C(x)+B(x)D(x)$		$2N-1$		$2N-1$
$A(x)C(x)+B(x)D(x)+$		$2N-1$	T_X	$2N-1$
$+(A(x)+B(x))(C(x)+D(x))$				$2N-1$
$\ell(x)$		$2N-2$	T_X	$4N-1$

of degree equal (at most) to about the half the original ones, plus a little overhead needed to combine the partial results and to obtain the final output. Moreover these three multiplications can be computed in a parallel way, and this is the reason why within the time complexity (8) does not appear the factor 3, present, in contrast, in (6) and (7). It should be also pointed out that this additional overhead is relatively small, being limited to $4m-4$ XOR gates in (7) and characterized by an additional time delay equal to $3T_X$ in (8).

Moreover, provided that also $m/2$ is even, this strategy can be further applied, in order to gain a further reduction in the gate count. For instance, assuming that m is a power of 2, after k iterations we will obtain:

$$\begin{aligned} (\Sigma'_A)_m &= 3^k(\Sigma'_A)_{m/2^k} \\ (\Sigma'_X)_m &= 3^k(\Sigma'_X)_{m/2^k} + 8m[(3/2)^k-1] - 2(3^k-1) \\ (\Theta')_m &= (\Theta')_{m/2^k} + 3kT_X \end{aligned} \tag{9}$$

These results show a clear trade off between time and space complexity. Therefore, to significantly reduce the number of gates we have to increase the corresponding number of iterations, although, as a side-effect, the time delay of the multiplier will also rise, just linearly in the same number of iterations. Of course, an interesting question is: how much can we iterate the algorithm, provided that we want to reduce the space complexity as much as possible? It is easy to see that the optimal stop condition for this recursion is $m/2^k = 4$, a value for which a parallel and direct multiplication is more advantageous over the recursive scheme. In fact, iterating the algorithm we obtain $(\Sigma'_A)_4 = 12$ and $(\Sigma'_X)_4 = 15$, from which $(\Sigma'_{TOT}) = (\Sigma'_A)_4 + (\Sigma'_X)_4 = 27$, while $(\Theta')_4 = T_A + 4T_X$. On the contrary, using a direct strategy we have $(\Sigma'_A)_4 = 16$ and $(\Sigma'_X)_4 = 9$, from which $(\Sigma'_{TOT}) = (\Sigma'_A)_4 + (\Sigma'_X)_4 = 25$, while $(\Theta')_4 = T_A + 2T_X$. Therefore, taking into account this stop condition, the corresponding complexities, in case of $m = 2^t$, will be:

$$\begin{aligned} (\Sigma'_A)_m &= 16 \cdot 3^{\log_2 m - 2} \\ (\Sigma'_X)_m &= 8m \cdot [(3/2)^{\log_2 m - 2} - 1] + 7 \cdot 3^{\log_2 m - 2} + 2 \\ (\Theta')_m &= T_A + T_X(3\log_2 m - 4) \end{aligned} \tag{10}$$

Table 2. Comparing different polynomial multipliers over GF(2).

Scheme	Σ_A	Σ_X	Σ_{TOT}	Θ
Direct	m^2	$m^2 - 2m + 1$	$2m^2 - 2m + 1$	$T_A + T_X \lceil \log_2(m-1) \rceil$
$m = 256$	65,536	65,025	130,561	$T_A + 8T_X$
$m = 1024$	1,048,576	1,046,529	2,095,105	$T_A + 10T_X$
$m = 2048$	4,194,304	4,190,209	8,384,513	$T_A + 11T_X$
Paar ([8])	$m^{\log_2 3}$	$6m^{\log_2 3} - 8m + 2$	$7m^{\log_2 3} - 8m + 2$	$T_A + 3T_X \log_2 m$
$m = 256$	6,561	37,320	43,881	$T_A + 24T_X$
$m = 1024$	59,049	346,104	405,153	$T_A + 30T_X$
$m = 2048$	177,147	1,046,500	1,223,647	$T_A + 33T_X$
Proposed	$16 \cdot 3^{\log_2 m-2}$	$8m[(\frac{3}{2})^{\log_2 m-2} - 1]$ $+7 \cdot 3^{\log_2 m-2} + 2$	$8m[(\frac{3}{2})^{\log_2 m-2} - 1]$ $+23 \cdot 3^{\log_2 m-2} + 2$	$T_A + T_X(3\log_2 m - 4)$
$m = 256$	11,664	26,385	38,049	$T_A + 20T_X$
$m = 1024$	104,976	247,689	352,665	$T_A + 26T_X$
$m = 2048$	314,928	751,255	1,066,183	$T_A + 29T_X$

which slightly improves the results reached in [8]. For a quantitative comparison see also Table 2, where it should be clear how our scheme pays a greater number of AND gates, if compared with [8], but in order to reduce both the overall number of gates Σ_{TOT} and the time complexity Θ.

Now we have to generalize the previous results, in order to make the scheme suitable for generating AOPs. In fact, it is possible to employ the same strategy also when m is not a power of 2. To make the design very modular, we do not optimize the structure of the multiplier distinguishing the two cases, m even and odd, as done in [12]. In contrast, we simply expand the circuit registers to handle, at each step, polynomials of odd degree, that is with an even number of coefficients. As a consequence the following generalization can be derived and used to multiply polynomials of any degree ($m \geq 4$):

$$\begin{aligned} (\Sigma'_A)_m &= 16 \cdot 3^{\lceil \log_2 m \rceil - 2} \\ (\Sigma'_X)_m &= 4 \cdot \sum_{i=0}^{\lceil \log_2 m \rceil - 3} 3^i \lceil \tfrac{m}{2^i} \rceil + 7 \cdot 3^{\lceil \log_2 m \rceil - 2} + 2 \\ (\Theta')_m &= T_A + T_X(3\lceil \log_2 m \rceil - 4) \end{aligned} \tag{11}$$

At the end of this first phase, the circuit outputs the coefficients of the product polynomial $\ell(x)$, that is the bit vector $(\ell_0, \ell_1, \ldots, \ell_{2m-2})$. The subsequent step will be the computation of field element $c(x)$ as the remainder $c(x) = \ell(x) \quad mod \quad p(x)$.

3.2 Reduction Phase

Let $\ell(x) = (\ell_0, \ell_1, \ldots, \ell_{2m-2})$ the polynomial given by the ordinary product of $a(x)$ and $b(x)$. The current phase prescribes the computation of field element $c(x) = a(x) \cdot b(x) \in GF(2^m)$ as the remainder of the polynomial division of $\ell(x)$ by the generator polynomial $p(x)$. To speed up this computation it is possible to take advantage of the structure of the generator $p(x)$. Thanks to the regular form of this polynomial it is easy to express the coefficients of the field element c_i in terms of coefficients ℓ_i. Specifically, it can be shown that the field element $c(x) \equiv (c_0, c_1, \ldots, c_{m-1})$ can be computed as

$$c_i = \begin{cases} \ell_i + \ell_m + \ell_{m+i+1} & \text{if } i = 0, 1, \ldots, m-3 \\ \ell_{m-2} + \ell_m & \text{if } i = m-2 \\ \ell_{m-1} + \ell_m & \text{if } i = m-1 \end{cases}$$

Of course this step can be accomplished according to a time complexity equal to $(\Theta'')_m = 2T_X$, while the relating space complexity is given by $(\Sigma''_X)_m = 2m - 2$.

As a consequence, the characteristics of the overall multiplier taking in input the two bit vectors $(a_0, a_1, \ldots, a_{m-1})$ and $(b_0, b_1, \ldots, b_{m-1})$ and producing, at the output of the circuit, the product element $(c_0, c_1, \ldots, c_{m-1})$, will be given by:

$$\begin{aligned} (\Sigma_A)_m &= (\Sigma'_A)_m + (\Sigma''_A)_m &&= 16 \cdot 3^{\lceil \log_2 m \rceil - 2} \\ (\Sigma_X)_m &= (\Sigma'_X)_m + (\Sigma''_X)_m &&= 4 \cdot \sum_{i=0}^{\lceil \log_2 m \rceil - 3} 3^i \lceil \tfrac{m}{2^i} \rceil + 7 \cdot 3^{\lceil \log_2 m \rceil - 2} + 2m \\ (\Theta)_m &= (\Theta')_m + (\Theta'')_m &&= T_A + T_X(3\lceil \log_2 m \rceil - 2) \end{aligned} \tag{12}$$

It should be noted that the final space complexities are notably lower than those currently available in literature and belonging to the same class ([3,6,7]). For a direct comparison see also Table 3, where it is evident how our scheme does exchange time complexity in order to gain a more consistent reduction in both the number of AND and XOR gates. Moreover, this gain grows as m grows. For instance, if $m = 226$, our multiplier provides a factor reduction, in the overall gate count, equal to 2.7, with respect to the best method ([3]), paying a corresponding time expansion factor of 2. On the other hand, in case of $m = 2026$, the area reduction factor becomes 7.7, while the corresponding time expansion rises only up to 2.36.

As an example, in Figure 1 is reported the scheme of the overall multiplier, when the generating polynomial is $p(x) = 1 + x + x^2 + x^3 + x^4 + x^5 + x^6 + x^7 + x^8 + x^9 + x^{10}$. In this case the two inputs $a(x)$ and $b(x)$ have been rewritten as $a(x) = A(x) + x^5 B(x)$ and $b(x) = C(x) + x^5 D(x)$ respectively, where

$$\begin{aligned} A(x) &= a_0 + a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4 \\ B(x) &= a_5 + a_6 x + a_7 x^2 + a_8 x^3 + a_9 x^4 \end{aligned}$$

and analogously

$$\begin{aligned} C(x) &= b_0 + b_1 x + b_2 x^2 + b_3 x^3 + b_4 x^4 \\ D(x) &= b_5 + b_6 x + b_7 x^2 + b_8 x^3 + b_9 x^4 \end{aligned}$$

Therefore, the field element $c(x) = a(x) \cdot b(x) \in GF(2^{10})$ can be computed by means of three multiplication circuits for polynomials of degree 4, to obtain $(A + B) \cdot (C + D)$, $A \cdot C$ and $B \cdot D$, plus some XOR gates, needed to recombine partial results (block Recombination), and to perform the reduction phase (block

Table 3. Comparing different Canonical Basis multipliers with generating AOPs.

Scheme	Σ_A	Σ_X	Θ
Itoh-Tsujii [7]	m^2+2m+1	m^2+2m	$T_A+T_X\lceil\log_2 m+\log_2(m+2)\rceil$
$m=226$	51,529	51,528	T_A+16T_X
$m=1018$	1,038,361	1,038,360	T_A+20T_X
$m=2026$	4,108,729	4,108,728	T_A+22T_X
Hasan et al. [5]	m^2	m^2+m-2	$T_A+T_X(\lceil\log_2(m-1)\rceil+m)$
$m=226$	51,076	51,300	T_A+234T_X
$m=1018$	1,036,324	1,037,340	$T_A+1028T_X$
$m=2026$	4,104,676	4,106,700	$T_A+2037T_X$
Koç-Sunar [3]	m^2	m^2-1	$T_A+T_X(\lceil\log_2(m-1)\rceil+2)$
$m=226$	51,076	51,075	T_A+10T_X
$m=1018$	1,036,324	1,036,323	T_A+12T_X
$m=2026$	4,104,676	4,104,675	T_A+13T_X
Proposed	$16\cdot 3^{\lceil\log_2 m\rceil-2}$	$4\cdot\sum_{i=0}^{\lceil\log_2 m\rceil-3} 3^i\lceil\frac{m}{2^i}\rceil$ $+7\cdot 3^{\lceil\log_2 m\rceil-2}+2m$	$T_A+T_X(3\lceil\log_2 m\rceil-2)$
$m=226$	11,664	25,635	T_A+22T_X
$m=1018$	104,976	249,627	T_A+28T_X
$m=2026$	314,928	754,365	T_A+31T_X

Reduction Phase). To make fully modular the circuit design (which could be an advantage, especially if $m \gg 10$), we do not directly deal with these polynomials of degree 4. Instead we extend these polynomials by a single bit, in order to obtain polynomials of degree 5. This provides us with the possibility to further iterate the algorithm and to directly employ modules architecturally equivalent to the previous ones. In fact, each of these three products can be computed, in turn, by means of other three multiplication circuits for polynomials of degree 2, for the parallel computation of $(A'+B')\cdot(C'+D')$, $A'\cdot C'$ and $B'\cdot D'$, plus the XOR gates needed for the recombination. Conversely, the latter 9 polynomial multiplications are not further iterated, because of the lower time and space complexities provided by a direct multiplication.

4 Type I Optimal Normal Basis

The previous scheme can be also adopted in case of Type I ONB, following the smart strategy proposed in [3]. Specifically, let $p(x)=1+x+x^2+\ldots+x^m$ an *N-polynomial* over $GF(2^m)$, and let $a(\gamma)$ and $b(\gamma)$ be two elements of $GF(2^m)$, represented through the m-bit vectors $(a_0,a_1,\ldots,a_{m-1})$ and $(b_0,b_1,\ldots,b_{m-1})$, with respect to the root γ of $p(x)$. Given that $p(x)$ is also an AOP, the root γ satisfies the property $\gamma^{m+1}=1$, in fact

$$p(x)=1+x+x^2+\ldots+x^m=\frac{x^{m+1}+1}{x+1} \tag{13}$$

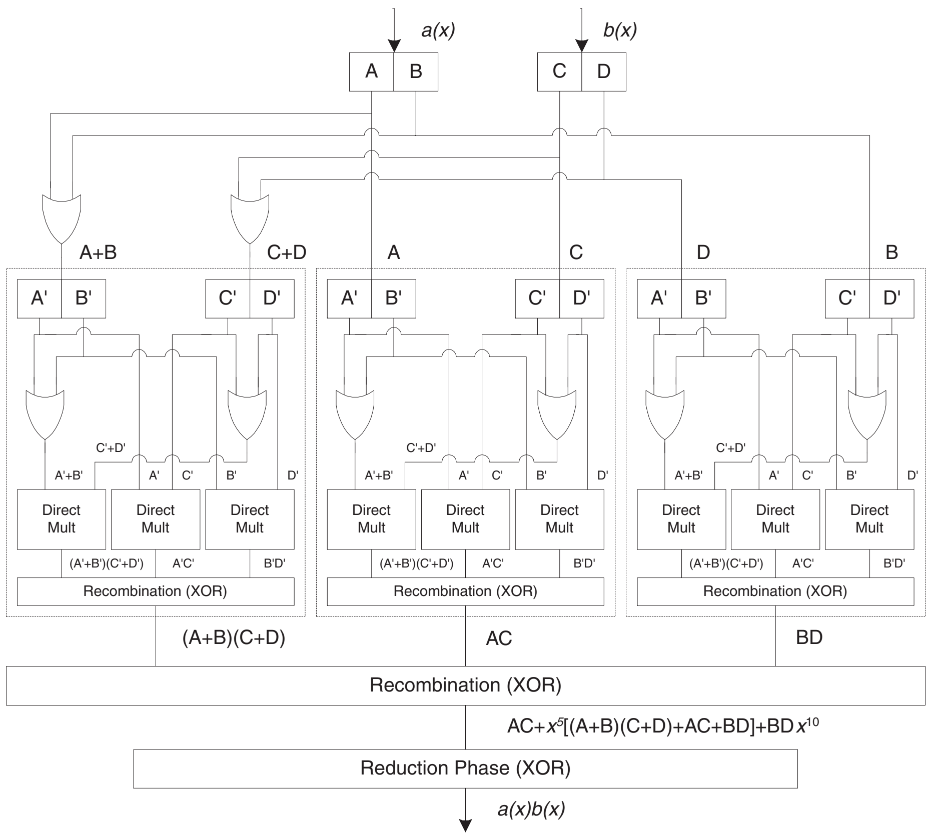

Fig. 1. Multiplier for Canonical Basis over $GF(2^{10})$ with generating polynomial AOP.

As a consequence, the set

$$\Psi = (\gamma, \gamma^2, \ldots, \gamma^m) \tag{14}$$

can also be used as a basis for $GF(2^m)$. More precisely, (14) is nothing but a shifted version of the Canonical Basis, therefore the elements of $GF(2^m)$ represented in Type I ONB can be quickly converted in Canonical Basis, and vice-versa, by means of a simple permutation of the components. In fact, thanks to the relation $\gamma^{m+1} = 1$, we can write the conversion

$$a(\gamma) = \Sigma_{i=0}^{m-1} a_i \gamma^{2^i} = \Sigma_{i=1}^{m} a'_i \gamma^i \tag{15}$$

by means of the permutation P defined as

$$a'_{2^i \ mod\ (m+1)} = a_i \qquad \text{for } i = 0, 1, \ldots, m-1 \tag{16}$$

Therefore, the elements to be multiplied in Type I ONB will be simply converted in Canonical Basis, through the permutation P, before entering the multiplier. The output of the circuit, computed according to the complexities given

Table 4. Comparing different Type I ONB multipliers.

Scheme	Σ_A	Σ_X	Θ
Massey-Omura [4]	m^2	$2m^2 - 2m$	$T_A + T_X(\lceil \log_2(m-1) \rceil + 1)$
$m = 226$	51,076	101,700	$T_A + 9T_X$
$m = 1018$	1,036,324	2,070,612	$T_A + 11T_X$
$m = 2026$	4,104,676	8,205,300	$T_A + 12T_X$
Hasan et al. [6]	m^2	$m^2 - 1$	$T_A + T_X(\lceil \log_2(m-1) \rceil + 1)$
$m = 226$	51,076	51,075	$T_A + 9T_X$
$m = 1018$	1,036,324	1,036,323	$T_A + 11T_X$
$m = 2026$	4,104,676	4,104,675	$T_A + 12T_X$
Koç-Sunar [3]	m^2	$m^2 - 1$	$T_A + T_X(\lceil \log_2(m-1) \rceil + 2)$
$m = 226$	51,076	51,075	$T_A + 10T_X$
$m = 1018$	1,036,324	1,036,323	$T_A + 12T_X$
$m = 2026$	4,104,676	4,104,675	$T_A + 13T_X$
Proposed	$16 \cdot 3^{\lceil \log_2 m \rceil - 2}$	$4 \cdot \sum_{i=0}^{\lceil \log_2 m \rceil - 3} 3^i \lceil \frac{m}{2^i} \rceil$ $+7 \cdot 3^{\lceil \log_2 m \rceil - 2} + 2m$	$T_A + T_X(3\lceil \log_2 m \rceil - 2)$
$m = 226$	11,664	25,635	$T_A + 22T_X$
$m = 1018$	104,976	249,627	$T_A + 28T_X$
$m = 2026$	314,928	754,365	$T_A + 31T_X$

in (12) and still represented in Canonical Basis, will be restored in Normal Basis thanks to the inverse permutation P^{-1}. It should be noted that these two additional permutations do not increase the overall time and space complexity of the multiplier. In fact, P, and its inverse P^{-1}, can be directly implemented by wiring the fan-in and fan-out of the circuit, without modifying any complexity. Therefore, our scheme is able to maintain the previously discussed gate count reduction also in case of Type I ONB. This reduction is significant, especially if compared with the one provided by the other fast parallel schemes currently available in literature ([4,6,3]), as reported in Table 4. Finally, also in this case the gain factor becomes more consistent as soon as m grows, as previously seen for Canonical Basis.

5 Conclusions

In this paper we have proposed a new low space complexity scheme for fast parallel multiplication of field elements represented through both Canonical and Type I Optimal Normal Bases. Specifically, the discussed strategy shows how to avoid quadratic space complexity, paying only a limited increase in the corresponding time delay. As reported in Table 3 and 4, the proposed scheme offers a circuit complexity significantly lower compared to the other fast parallel schemes present in the open literature ([3,4,5,6,7]). This characteristic makes the employment of this multiplier particularly suitable for applications characterized by specific space constraints, such as those based on smart cards, token hardware, mobile phones or other portable devices.

References

1. Aho A.V., Hopcroft J.E., Ullman J.D., "The Design and Analysis of Computer Algorithms", Addison-Wesley, Reading, Mass., 1975.
2. Drolet G., "A New Representation of Elements of Finite Fields $GF(2^m)$ Yelding Small Complexity Arithmetic Circuit", IEEE Trans. on Computers, vol.47, pp.938-946, 1998.
3. Koç C.K., Sunar B., "Low Complexity Bit-Parallel Canonical and Normal Basis Mutipliers for a Class of Finite Fields", IEEE Trans. on Computers, vol.47, pp.353-356, March 1998.
4. Omura J., Massey J., "Computational method and apparatus for finite field arithmetic", U.S. Patent Number 4,587,627, May 1986.
5. Hasan M.H., Wang M.Z., Bhargava V.K., "Modular construction of low complexity parallel multipliers for a class of finite fields $GF(2^m)$", IEEE Trans. on Computers, vol.41, no. 8, pp.962-971, August 1992.
6. Hasan M.H., Wang M.Z., Bhargava V.K., "A modified Massey-Omura Multiplier for a class of finite fields", IEEE Trans. on Computers, vol.42, no. 10, pp.1278-1280, October 1993.
7. Itho T., Tsujii S., "Structure of parallel multipliers for a class of Finite Fields $GF(2^m)$", Information and Computation, vol.83, pp.21-40, 1989.
8. Paar C., "A new architecture for a parallel finite field multiplier with low complexity based on composite fields", IEEE Trans. on Computers, vol.45, no. 7, pp.846-861, July 1996.
9. Menezes A.J., Blake I., Gao X., Mullin R. Vanstone S. and Yaghoobian T., "Applications of Finite Fields", Boston, MA: Kluwer Academic Publisher, 1993.
10. Mastrovito E.D., "VLSI Architectures for multiplication over finite field $GF(2^m)$", In T. Mora, editor, Applied Algebra Algebraic Algorithms, and Error-Correcting Codes, 6-th International Conference, AAECC-6, pp. 297-309, Roma, Italy, July 1988. New York, NY: Springer-Verlag.
11. Knuth D.E., "The art of the computing programming", Vol.2: Seminumerical algorithms, Adison-Wilsey, Reading, MA., 1969.
12. Elia M., Leone M. and Visentin C., "Low Complexity bit-parallel multipliers for $GF(2^m)$ with generator polynomial $x^m + x^k + 1$", Electronics Letters, Vol.35, No.7, April 1999.