A New Architecture for a Parallel Finite Field Multiplier with Low Complexity Based on Composite Fields *

Christof Paar †

IEEE Transactions on Computers, July 1996, vol 45, no 7, pp 856-861

Abstract

In this paper a new bit-parallel structure for a multiplier with low complexity in Galois fields is introduced. The multiplier operates over *composite fields* $GF((2^n)^m)$, with $k = nm$. The Karatsuba-Ofman algorithm is investigated and applied to the multiplication of polynomials over $GF(2^n)$. It is shown that this operation has a complexity of order $\mathcal{O}(k^{\log_2 3})$ under certain constraints regarding k. A complete set of primitive field polynomials for composite fields is provided which perform modulo reduction with low complexity. As a result, multipliers for fields $GF(2^k)$ up to $k = 32$ with low gate counts and low delays are listed. The architectures are highly modular and thus well suited for VLSI implementation.

*This paper was presented in part at the Swedish-Russian Workshop on Information Theory, August 22-27, 1993, Mölle, Sweden

†The author is with the Electrical and Computer Engineering Department, Worcester Polytechnic Institute, Worcester, MA 01609. E-mail: christof@ece.wpi.edu

1 Introduction

Finite fields play an increasingly important role in modern digital communication systems. Typical areas of applications are cryptographic schemes [1] or error correction codes such as Reed-Solomon codes [2]. For efficient VLSI implementation of such systems efficient hardware structures for the two fundamental field operations, addition and multiplication, must be provided. Whereas addition can be implemented with a very low space and time complexity for fields in standard representation, fast multipliers usually possess a much higher complexity. During the last decade various bit-parallel multipliers over Galois fields $GF(2^k)$ have been developed, see e.g. [3] [4] [5] [6]. All these multipliers show a space complexity — measured in the number of two input mod-2 adders (logical XOR) and number of two input mod-2 multipliers (logical AND) — of $\mathcal{O}(k^2)$. This paper presents a new architecture of a bit parallel, i.e. fast, multiplier for extension fields of $GF(2)$ with a significantly improved space complexity. The application of (multiple) field extensions to multipliers has been proposed before in [7], [8] by Afanasyev and in [9] by Pincin. The results there show also a low space complexity although the internal structure of the multiplier is different.

We consider finite fields $GF(2^n)$ with $n > 1$. The elements of an extension field $GF((2^n)^m)$ may be represented in the standard (or canonical) base as polynomials with a maximum degree of $m-1$ over $GF(2^n)$: $A(x) = a_{m-1}x^{m-1} + \cdots + a_0$, where $a_i \in GF(2^n)$ and $A = A(x) \bmod P(x) \in GF((2^n)^m)$. The field polynomial of the extension field is an irreducible (or even primitive) polynomial $P(x)$ of degree m over $GF(2^n)$. Fields of the form $GF((2^n)^m)$ are sometimes referred to as *composite fields* [10] and have further applications, for instance in the generation of m-sequences [11]. Composite fields $GF((2^n)^m)$ are isomorphic to fields $GF(2^k)$ iff $k = nm$.

Multiplication of two elements A and B of a composite field can be performed in the standard representation as:

$$A(x) \times B(x) \bmod P(x). \tag{1}$$

The field multiplication in (1) may be performed in two steps:

1. Ordinary polynomial multiplication ($\times$);
2. Reduction modulo the field polynomial (mod).

We will treat both steps separately. The basic arithmetic operations, addition and multiplication, which are required for both steps are performed in the ground field $GF(2^n)$.

The key idea of the multiplier introduced here is the application of the Karatsuba-Ofman algorithm (KOA) [12] [13] for efficient polynomial multiplication to Step 1. *Efficient* refers to the fact that the algorithm saves multiplications at the cost of extra additions. Hence, if the algorithm is expected to improve the complexity, multiplication in the field must be more "costly" than addition. This condition is naturally fulfilled for polynomials over fields $GF(2^n)$: Addition can be realized with n XOR gates, multiplication requires n^2 AND gates and at least $n^2 - 1$ XOR gates using traditional approaches.

In the sequel the following notations will be used for the field polynomials: $Q(y) = y^n + q_{n-1}y^{n-1} + \cdots + q_0$, with $q_i \in GF(2)$ denotes an irreducible polynomial of the ground field $GF(2^n)$, and $P(x) = x^m + p_{n-1}x^{n-1} + \cdots + p_0$, with $p_i \in GF(2^n)$ denotes an irreducible polynomial of the composite field $GF((2^n)^m)$. All irreducible polynomials $Q(y)$ and $P(x)$ used in this paper are (monic) primitive polynomials. By $(1, \omega, \omega^2, \ldots, \omega^{n-1})$ we denote a base for $GF(2^n)$ over $GF(2)$, where $Q(\omega) = 0$.

2 Preliminaries

2.1 General Multiplication in $GF(2^n)$

Review of the Mastrovito Multiplier

The multiplier proposed in this paper uses the architecture of Mastrovito [5], [14] to perform multiplication in the ground field $GF(2^n)$. First, we will introduce a matrix notation for the multiplication $A(y)B(y) = C(y) \bmod Q(y)$ in the field $GF(2^n)$. All elements are binary polynomials of degree less than n:

$$c_{n-1}y^{n-1} + \ldots + c_0 = (a_{n-1}y^{n-1} + \ldots + a_0)(b_{n-1}y^{n-1} + \ldots + b_0) \bmod Q(y).$$

Alternatively, the elements $B(y)$ and $C(y)$ can be represented as column vectors containing the polynomial coefficients. By introducing the matrix $\mathbf{Z} = f(A(y), Q(y))$ the multiplication can be described as:

$$C = \begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_{n-1} \end{pmatrix} = \mathbf{Z}B = \begin{pmatrix} f_{0,0} & \cdots & f_{0,n-1} \\ \vdots & \ddots & \vdots \\ f_{n-1,0} & \cdots & f_{n-1,n-1} \end{pmatrix} \begin{pmatrix} b_0 \\ b_1 \\ \vdots \\ b_{n-1} \end{pmatrix}. \tag{2}$$

The matrix $\mathbf{Z}$ is named "product matrix". Its coefficients $f_{ij} \in GF(2)$ depend recursively on the coefficients a_i and on the coefficients q_i of the $\mathbf{Q}$ matrix which will be introduced below in Equation (4):

$$f_{ij} = \begin{cases} a_i & ; \quad j = 0 \quad ; \quad i = 0, \ldots, n-1 \\ u(i-j)a_{i-j} + \sum_{t=0}^{j-1} q_{j-1-t,i}\, a_{n-1-t} & ; \quad j = 1, \ldots, n-1 \quad ; \quad i = 0, \ldots, n-1 \end{cases} \tag{3}$$

where the step function u is defined as

$$u(\mu) = \begin{cases} 1 & \mu \geq 0 \\ 0 & \mu < 0. \end{cases}$$

The matrix-vector product in Equation (2) describes the entire field multiplication. The $\mathbf{Q}$ matrix which is required to build $\mathbf{Z}$ is a function of the binary field polynomial $Q(y)$ of

degree n. Its binary entries $q_{i,j}$ are defined such that:

$$\begin{pmatrix} y^n \\ y^{n+1} \\ \vdots \\ y^{2n-2} \end{pmatrix} \equiv \begin{pmatrix} q_{0,0} & \cdots & q_{0,n-1} \\ \vdots & \ddots & \vdots \\ q_{n-2,0} & \cdots & q_{n-2,n-1} \end{pmatrix} \begin{pmatrix} 1 \\ y \\ \vdots \\ y^{n-1} \end{pmatrix} \bmod Q(y). \tag{4}$$

The $\mathbf{Q}$ matrix describes the representation of the polynomials $y^n, y^{n+1}, \ldots, y^{2n-2}$ in the equivalence classes $\bmod Q(y)$, i.e. after the reduction modulo $Q(y)$.

The implementational complexity of the matrix-vector product (3) depends solely on the primitive polynomial $Q(y)$. In [5] primitive polynomials are given for fields $GF(2^n)$, $n = 2, 3, \ldots, 16$. The polynomials are optimum with respect to the number of gates required to multiply in the field. For the fields in this range in which primitive trinomials of the form:

$$Q(y) = y^n + y + 1 \tag{5}$$

exist, the space complexity is given by: $\#\text{AND} + \#\text{XOR} = 2n^2 - 1$. Such trinomials were found for $n = 2, 3, 4, 6, 7, 9, 10, 11, 15$. For the case $n = 5$ the trinomial $Q(y) = y^5 + y^2 + 1$ exists. By exploiting the specific redundancies of the corresponding product matrix (2) the same complexity could be realized. For other values of n the complexity is higher as shown in [14, Table 4.5].

The delay (or time complexity) of the multiplier is upper bounded by: $T = T_{\text{AND}} + T_{\text{XOR}} \leq 1 + 2\lceil \log_2 n \rceil$, measured in gate delays.

Some Comments on the Mastrovito Multiplier

Next, we will state some additional facts about the Mastrovito multiplier. First we will give a formula for computing the matrix $\mathbf{Q}$. The binary entries $q_{i,j}$ of $\mathbf{Q}$ in Equation (4) can be computed recursively after the first row is filled with the coefficients of $Q(y) = y^n + q_{n-1}y^{n-1} + \ldots + q_1 y + 1$, i.e. $q_{0,j} = q_j$ where $q_0 = 1$, through:

$$q_{i,j} = \begin{cases} q_{i-1,n-1} & ; \quad i = 1, \ldots, n-2 \quad ; \quad j = 0; \\ q_{i-1,j-1} + q_{i-1,n-1}q_{0,j} & ; \quad i = 1, \ldots, n-2 \quad ; \quad j = 1, \ldots, n-1. \end{cases}$$

The recursion follows immediately from [6, Lemma 1] through the substitutions $f_j \to q_{0,j}$ and $p_j^{[i]} \to q_{i,j}$.

Since the matrix-vector operation in Equation (2) requires exactly $n^2 \bmod 2$ multiplications, the space complexity can be further specified as:

$$\#\text{AND} = n^2 \tag{6}$$

$$\#\text{XOR} = n^2 - 1 \tag{7}$$

where Equation (7) is only valid for fields with irreducible polynomials which possess property (5). The time complexity can be further specified into multiples of XOR and AND gate delays. The delays will be denoted as $\mathcal{T}_{\text{xor}}$ and $\mathcal{T}_{\text{and}}$, respectively. If it is taken into consideration that each path through the multiplier contains only one mod 2 multiplier, it follows directly that the overall delay can be upper bounded by:

$$T \leq \mathcal{T}_{\text{and}} + 2\mathcal{T}_{\text{xor}}\lceil \log_2 n \rceil. \tag{8}$$

2.2 Multiplication with a constant in $GF(2^n)$

In Section 4 it will be shown that for the achievement of a low complexity for the operation "mod $P(x)$"— which is the second step in the field multiplication (1) — it is crucial to have an efficient scheme for the multiplication of an arbitrary element with a constant in $GF(2^n)$. The results from Section 2.1 for multiplication of two arbitrary elements can be applied directly to constant multiplication as well. If Equation (3) is applied to a fixed element $A(y) = a_{n-1}y^{n-1} + \cdots + a_0$, it yields a *fixed* binary product matrix $\mathbf{Z}$.

Example: Let $Q(y) = y^7 + y + 1$ the primitive field polynomial of $GF(2^7)$. The primitive element of the field is denoted ω, where $Q(\omega) = 0$. The multiplication with the field element $A(y) = \omega^{47} = y^4 + y^3 + y^2 + y + 1$ is described by:

$$C = \omega^{47}B = \mathbf{Z}B = \begin{pmatrix} b_0 + b_3 + b_4 + b_5 + b_6 \\ b_0 + b_1 + b_3 \\ b_0 + b_1 + b_2 + b_4 \\ b_0 + b_1 + b_2 + b_3 + b_5 \\ b_0 + b_1 + b_2 + b_3 + b_4 + b_6 \\ b_1 + b_2 + b_3 + b_4 + b_5 \\ b_2 + b_3 + b_4 + b_5 + b_6 \end{pmatrix}. \tag{9}$$

Each operation "+" in (9) denotes a mod 2 addition.

Considering (9) it is obvious that constant multiplication in $GF(2^n)$ does not require any multiplication but only mod 2 additions. The *average* complexity for constant multiplication in $GF(2^n)$ is given by [14, Section 5.1.2]

$$\#\overline{\text{XOR}} = \frac{n^2}{2} - n. \tag{10}$$

However, to realize constant multiplication with *low complexity* it is necessary to solve the optimization problem on Boolean equations of form (9). The cost function of the optimization problem is the number of mod 2 additions required to realize a set of n equations in n variables b_i, where each equation is a sum over certain b_i. We applied a greedy algorithm to the problem which yields suboptimum solutions. In every step of the iterative algorithm the occurrence of all possible pairs $b_i + b_j$ is determined. The most often occurring pair

$b_k + b_l$ is precomputed. Thus, a locally optimum solution is found. The pair is considered a new element $b_\mu = b_k + b_l$. In the next iteration step again all possible pairs $b_i + b_j$ are investigated, including the new element b_μ. The algorithm eventually terminates when all possible pairs occur only once. E.g., application of the algorithm to Equation (9) results in optimized sums which can be realized with 14 XOR gates whereas a direct implementation requires 26 XOR gates.

3 Efficient Polynomial Multiplication

3.1 The Karatsuba-Ofman Algorithm

In this section an efficient scheme for multiplying two polynomials will be derived. This is the first and, with respect to the complexities, major step for performing the entire field multiplication (1). We apply a "divide-and-conquer" algorithm which was first described by Karatsuba and Ofman in 1962 in the "Doklady Akademii Nauk SSSR," the English translation of which followed in 1963 [12]. A more compact version is described in [13, Section 4.3.3]. A detailed description of the algorithm's computational complexity is given in [15, Section 3], where the KOA is referred to as "Split." However, this reference contains an error in the derivation of the additive complexity, leading to a somewhat incorrect complexity formula[1]. The KOA provides a recursive algorithm which reduces the multiplicative complexity of m^2 and — for large enough m — the additive complexity of $(m-1)^2$ required by the "school book method."

We consider the multiplication of two polynomials $A(x)$ and $B(x)$ with a maximum degree of $m-1$ over a field $\mathcal{F}$, i.e. each polynomial possesses at most m coefficients from $\mathcal{F}$. We are interested in finding the product $C'(x) = A(x)B(x)$ with $\deg(C'(x)) \leq 2m-2$. The consideration here is restricted to polynomials where m is a power of two: $m = 2^t$, t integer. To apply the algorithm, both polynomials are split into a lower and an upper half:

$$\begin{aligned} A &= x^{\frac{m}{2}}(x^{\frac{m}{2}-1}a_{m-1} + \cdots + a_{\frac{m}{2}}) + (x^{\frac{m}{2}-1}a_{\frac{m}{2}-1} + \cdots + a_0) = x^{\frac{m}{2}}A_h + A_l \\ B &= x^{\frac{m}{2}}(x^{\frac{m}{2}-1}b_{m-1} + \cdots + b_{\frac{m}{2}}) + (x^{\frac{m}{2}-1}b_{\frac{m}{2}-1} + \cdots + b_0) = x^{\frac{m}{2}}B_h + B_l. \end{aligned} \quad (11)$$

Using (11), a set of auxiliary polynomials $D(x)$ is defined:

$$\begin{aligned} D_0(x) &= A_l(x)B_l(x) \\ D_1(x) &= [A_l(x) + A_h(x)][B_l(x) + B_h(x)] \\ D_2(x) &= A_h(x)B_h(x). \end{aligned} \quad (12)$$

The product polynomial $C'(x) = A(x)B(x)$ is achieved by:

$$C'(x) = D_0(x) + x^{\frac{m}{2}}[D_1(x) - D_0(x) - D_2(x)] + x^m D_2(x). \quad (13)$$

[1] the error will be outlined later

Thus far the procedure has reduced the number of coefficient multiplications to $3/4m^2$. However, the algorithm can be applied recursively to the three polynomial multiplications in (12). The next iteration step splits the polynomials $A_l, A_h, \text{and}(A_l + A_h)$ and their B counterparts again in half. The algorithm eventually terminates after t steps. In the final step the polynomials $D_i^{(t)}(x)$ are degenerated into single coefficients, i.e. $\deg(D^{(t)}(x)) = 0$. Since every step exactly halves the number of coefficients, the algorithm terminates after $t = \log_2 m$ steps.

The following two theorems provide expressions for the computational and the time complexity of the KOA for polynomials over fields of characteristic 2 with respect to a parallel hardware implementation.

Theorem 1 *Consider two arbitrary polynomials in one variable of degree less or equal $m-1$, where m is a power of two, with coefficients in a field $\mathcal{F}$ of characteristic 2. By using the Karatsuba-Ofman algorithm the polynomials can be multiplied with:*

$$\#\otimes = m^{\log_2 3}, \tag{14}$$

$$\#\oplus \leq 6m^{\log_2 3} - 8m + 2, \tag{15}$$

multiplications and additions, respectively, in $\mathcal{F}$.

Theorem 2 *Consider two arbitrary polynomials in one variable of degree less or equal $m-1$, where m is a power of two, with coefficients in a field $\mathcal{F}$ of characteristic 2. A parallel realization of the Karatsuba-Ofman algorithm for the multiplication of the two polynomials can be implemented with a time complexity (or delay) of:*

$$T = T_\otimes + 3\,(\log_2 m)\,T_\oplus, \tag{16}$$

where "$T_\otimes$" and "$T_\oplus$"denote the delay of one multiplier and one adder, respectively, in $\mathcal{F}$.

It should be noted that the subtractions in (13) are additions if $\mathcal{F}$ has characteristic 2. For the proof of the theorems three stages of the algorithm will be distinguished:

Proof.

1. In the first stage the mere splitting of the polynomials is considered. Since splitting itself takes no computation, only the two summations in Equation (12) are of interest. Taking into account that the number of polynomials triples in each iteration step, whereas the length of the polynomials is reduced by half, one obtains:

   $$\#\oplus_1 = \sum_{i=1}^{\log_2 m} 3^{i-1} 2\frac{m}{2^i} = 2m^{\log_2 3} - 2m.$$

   Since all additions of one iteration can be performed in parallel in a hardware realization, the delay equals:

   $$T_1 = T_\oplus \log_2 m,$$

   where "$T_\oplus$" denotes the delay for one adder in $\mathcal{F}$.

2. In the second stage the achieved $3^{\log_2 m} = m^{\log_2 3}$ polynomials (each consisting of one coefficient) are actually multiplied. This requires:

$$\#\otimes_2 = m^{\log_2 3}$$

multiplications. The delay of a parallel implementation is:

$$T_2 = T_\otimes,$$

where "$T_\otimes$" denotes the delay caused by one multiplier in $\mathcal{F}$.

3. The third stage merges the polynomials according to Equation (13). There are two kinds of additions (or subtractions) involved: Subtracting three polynomials with $2^i - 1$ coefficients and $2^i - 2$ additions due to the overlapping[2] of three terms:

$$\#\oplus_3 = \sum_{i=1}^{\log_2 m} 3^{\log_2 m - i}[2(2^i - 1) + (2^i - 2)] = 4m^{\log_2 3} - 6m + 2.$$

The delay equals:

$$T_3 = 2\,(\log_2 m)\,T_\oplus.$$

The overall complexities in the Theorems 1 and 2 are now obtained by summation of the partial complexities. □

However, the right hand side of the additive complexity (15) is an upper bound because the recursive algorithm bears redundancies which can be eliminated in a parallel realization. For instance, for the value $m = 4$ the upper bound in (15) can be reduced from 24 to 22.

3.2 Karatsuba-Ofman Algorithm for Polynomials over $GF(2^n)$

If the Karatsuba-Ofman algorithm is applied to multiplication in composite fields, the polynomials $A(x), B(x)$ are elements of the field $GF((2^n)^m)$. The operations described above refer to arithmetic with the coefficients a_i, b_j which are elements of $GF(2^n)$. As a consequence, we can now build a multiplier in the field $GF((2^n)^m)$ by using *identical modules* providing $GF(2^n)$ arithmetic. Its modularity makes the multiplier especially suited for VLSI implementations. The module "$GF(2^n)$ adder" simply consists of n parallel mod 2 adders. For the module "$GF(2^n)$ multiplier" the parallel structures described in Section 2.1 were used. Assuming condition (5) for all field polynomials $Q(y)$ of the ground field, the overall complexity for polynomial multiplication (in AND and XOR gates) follows from the Equations (14) and (15):

$$\#\text{AND} \;=\; n^{2-\log_2 3}\; k^{\log_2 3} \tag{17}$$

$$\#\text{XOR} \;\leq\; \left(\frac{k}{n}\right)^{\log_2 3} (n^2 + 6n - 1) - 8k + 2n \;\;;\;\; \text{certain n} \tag{18}$$

[2]Reference [15] is here wrong by claiming that only $2^i - 4$ coefficients overlap.

where $k = nm$ and $m = 2^t$. Both formulas (17), (18) imply that the *order* of elementary gates increases only proportional to $k^{\log_2 3}$ as k increases if n can be kept under a certain limit. This is in particular possible for all applications where k is a power of two which are of great technical interest.

To achieve an expression for the time complexity, Equation (16) with appropriate expressions for $T_\oplus$ and $T_\otimes$ can be applied. Addition in $GF(2^n)$ has a delay of one XOR gate, i.e. $T_\oplus = \mathcal{T}_{\text{xor}}$. The delay for multiplication, $T_\otimes$, in the ground field $GF(2^n)$ is upper bounded by (8). Hence, the overall delay for parallel multiplication of polynomials of degree $m-1$, $m = 2^t$, over $GF(2^n)$ can be upper bounded by:

$$T \leq \mathcal{T}_{\text{xor}}(2\lceil \log_2 n \rceil + 3\log_2 m) + \mathcal{T}_{\text{and}}. \tag{19}$$

4 Reduction Modulo the Primitive Polynomial

This section describes the second step of the field multiplication, the operation "mod $P(x)$". The pure polynomial multiplication of two polynomials $A(x) \times B(x)$ results in a product polynomial $C'(x)$ over $GF(2^n)$ with $\deg(C'(x)) \leq 2m-2$. In order to perform a multiplication in $GF((2^n)^m)$, $C'(x)$ must be reduced modulo the field polynomial $P(x)$.

The General Case $GF((2^n)^m)$

First, general composite fields $GF((2^n)^m)$ with arbitrary values for n and m will be considered. The modulo operation results in a polynomial $C(x)$ with $\deg(C(x)) \leq m-1$ which represents the desired field element: $C(x) = c_{m-1}x^{m-1} + \cdots + c_0 \equiv C'(x) \bmod P(x)$, where $C(x) \in GF((2^n)^m)$.

The reduction modulo $P(x)$ can be viewed as a linear mapping of the $2m-1$ coefficients of $C'(x)$ into the m coefficients of $C(x)$. This mapping can be represented in a matrix notation as follows:

$$\begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_{m-1} \end{pmatrix} = \begin{pmatrix} 1 & 0 & \cdots & 0 & r_{0,0} & \cdots & r_{0,m-2} \\ 0 & 1 & \cdots & 0 & r_{1,0} & \cdots & r_{1,m-2} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 & r_{m-1,0} & \cdots & r_{m-1,m-2} \end{pmatrix} \begin{pmatrix} c'_0 \\ \vdots \\ c'_{m-1} \\ c'_m \\ \vdots \\ c'_{2m-2} \end{pmatrix} \tag{20}$$

The matrix on the right hand side of (20) consists of a (m, m) identity matrix and a $(m, m-1)$ matrix $\mathbf{R}$ which we may name the *reduction matrix*. $\mathbf{R}$ is solely a function of the chosen field polynomial $P(x) = x^m + p_{m-1}x^{m-1} + \cdots + p_0$, i.e. to every $P(x)$ a reduction matrix is uniquely assigned. $\mathbf{R}$'s recursive dependency on $P(x)$ is the following:

$$r_{ij} = \begin{cases} p_j & ; \quad i = 0, \ldots, m-1 \quad ; \quad j = 0 \\ r_{i-1,j-1} + r_{m-1,j-1}r_{i0} & ; \quad i = 0, \ldots, m-1 \quad ; \quad j = 1, \ldots, m-2 \end{cases} \tag{21}$$

where $r_{i-1,j-1} = 0$ if $j = 0$. From Equation (21) it follows directly that $r_{ij} \in GF(2^n)$ since $p_i \in GF(2^n)$. It should be emphasized that (20) does not require any general multiplication but only additions and multiplications with a constant from $GF(2^n)$. Both operations require only mod 2 adders.

First, a general expression for the *average* complexity will be derived. In general the matrix-vector product in Equation (20) requires $m(m-1)$ constant multiplications and $m(m-1)$ additions. On average one constant multiplication has the complexity given by Equation (10). Thus the average implementational complexity of (20) is:

$$\begin{aligned} \#\overline{\text{XOR}} &= m(m-1) \oplus +m(m-1) \otimes_{cnst} \\ &= \frac{1}{2}k(k(1+\frac{1}{n}) - n - 1), \quad \text{where } k = nm. \end{aligned} \tag{22}$$

However certain field polynomials yield matrices with a complexity considerably smaller than the average complexity in (22). In order to obtain such low complexity polynomials an exhaustive computer based search through *all* primitives polynomials $P(x)$ was performed for each pair of parameters (n, m). The number of primitive polynomials I_p of degree m over $GF(2^n)$ is given by [10]: $I_p = \frac{1}{m}\Phi(2^{mn} - 1)$, where $\Phi()$ denotes the Euler function. The complexity of multiplication with each of the I_p reduction matrices was evaluated as follows. For every matrix the number of additions and constant multiplications was computed. Redundancies within the rows of $\mathbf{R}$, i.e. at least two elements are equal: $r_{ij} = r_{ik}$, were taken into account, thus reducing the number of constant multiplications. However, we did not consider all possible redundancies and the best polynomials P_{opt} found during the search can thus be considered suboptimum.

The Special Case $GF((2^n)^2)$

For the special case $m = 2$ we can perform the two operations polynomial multiplication and modulo reduction in just one single step due to its simplicity. This method was first described in [8]. In this case we know that there exist primitive polynomials of the form $P(x) = x^2 + x + p_0$ [16, Theorem 11]. Hence the multiplication of two field elements in $GF((2^n)^m)$ is the following:

$$C(x) = A(x)B(x) \bmod P(x) = [a_0b_0 + p_0a_1b_1] + x[(a_0 + a_1)(b_0 + b_1) + a_0b_0]. \tag{23}$$

Equation (23) requires only 3 multiplications, 4 additions and one constant multiplication with p_0. Hence, the exhaustive search determined in the cases $m = 2$ those primitive polynomials $P(x)$ which have a constant p_0 with a minimum complexity with respect to constant multiplication.

The maximum delay in Equation (23) is caused by the computation of the coefficient $c_0 = a_0b_0 + p_0a_1b_1$. It is composed of the delays for one general multiplication a_0b_0 and a_1b_1, one multiplication with p_0 and one addition.

k	n	m	$P(x)$	mod XOR	$A\ B \bmod P$ AND	$A\ B \bmod P$ XOR	k^2	$A\ B \bmod P$ $\mathcal{T}_{\text{and}}$	$A\ B \bmod P$ $\mathcal{T}_{\text{xor}}$
4	2	2	$1, 1, \omega^2$	1	**12**	**18**	16	**1**	**4**
6	3	2	$1, 1, \omega^6$	1	**27**	**37**	36	**1**	**5**
8	4	2	$1, 1, \omega^{14}$	1	**48**	**62**	64	**1**	**5**
10	5	2	$1, 1, \omega^3$	3	**75**	**95**	100	**1**	**7**
12	6	2	$1, 1, \omega^{62}$	1	**108**	**130**	144	**1**	**6**
	3	4	$1, 0, 0, 1, \omega^6$	21	**81**	**159**		**1**	**11**
14	7	2	$1, 1, \omega^{124}$	3	**147**	**175**	196	**1**	**8**
16	4	4	$1, 1, 1, 0, \omega$	35	**144**	**258**	256	**1**	**12**
18	9	2	$1, 1, \omega^5$	5	**243**	**281**	324	**1**	**8**
20	5	4	$1, 0, 0, \omega, \omega$	34	**225**	**360**	400	**1**	**14**
22	11	2	$1, 1, \omega^{2036}$	11	**363**	**415**	484	**1**	**12**
24	6	4	$1, \omega^{62}, \omega^{61}, \omega^3\ \omega^2$	60	**324**	**507**	576	**1**	**14**
26	13	2	$1, 1, \omega^{8188}$	7	**507**	**665**	676	**1**	**10**
28	7	4	$1, 0, 0, \omega^{126}\ \omega^{126}$	46	**441**	**632**	784	**1**	**13**
30	15	2	$1, 1, \omega^{32766}$	1	**675**	**733**	900	**1**	**7**
32	4	8	$1, 0, 0, 1, 0, 0, 1, 0, \omega$	91	**432**	**896**	1024	**1**	**15**

Table 1: Composite fields $GF((2^n)^m)$ up to $nm = 32$, primitive field polynomials, and the space complexities and theoretical delays of parallel multipliers

5 Results

Table 1 gives insight in the complexities and architectures of parallel multipliers in composite fields $GF(2^k)$ $k = 2, 4, \ldots, 32$. For each field an optimized field polynomial $P(x)$ and a multiplier with a minimum complexity is given. A description of the table's contents is given below. All columns are explained from left to right, where each column is named after its heading symbol.

k, n, m: k denotes the field order 2^k, where the parameters n and m determine the composition $GF((2^n)^m)$ of the field. The binary field polynomials $Q(y)$ of the ground fields $GF(2^n)$ are given in [14, Table 4.5].

P(x): Primitive polynomials over $GF(2^n)$ are given which possess minimum complexity with respect to the operation "mod$P(x)$". The character ω denotes a primitive element of the field $GF(2^n)$, such that $Q(\omega) = 0$. Each row contains the m coefficients of a polynomial, highest coefficient leftmost (e.g. for $n = m = 2$, $P(x) = x^2 + x + \omega^2$.)

mod: In the cases $m = 4, 8$ the space complexity for the operation mod$P(x)$ is given. For the case $m = 2$ the complexity for multiplying with the coefficient p_0 of $P(x)$ is given.

AB mod **P:** The overall space complexity for a parallel multiplier in $GF((2^n)^m)$ is given in bold face letters.

k²: The space complexity of many previously suggested architectures is lower bounded by k^2-1 XOR gates and k^2 AND gates, where $k=nm$. In order to allow comparison with those multipliers, we provide the values k^2.

A B mod **P:** The theoretical delay of the entire multiplier is shown in bold face letters. However, delays caused by routing or high fanouts which may occur in an actual VLSI implementation are not considered.

As an example, the multiplier over $GF(2^{16})$ is explained below.

Example. The field considered is $GF((2^4)^4)$. The field polynomial of the ground field $GF(2^4)$ is $Q(y)=y^4+y+1$. The primitive polynomial of the composite field is $P(x)=x^4+x^3+x^2+\omega$, where $Q(\omega)=0$. The operation modulo $P(x)$ requires 35 mod 2 adders. The overall complexity for a parallel multiplier in the composite field results in 144 mod 2 multipliers and 258 mod 2 adders. The theoretical delay, i.e. the critical path through the multiplier, consists of 12 $\mathcal{T}_{\text{xor}}$ (XOR gate delays) and 1 $\mathcal{T}_{\text{and}}$ (AND gate delay). Figure 1 provides a block diagram of the multiplier's architecture. The input variables are $a_0,\ldots,a_3$ and $b_0,\ldots,b_3$, the output variables are $c_0,\ldots,c_3$. Each set of variables represents a polynomial, which is an element in $GF((2^4)^4)$ in standard representation. Each variable is actually a four bit wide bus, representing an element in the ground field. The blocks having an "ω" attached are multipliers with the constant element ω. The multiplier was also implemented on an FPGA, serving as a coprocessor for Reed-Solomon decoder with 16 bit symbols realized on a digital signal processor [17].

6 Conclusions

It is shown that the introduction of composite fields $GF((2^n)^m)$ leads to a significantly improved parallel multiplier with respect to the number of mod 2 adders and multipliers if compared to traditional architectures over $GF(2^k)$ with $k=nm$. The multiplication of two polynomials, which is the most costly step in standard based Galois field multiplication, can be performed with an asymptotical complexity of $\mathcal{O}(k^{\log_2 3})$ under the condition that $k=n2^t$ and that n can be kept under a certain limit. It is found that the number of gates for modulo reduction takes less than 10% of the overall gate count for the entire field multiplication for the cases considered.

An improved multiplier is given for every field $GF(2^k)$, $k=2,4,\ldots,32$, if compared to the $2k^2-1$ complexity bound of many traditional architectures. To the authors knowledge the gate count of 48 AND/62 XOR is the lowest one reported in technical literature for bit parallel multiplication in $GF(2^8)$. Considering the wide range of application of this field, e.g. in space communication or in optical storage systems such as CDs, this architecture is certainly of technical interest.

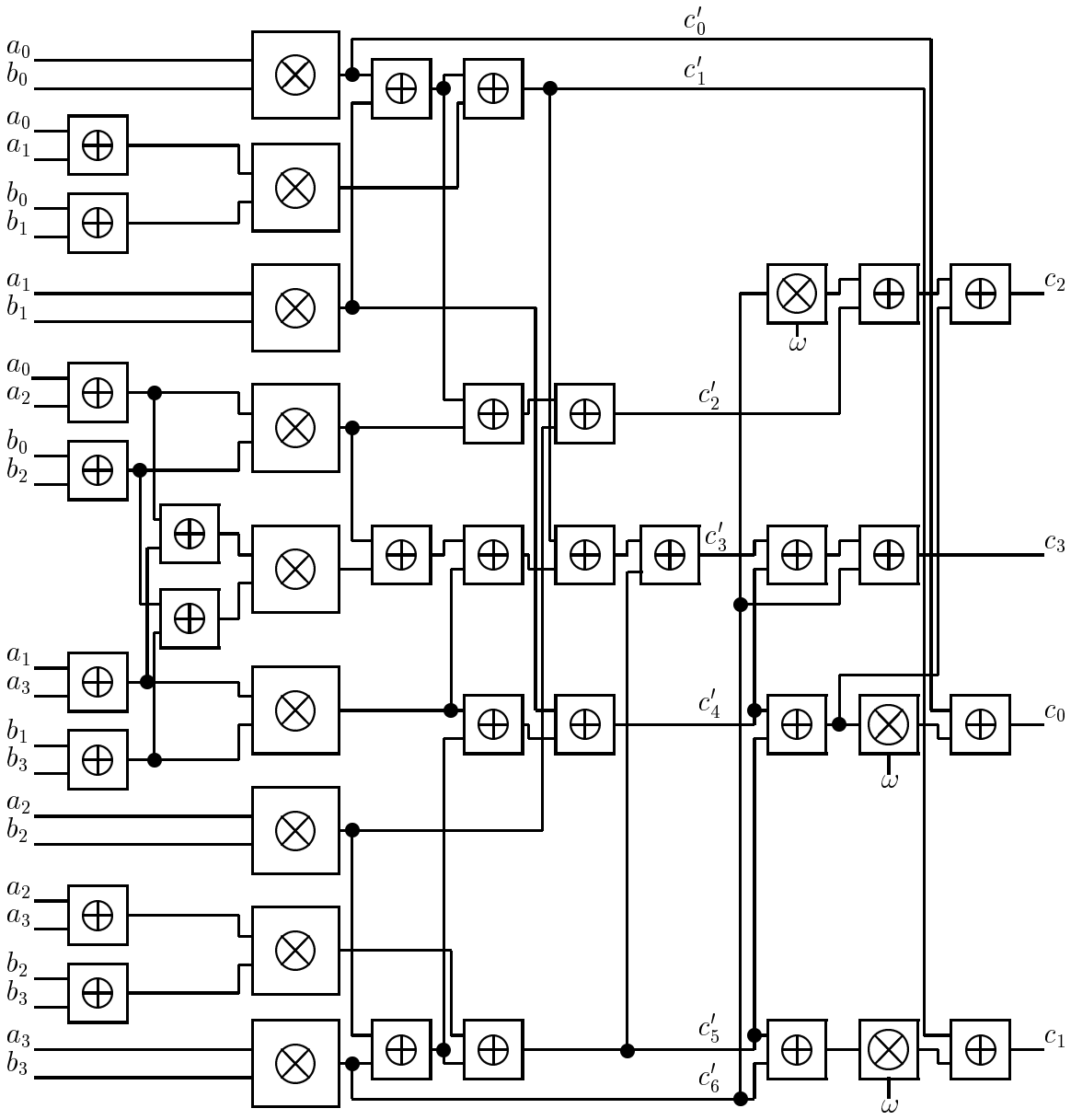

Figure 1: Block diagram of a parallel multiplier in $GF((2^4)^4)$

A further advantage for a VLSI implementation is the high natural modularity of the architecture. A multiplier can be build by using a relatively small number of the two modules $GF(2^n)$ adder and multiplier.

References

[1] A. Odlyzko, "Discrete logarithms in finite fields and their cryptographic significance," in *Lecture Notes in Computer Science 209*, pp. 224–316, Springer-Verlag, Berlin, 1984.

[2] R. Blahut, *Theory and Practice of Error Control Codes.* Reading, Massachusetts: Addison-Wesley, 1983.

[3] C. Wang, T. Truong, H. Shao, L. Deutsch, J. Omura, and I. Reed, "VLSI architectures for computing multiplications and inverses in $GF(2^m)$," *IEEE Trans. Comp.*, vol. C-34, pp. 709–717, August 1985.

[4] T. Itoh and S. Tsujii, "Structure of parallel multipliers for a class of fields $GF(2^k)$," *Inform. and Comp.*, vol. 83, pp. 21–40, 1989.

[5] E. Mastrovito, "VLSI design for multiplication over finite fields $GF(2^m)$," in *Lecture Notes in Computer Science 357*, pp. 297–309, Springer-Verlag, Berlin, March 1989.

[6] M. Hasan, M. Wang, and V. Bhargava, "Modular construction of low complexity parallel multipliers for a class of finite fields $GF(2^m)$," *IEEE Trans. Comp.*, vol. 41, pp. 962–971, August 1992.

[7] V. Afanasyev, "Complexity of VLSI implementation of finite field arithmetic," in *II. Intern. Workshop on Algebraic and Combinatorial Coding Theory*, (Leningrad, USSR), pp. 6–7, September 1990.

[8] V. Afanasyev, "On the complexity of finite field arithmetic," in *5th Joint Soviet-Swedish Intern. Workshop on Information Theory*, (Moscow, USSR), pp. 9–12, January 1991.

[9] A. Pincin, "A new algorithm for multiplication in finite fields," *IEEE Trans. Comp.*, vol. 38, pp. 1045–1049, July 1989.

[10] D. Green and I. Taylor, "Irreducible polynomials over composite Galois fields and their applications in coding techniques," *Proc. IEE*, vol. 121, pp. 935–939, September 1974.

[11] J. Komo and M. Lam, "Primitive polynomials and m-sequences over $GF(q^m)$," *IEEE Trans. Inform. Theory*, vol. 39, pp. 643–647, March 1993.

[12] A. Karatsuba and Y. Ofman, "Multiplication of multidigit numbers on automata," *Sov. Phys.-Dokl. (Engl. transl.)*, vol. 7, no. 7, pp. 595–596, 1963.

[13] D. Knuth, *The Art of Computer Programming. Volume 2: Seminumerical Algorithms.* Reading, Massachusetts: Addison-Wesley, 2nd ed., 1981.

[14] E. Mastrovito, *VLSI Architectures for Computation in Galois Fields.* PhD thesis, Linköping University, Dept. Electr. Eng., Linköping, Sweden, 1991.

[15] R. Fateman, "Polynomial multiplication, powers and asymptotic analysis: Some comments," *SIAM J. Comput.*, vol. 7, pp. 196–21, September 1974.

[16] C. Paar, *Efficient VLSI Architectures for Bit-Parallel Computation in Galois Fields.* PhD thesis, (Engl. transl.), Institute for Experimental Mathematics, University of Essen, Essen, Germany, June 1994. ISBN 3–18–332810–0.

[17] C. Paar and O. Hooijen, "Implementation of a reprogrammable Reed-Solomon decoder over $GF(2^{16})$ on a digital signal processor with external arithmetic unit," in *Fourth International ESA Workshop on Digital Signal Processing Techniques Applied to Space Communications*, (King's College, London), September 26–28 1994.