Overlap-free Karatsuba-Ofman Polynomial Multiplication Algorithms

Haining Fan, Jiaguang Sun, Ming Gu and Kwok-Yan Lam

Published on: IET Information security, vol. 4, no. 1, pp. 8-14, 2010.

I am sorry that I have made the following mistake:

The first subquadratic integer multiplication algorithm was invented by A.A. Karatsuba himself,

not Karatsuba and Ofman. [1]

Abstract

We describe how a simple way to split input operands allows for fast VLSI implementations of subquadratic $GF(2)[x]$ Karatsuba-Ofman multipliers. The theoretical XOR gate delay of the resulting multipliers is reduced significantly. For example, it is reduced by about 33% and 25% for $n = 2^t$ and $n = 3^t$ $(t > 1)$, respectively. To the best of our knowledge, this parameter has never been improved since the original Karatsuba-Ofman algorithm was first used to design $GF(2^n)$ multipliers in 1990.

Index Terms

Karatsuba algorithm, Karatsuba-Ofman algorithm, polynomial multiplication, subquadratic space complexity multiplier, finite fields, Galois fields.

I. Introduction

Published in 1962 [2], Karatsuba-Ofman's algorithm (KOA) was the first integer multiplication method that broke the quadratic complexity barrier in positional number systems. Due to its

Haining Fan, Jiaguang Sun, Ming Gu, Kwok-Yan Lam are with the School of Software, Tsinghua University, Beijing, China. E-mails: {fhn, sunjg, guming, lamky}@tsinghua.edu.cn

simplicity, its polynomial version is widely adopted to design VLSI parallel multipliers in $GF(2^n)$-based cryptosystems [13]-[34].

Two parameters are often used to measure the performance of a $GF(2^n)$ parallel multiplier, namely, the space and time complexities. The space complexity is represented in terms of the total number of 2-input XOR and AND gates used. The corresponding time complexity is given in terms of the maximum delay faced by a signal due to these XOR and AND gates. Symbols "T_A" and "T_X" are often used to represent the delays of one 2-input AND gate and one 2-input XOR gate, respectively. The existing bit parallel $GF(2^n)$ multipliers may be simply classified into the following three categories according to the asymptotic space complexity of the multiplication algorithm: quadratic, subquadratic and hybrid multipliers. A number of quadratic multipliers have been proposed in the literature in which different basis representations of $GF(2^n)$ elements are used, e.g., polynomial, shifted polynomial, normal, dual, weakly dual, and triangular bases. Their time complexities are lower than those of subquadratic multipliers. The main advantage of subquadratic multipliers is that their low asymptotic space complexities make it possible to implement VLSI multipliers for large values of n. But when the size of operands is small, e.g., 32-bit, the space complexity may not remain as the critical factor considered by a cryptographic processor designer. Instead, the computational speed becomes the key factor. Based on this consideration, the hybrid approach is often used to design practical multipliers [6] [18] [21] [23] [32]. These multipliers first perform a few KOA iterations to reduce the whole space complexities, and then a quadratic multiplication algorithm on small input operands to achieve relatively high speed performance. By selecting different stop conditions for the KOA iterations, the hybrid approach can provide a trade-off between the time and space complexities. For the purpose of comparison, reference [21] implemented four parallel $GF(2^{233})$ multipliers on Xilinx FPGAs, namely classical, hybrid Karatsuba, Massey-Omura, and Sunar-Koç, and analyzed their time and space complexities in detail. It was shown that for polynomial bases representation the hybrid Karatsuba is the best choice, while for normal bases the Sunar-Koç. An improved structure of

the hybrid Karatsuba multiplier of [21] was presented later in [32], where different possibilities of implementing Karatsuba multipliers were also studied.

Some other related work on KOA multipliers include the following. In [15], the exact space and time complexities of KOA multipliers were derived. A generalization of the KOA was proposed in [22], and the other generalization, i.e., the Winograd short convolution algorithm, was presented in [25]. In reference [26], a non-redundant KOA multiplier was proposed. Its space complexity is lower than that of the original KOA multiplier. References [27] and [28] presented some improved t-term ($4 < t < 19$) $GF(2)[x]$ Karatsuba-like formulae. These formulae may be mixed with the 2-term and 3-term formulae within recursive constructions, leading to low space complexity subquadratic $GF(2)[x]$ multipliers. FPGA and ASIC KOA implementations can be found in, for example, [19], [21], [29], [31], [32] and [34]. The interested readers are referred to [34] for a more detailed comparison of some hardware designs.

In the work, we will propose a new algorithm for fast hardware implementations of the polynomial KOA. The proposed algorithm uses a simple and straightforward method to split input operands [3] [4]. The theoretical XOR gate delay of the proposed subquadratic Karatsuba-Ofman $GF(2)[x]$ multiplier is reduced significantly. For example, it is reduced by about 33% and 25% for $n = 2^t$ and $n = 3^t$ $(t > 1)$, respectively. To the best of our knowledge, this parameter has never been improved since the original KOA was first used to design $GF(2^n)$ multipliers in 1990 [13].

A. The original KOA in $GF(2)[x]$

Let $A = \sum_{i=0}^{n-1} a_i x^i$ and $B = \sum_{i=0}^{n-1} b_i x^i$ be two $GF(2)[x]$ elements. To explain the general idea of KOA easily, we will assume that $n = 2m = 2^t$ $(t > 1)$ in the following.

First, the previous KOA implementations split polynomials A and B into the "most significant half" and the "least significant half" as follows:

$$A = \sum_{i=0}^{n-1} a_i x^i = x^m \sum_{i=0}^{m-1} a_{m+i} x^i + \sum_{i=0}^{m-1} a_i x^i = x^m A_H + A_L,$$

$$B = \sum_{i=0}^{n-1} b_i x^i = x^m \sum_{i=0}^{m-1} b_{m+i} x^i + \sum_{i=0}^{m-1} b_i x^i = x^m B_H + B_L,$$

where $A_H = \sum_{i=0}^{m-1} a_{m+i} x^i$, $A_L = \sum_{i=0}^{m-1} a_i x^i$, B_H and B_L are defined similarly.

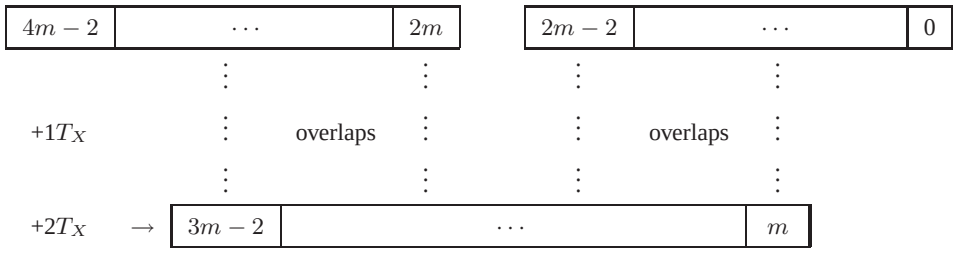
Then the product AB is computed recursively using

$$AB = A_H B_H x^{2m} + \{[(A_H + A_L)(B_H + B_L)] - [A_H B_H + A_L B_L]\} x^m + A_L B_L. \quad (1)$$

We note that "$-$" is the same as "+" in $GF(2)$, and a 2-input XOR gate can be used to realize a "$-$" or "+" operation. For VLSI implementations of (1), the expressions in the two square brackets are calculated concurrently, and one XOR gate delay, i.e., $1T_X$, is required. Then the "$-$" operation is performed at a cost of $1T_X$. Therefore, two XOR gate delays $2T_X$ are required to compute the expression in the curly bracket besides the gate delays to compute the three partial products $A_H B_H$, $A_L B_L$ and $(A_H + A_L)(B_H + B_L)$. Finally, the three polynomials $A_H B_H x^{2m}$, $[(A_H + A_L)(B_H + B_L) - A_H B_H - A_L B_L] x^m$ and $A_L B_L$ in (1) are XORed by adding coefficients of common exponents of x together. The VLSI module used to perform this XOR operation is called the overlap module [32]. In order to explain overlaps of common exponents of x clearly, we present the following table, which shows ranges of x's exponents in these three polynomials.

TABLE I

RANGES OF x'S EXPONENTS IN THE THREE POLYNOMIALS OF (1)



From the table, it is clear that overlaps occur only when $n \geq 4$ (or $m \geq 2$), and there is no overlap when $n = 2$ (or $m = 1$).

Because of these overlaps, one XOR gate delay is required in the overlap module to compute the summation of the three polynomials $A_H B_H x^{2m}$, $[(A_H + A_L)(B_H + B_L) - A_H B_H - A_L B_L] x^m$

and $A_L B_L$ in (1). Therefore, a total of 3 XOR gate delays, i.e., $3T_X$, are required in (1) besides the cost of the recursive computation of the three partial products.

In order to compute the exact complexities of the above binary polynomial KOA, we introduce some symbols from [7]. Let $\mathcal{S}$ and $\mathcal{D}$ stand for "Space" and "Delay", respectively. We use $\mathcal{S}^{\otimes}(n)$ and $\mathcal{S}^{\oplus}(n)$ to denote the numbers of multiplication (AND) and addition (XOR) operations, $\mathcal{D}^{\otimes}(n)$ and $\mathcal{D}^{\oplus}(n)$ the gate delays introduced by multiplication and addition operations, respectively.

Our earlier discussion shows that the XOR gate delay $\mathcal{D}^{\oplus}(n) = \mathcal{D}^{\oplus}(n/2)+3$. It is easy to see that $2T_X$ is required to compute the product of two polynomials of degree 1, i.e., $\mathcal{D}^{\oplus}(2) = 2$. Thus, we have established the recurrence relation of the XOR gate delay. Similarly, we may obtain the recurrence relations of $\mathcal{S}^{\otimes}(n)$, $\mathcal{S}^{\oplus}(n)$ and $\mathcal{D}^{\otimes}(n)$. These recurrence relations describe the time and space complexities of the original KOA [32].

$$\begin{cases} \mathcal{S}^{\otimes}(2) = 3, \\ \mathcal{S}^{\otimes}(n) = 3\mathcal{S}^{\otimes}(n/2); \end{cases} \qquad \begin{cases} \mathcal{D}^{\otimes}(2) = 1, \\ \mathcal{D}^{\otimes}(n) = \mathcal{D}^{\otimes}(n/2); \end{cases}$$

$$\begin{cases} \mathcal{S}^{\oplus}(2) = 4, \\ \mathcal{S}^{\oplus}(n) = 3\mathcal{S}^{\oplus}(n/2) + 4n - 4; \end{cases} \text{ and } \begin{cases} \mathcal{D}^{\oplus}(2) = 2, \\ \mathcal{D}^{\oplus}(n) = \mathcal{D}^{\oplus}(n/2) + 3. \end{cases}$$

After solving the above recurrence relations using the formula derived in [7], we obtain the following complexity results for the binary polynomial KOA [15], [32].

$$\begin{cases} \mathcal{S}^{\otimes}(n) = n^{\log_2 3}, \\ \mathcal{S}^{\oplus}(n) = 6n^{\log_2 3} - 8n + 2, \\ \mathcal{D}^{\otimes}(n) = 1, \\ \mathcal{D}^{\oplus}(n) = 3\log_2 n - 1. \end{cases} \tag{2}$$

B. Motivation

Besides KOA, a Toeplitz matrix-vector product approach was presented recently to construct subquadratic $GF(2^n)$ multipliers [7]. It takes advantage of the shifted polynomial basis [8] and applies the coordinate transformation technique of [9] and [10]. Both the space and time complexities of the resulting multiplier are better than those of the best KOA-based subquadratic

multipliers. For example, with $n = 2^t$ ($t > 1$), the space complexity is about 8% better, while the time complexity is about 33% better, respectively.

Since these Toeplitz matrix-vector product formulae are obtained by transposing [5, Th. 6, p. 17] corresponding polynomial KOA-like formulae, the following question arises naturally: is it possible to reduce the time or space complexity of the KOA-based subquadratic $GF(2)[x]$ VLSI multiplier further? In the next section, we will answer this question positively, namely, we will improve the theoretical XOR gate delay of the KOA-based subquadratic $GF(2)[x]$ multiplier. The improved KOA algorithm can be used to design multipliers in both ring $GF(2)[x]$ and finite field $GF(2^n)$, while the Toeplitz matrix-vector product method cannot be used directly to design $GF(2)[x]$ multipliers.

II. New Method for Fast Implementations of $GF(2)[x]$ KOAs

We first introduce the splitting method in [3] and [4]. Instead of splitting input operands into the "most significant half" and the "least significant half", the method split operands according to the parity of x's exponent. That is to say, we may rewrite A and B as follows

$$A = \sum_{i=0}^{n-1} a_i x^i = \sum_{i=0}^{m-1} a_{2i} x^{2i} + \sum_{i=0}^{m-1} a_{2i+1} x^{2i+1} = \sum_{i=0}^{m-1} a_{2i} x^{2i} + x \sum_{i=0}^{m-1} a_{2i+1} x^{2i},$$

$$B = \sum_{i=0}^{n-1} b_i x^i = \sum_{i=0}^{m-1} b_{2i} x^{2i} + \sum_{i=0}^{m-1} b_{2i+1} x^{2i+1} = \sum_{i=0}^{m-1} b_{2i} x^{2i} + x \sum_{i=0}^{m-1} b_{2i+1} x^{2i}.$$

Now let $y = x^2$, $A_e(y) = \sum_{i=0}^{m-1} a_{2i} y^i$, $A_o(y) = \sum_{i=0}^{m-1} a_{2i+1} y^i$, and $B_e(y)$ and $B_o(y)$ are defined similarly. Operands A and B can be rewritten as $A = A_e(y) + xA_o(y)$ and $B = B_e(y) + xB_o(y)$. Since $A_e(y)$, $A_o(y)$, $B_e(y)$ and $B_o(y)$ are polynomials in y of degree less than m, multiplication operations among them may also be computed recursively. Therefore, we have

the following KOA-like formula

$$\begin{aligned} AB &= (A_e(y) + xA_o(y))(B_e(y) + xB_o(y)) \\ &= \{A_e(y)B_e(y) + x^2A_o(y)B_o(y)\} + x\{A_e(y)B_o(y) + A_o(y)B_e(y))\} \\ &= \{[A_e(y)B_e(y) + yA_o(y)B_o(y)]\} + \\ &\quad x\{[(A_e(y) + A_o(y))(B_e(y) + B_o(y))] - [A_e(y)B_e(y) + A_o(y)B_o(y)]\}. \end{aligned} \tag{3}$$

Clearly, formula (3) also includes three partial products. For VLSI implementations of (3), multiplying a polynomial by x or $y = x^2$ is equivalent to shifting its coefficients left, and no gate is required. It is easy to see that the expansion of $\{A_e(y)B_e(y) + yA_o(y)B_o(y)\}$ in (3) contains only terms with even exponents of x since $y = x^2$, and the expansion of $x\{[(A_e(y) + A_o(y))(B_e(y) + B_o(y))] + [A_e(y)B_e(y) + A_o(y)B_o(y)]\}$ contains only terms with odd exponents of x. Thus, no overlap exists when computing their summation, and no gate is required either. Moreover, the expressions in the three square brackets can be computed concurrently, and these addition operations require one XOR gate delay $1T_X$. Since the "$-$" operation also needs $1T_X$, we know that computing AB via (3) needs $only$ a total of $2T_X$ besides the cost of the recursive computation of the three partial products. Compared to the $3T_X$ gate delays required in formula (1), one XOR gate delay $1T_X$ is saved for each recursive iteration. Consequently, the following recurrence relations, which describe the algorithm complexities, can be established.

$$\begin{cases} \mathcal{S}^{\otimes}(2) = 3, \\ \mathcal{S}^{\otimes}(n) = 3\mathcal{S}^{\otimes}(n/2); \end{cases} \qquad \begin{cases} \mathcal{D}^{\otimes}(2) = 1, \\ \mathcal{D}^{\otimes}(n) = \mathcal{D}^{\otimes}(n/2); \end{cases}$$

$$\begin{cases} \mathcal{S}^{\oplus}(2) = 4, \\ \mathcal{S}^{\oplus}(n) = 3\mathcal{S}^{\oplus}(n/2) + 4n - 4; \end{cases} \text{ and } \begin{cases} \mathcal{D}^{\oplus}(2) = 2, \\ \mathcal{D}^{\oplus}(n) = \mathcal{D}^{\oplus}(n/2) + 2. \end{cases}$$

Their solutions are as follows:

$$\begin{cases} \mathcal{S}^{\otimes}(n) = n^{\log_2 3}, \\ \mathcal{S}^{\oplus}(n) = 6n^{\log_2 3} - 8n + 2, \\ \mathcal{D}^{\otimes}(n) = 1, \\ \mathcal{D}^{\oplus}(n) = 2\log_2 n. \end{cases}$$

Compared to the complexities of the original $GF(2)[x]$ KOA listed in (2), the proposed method reduces the XOR gate delay $\mathcal{D}^{\oplus}(n)$ from $(3\log_2 n - 1)$ to $2\log_2 n$, or by about 33% for $n = 2^t$ ($t > 1$).

Similar to generalizations of the original KOA, which is also called 2-way split, we may derive some KOA-like formulae for j-way splits ($j > 2$). As an example, we now present the $GF(2)[x]$ KOA formula for $n = 3k = 3^t$ $(t > 1)$. It is based on the following 6-multiplication formula [5, p. 35].

$$\begin{aligned} & (a_2x^2 + a_1x + a_0)(b_2x^2 + b_1x + b_0) \\ = \; & a_0b_0 + [(a_0 + a_1)(b_0 + b_1) + a_0b_0 + a_1b_1]x + \\ & [(a_0 + a_2)(b_0 + b_2) + a_0b_0 + a_2b_2 + a_1b_1]x^2 + \\ & [(a_1 + a_2)(b_1 + b_2) + a_1b_1 + a_2b_2]x^3 + a_2b_2x^4. \end{aligned}$$

Let $y = x^3$ and split A as follows

$$\begin{aligned} A &= \sum_{i=0}^{n-1} a_i x^i = \sum_{i=0}^{k-1} a_{3i}x^{3i} + x\sum_{i=0}^{k-1} a_{3i+1}x^{3i} + x^2\sum_{i=0}^{k-1} a_{3i+2}x^{3i} \\ &= A_0(y) + xA_1(y) + x^2A_2(y), \end{aligned}$$

where $A_0(y) = \sum_{i=0}^{k-1} a_{3i}y^i$, $A_1(y) = \sum_{i=0}^{k-1} a_{3i+1}y^i$ and $A_2(y) = \sum_{i=0}^{k-1} a_{3i+2}y^i$.

Then we have

$$\begin{aligned} AB &= \{A_0B_0 + y[(A_1 + A_2)(B_1 + B_2) + A_1B_1 + A_2B_2]\} + \\ & x\{(A_0 + A_1)(B_0 + B_1) + A_0B_0 + A_1B_1 + yA_2B_2\} + \\ & x^2\{(A_0 + A_2)(B_0 + B_2) + A_0B_0 + A_1B_1 + A_2B_2\}, \end{aligned}$$

where "(y)"s in expressions $A_i(y)$ and $B_i(y)$ are omitted.

There are four partial products in the first curly bracket, and they are polynomials in y of degrees $2k-2$, $2k-1$, $2k-1$ and $2k-1$, respectively. Since the constant terms of the last three partial products are zeroes, we know that computing the expression in the first curly bracket requires $2k+(2k-2)+(2k-1)+(2k-1)=8k-4$ XOR gates. Similarly, it is easy to see that the total number of the XOR gates required in the last two curly brackets are $2k+(2k-2)+(2k-1)+(2k-1)=8k-4$ and $2k+3(2k-1)=8k-3$, respectively. But the summation $A_0B_0+A_1B_1$, which appears in the last two curly brackets, can be reused. Therefore, $2k-1$ XOR gates can be saved, and the total number of the XOR gates required in the above formula is $(8k-4)+(8k-4)+(8k-3)-(2k-1)=22k-10$ besides the cost of the recursive computation of the six partial products. Based on the above discussion, we obtain the following recurrence relations that describe the complexities of this formula. Their solutions will be presented in Table II in the next subsection.

$$\begin{cases} \mathcal{S}^{\otimes}(3)=6, \\ \mathcal{S}^{\otimes}(n)=6\mathcal{S}^{\otimes}(n/3); \end{cases} \qquad \begin{cases} \mathcal{D}^{\otimes}(3)=1, \\ \mathcal{D}^{\otimes}(n)=\mathcal{D}^{\otimes}(n/3); \end{cases}$$

$$\begin{cases} \mathcal{S}^{\oplus}(3)=12, \\ \mathcal{S}^{\oplus}(n)=6\mathcal{S}^{\oplus}(n/3)+\frac{22}{3}n-10; \end{cases} \text{ and } \begin{cases} \mathcal{D}^{\oplus}(3)=3, \\ \mathcal{D}^{\oplus}(n)=\mathcal{D}^{\oplus}(n/3)+3. \end{cases}$$

A. Comparisons

Table II compares asymptotic complexities of the proposed formulae with the previous KOA and Toeplitz matrix-vector product (TMVP) formulae over the ground field $GF(2)$, where #AND and #XOR denote the total numbers of AND and XOR gates, respectively. The size of operands is assumed to be $n=2^t$ or 3^t ($t>1$). These comparisons are made from a theoretical viewpoint. For practical designs of VLSI multipliers, it is a better choice to merge the proposed method into the hybrid approach discussed in the introduction section.

As shown in the table, the proposed method and the previous KOA have the same space complexities, but the XOR gate delay of the proposed method outperforms the previous KOA when $t > 1$. We list complexities of the TMVP in the table because both KOA and TMVP can be used to design $GF(2^n)$ subquadratic parallel multipliers, which is an important application field of these two algorithms. But we must emphasize that these two algorithms are $distinct$, and each of them have their own application fields [6]. Take the $GF(2^n)$ subquadratic parallel multiplier as an example. Since there is no known value of n for which an irreducible polynomial of weight $w < 6$ does not exist [11], we need only to select either an irreducible trinomial or an irreducible pentanomial of degree n to generate $GF(2^n)$. In order to adopt the TMVP approach in the design stage, the coordinate transformation technique must be used to obtain the desired Toeplitz matrix [7]. The corresponding transformation matrices for irreducible trinomials and a special type of irreducible pentanomials $f(u) = u^n + u^{k+1} + u^k + u^{k-1} + 1$ $(1 < k < n - 1)$ have been derived when the $GF(2^n)$ elements are represented in the shifted polynomial basis [7]. But no explicit transformation matrices are currently available for other bases, e.g., the polynomial basis. On the other hand, the KOA-based $GF(2^n)$ subquadratic parallel multiplier consists of two steps: (1) the KOA multiplication, and (2) a modulo reduction operation using an irreducible polynomial. The second step, which depends on the form of the field generating irreducible polynomials, has been studied by many authors. Therefore, a hardware engineer can use these theoretical results directly to design an $GF(2^n)$ subquadratic parallel multiplier. The interested reader is referred to a recent survey paper [12] for more details.

B. An Example

We now present an example to compare the proposed method with the original KOA.

Let $A = a_3x^3 + a_2x^2 + a_1x + a_0 = A_Hx^2 + A_L$ and $B = b_3x^3 + b_2x^2 + b_1x + b_0 = B_Hx^2 + B_L$, where $A_H = a_3x + a_2$, $A_L = a_1x + a_0$, $B_H = b_3x + b_2$ and $B_L = b_1x + b_0$ are polynomials of

TABLE II

COMPARISONS OF ASYMPTOTIC COMPLEXITIES

n	Algorithm	#AND	#XOR	Gate delay
2^t	KOA [15] [33]	$n^{\log_2 3}$	$6n^{\log_2 3} - 8n + 2$	$(3\log_2 n - 1)T_X + T_A$
	Proposed	$n^{\log_2 3}$	$6n^{\log_2 3} - 8n + 2$	$(2\log_2 n)T_X + T_A$
	TMVP [7]	$n^{\log_2 3}$	$5.5n^{\log_2 3} - 6n + 0.5$	$(2\log_2 n)T_X + T_A$
3^t	KOA [15] [33]	$n^{\log_3 6}$	$\frac{16}{3}n^{\log_3 6} - \frac{22}{3}n + 2$	$(4\log_3 n - 1)T_X + T_A$
	Proposed	$n^{\log_3 6}$	$\frac{16}{3}n^{\log_3 6} - \frac{22}{3}n + 2$	$(3\log_3 n)T_X + T_A$
	TMVP [7]	$n^{\log_3 6}$	$\frac{24}{5}n^{\log_3 6} - 5n + \frac{1}{5}$	$(3\log_3 n)T_X + T_A$

degree 1 in x. Then the original KOA computes the product AB using

$$AB = A_H B_H x^4 + \{[(A_H + A_L)(B_H + B_L)] + [A_H B_H + A_L B_L]\}x^2 + A_L B_L. \quad (4)$$

There are three products of polynomials of degree 1 in (4), and they can be computed recursively using the KOA at a cost of $2T_X$. For example, $A_L B_L = (a_1 x + a_0)(b_1 x + b_0)$ can be computed using

$$(a_1 x + a_0)(b_1 x + b_0) = a_1 b_1 x^2 + \{[(a_1 + a_0)(b_1 + b_0)] + [a_1 b_1 + a_0 b_0]\}x + a_0 b_0. \quad (5)$$

To show the role of the overlap in (4), let group the three products in (4) and write them as polynomials of degree 2 in x as follows:

$$\begin{aligned} k_2 x^2 + k_1 x + k_0 &:= A_H B_H; \\ d_2 x^2 + d_1 x + d_0 &:= [(A_H + A_L)(B_H + B_L)] + [A_H B_H + A_L B_L]; \\ e_2 x^2 + e_1 x + e_0 &:= A_L B_L. \end{aligned}$$

Then we have

$$\begin{aligned} AB &= (k_2 x^2 + k_1 x + k_0)x^4 + (d_2 x^2 + d_1 x + d_0)x^2 + (e_2 x^2 + e_1 x + e_0) \\ &= k_2 x^6 + k_1 x^5 + (k_0 + d_2)x^4 + d_1 x^3 + (d_0 + e_2)x^2 + e_1 x + e_0. \end{aligned} \quad (6)$$

Clearly, one XOR gate delay $1T_X$ is required to compute the overlap summations $(k_0 + d_2)$ and $(d_0 + e_2)$. Since we need $2T_X$ to perform the XOR operations in the curly bracket of (4), we know that the total number of XOR gate delays of the original KOA is 2+1+2=5.

Let $y = x^2$. The proposed method splits A and B as $A = a_2x^2 + a_0 + x(a_3x^2 + a_1) = A_e(y) + xA_o(y)$ and $B = B_e(y) + xB_o(y)$, where $A_e(y) = a_2y + a_0$, $A_o(y) = a_3y + a_1$, $B_e(y) = b_2y + b_0$ and $B_o(y) = b_3y + b_1$ are polynomials of degree 1 in y.

From (3), the proposed method computes AB using

$$\begin{aligned} AB &= (A_e(y) + xA_o(y))(B_e(y) + xB_o(y)) \\ &= \{A_e(y)B_e(y) + yA_o(y)B_o(y)\} + \\ &\quad x\{[(A_e(y) + A_o(y))(B_e(y) + B_o(y))] + [A_e(y)B_e(y) + A_o(y)B_o(y)]\}. \end{aligned}$$

Now define four polynomials of degree 2 in y as follows:

$$\begin{aligned} p_2y^2 + p_1y + p_0 &:= A_e(y)B_e(y); \\ q_2y^2 + q_1y + q_0 &:= A_o(y)B_o(y); \\ r_2y^2 + r_1y + r_0 &:= [(A_e(y) + A_o(y))(B_e(y) + B_o(y))]; \\ s_2y^2 + s_1y + s_0 &:= [A_e(y)B_e(y) + A_o(y)B_o(y)]. \end{aligned}$$

We need $1T_X$ to perform "+" operations in the last two equations. Since the proposed method is identical to the original KOA when the two input polynomials are of degree 1, i.e., formula (5), we need $2T_X$ to compute the three products of polynomials of degree 1 in y in the above four equations. Thus, we need a total of $3T_X$ to obtain all $GF(2)$ elements p_i, q_i, r_i and s_i in

the above four equations, where $i = 0$, 1 and 2. Now, the product AB can be computed using

$$\begin{aligned} AB &= \sum_{i=0}^{6} c_i x^i = \left(\sum_{i=0}^{3} a_i x^i\right)\left(\sum_{i=0}^{3} b_i x^i\right) \\ &= \{(p_2 y^2 + p_1 y + p_0) + y(q_2 y^2 + q_1 y + q_0)\} + \\ &\quad x\{(r_2 y^2 + r_1 y + r_0) + (s_2 y^2 + s_1 y + s_0)\} \\ &= q_2 x^6 + [p_2 + q_1]x^4 + [p_1 + q_0]x^2 + p_0 + \\ &\quad [r_2 + s_2]x^5 + [r_1 + s_1]x^3 + [r_0 + s_0]x. \end{aligned} \tag{7}$$

Clearly, one XOR gate delay $1T_X$ is required to obtain the summations in the five square brackets. Therefore, the total number of XOR gate delays required to compute AB is 3+1=4, and $1T_X$ is saved compared to the original KOA.

The following arithmetic circuit illustrates the two-level recursion formula (7). The circuits in the three rectangular dotted boxes are the same, and each of them implements the original KOA formula (5), which computes the product of two input polynomials of degree 1. Due to the parallelism, the six XOR operations "$+$" in the six doted circles contribute no gate delay to the total XOR gate delays of formula (7). The interested reader may compare this circuit diagram to Figure 8.1 of [6, p. 222] which illustrates the original KOA two-level recursion formula (6).

III. Conclusions

We have proposed a new method to implement the polynomial KOA for VLSI multipliers. It eliminates overlaps in the previous designs. The XOR gate delay of the proposed $GF(2)[x]$ KOA is significantly better than that of the previous KOA. Besides the theoretical significance, the proposed method is also suitable for practical VLSI applications, e.g., designs of hybrid $GF(2^n)$ multipliers.

Acknowledgment

The authors thank Dr. P. Zimmermann for pointing the splitting method of [4] out to us after we posted a preliminary version of this paper on the website "http://eprint.iacr.org" on October 7, 2007.

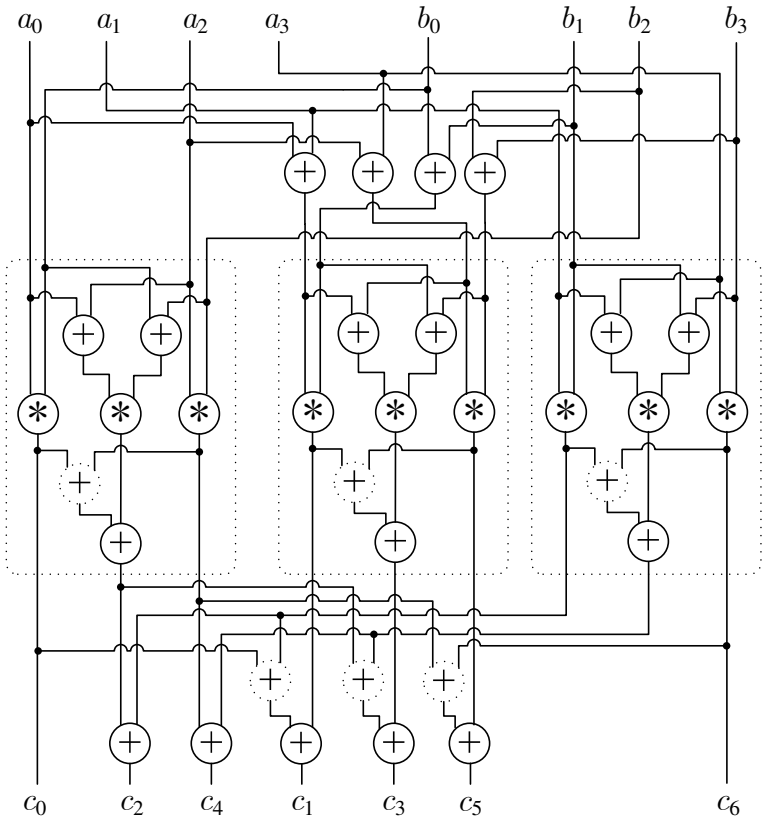


Fig. 1. An arithmetic circuit illustrating formula (7).

REFERENCES

[1] Karatsuba, A. : "The complexity of computations," Proc. Steklov Inst. Math. vol. 211, pp.169-183, 1995.

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

[3] Moenck, R. T. : "Practical fast polynomial multiplication," *Proc. 1976 ACM. Symposium on Symbolic and Algebraic Computation*, pp.136-148, 1976

[4] Hanrot, G., and Zimmermann, P. : "A Long Note on Mulders' Short Product," *Journal of Symbolic Computation* vol.37 , pp.391-401, 2004

[5] Winograd, S. : "Arithmetic Complexity of Computations", SIAM, 1980.

[6] Gathen, J. V. Z., and Gerhard, J. : "Modern Computer Algebra," Cambridge Univ. Press, First ed., 1999, Second ed., 2003.

[7] Fan, H., and Hasan, M. A. : "A New Approach to Subquadratic Space Complexity Parallel Multipliers for Extended Binary Fields," *IEEE Transactions on Computers*, vol. 56, no. 2, pp. 224-233, Feb. 2007.

[8] Fan, H., and Dai, Y. : "Fast bit parallel $GF(2^n)$ Multiplier for All Trinomials," *IEEE Transactions on Computers*, vol. 54, no. 4, pp. 485-490, Apr. 2005.

[9] Hasan, M. A., and Bhargava, V. K. : "Division and Bit-serial Multiplication over $GF(q^m)$," *IEE Proceedings-E*, vol. 139, no. 3, pp. 230-236, May 1992.

[10] Hasan, M. A. , and Bhargava, V. K. : "Architecture for Low Complexity Rate-Adaptive Reed-Solomon Encoder," *IEEE Transactions on Computers*, vol. 44, no. 7, pp. 938-942, July 1995.

[11] Seroussi, G. : "Table of Low-Weight Binary Irreducible Polynomials," *Technical Report HPL*-98-135, *Hewlett-Packard Laboratories*, Palo Alto, Calif., Aug. 1998 [Online]. Available: http://www.hpl.hp.com/techreports/98/HPL-98-135.html

[12] Erdem, S. S. , Yanik, T. , and Koç, Ç. K. : "Polynomial Basis Multiplication over $GF(2^m)$," *Acta Applicandae Mathematicae: An International Survey Journal on Applying Mathematics and Mathematical Applications*, vol. 93, no. 1-3, pp. 33-55, 2006.

[13] Afanasyev, V.B. : "Complexity of VLSI Implementation of Finite Field Arithmetic," *Proc. II. Intern. Workshop on Algebraic and Combinatorial Coding Theory*, USSR, pp.6-7, 1990

[14] Paar, C. : "Efficient VLSI Architectures for Bit-Parallel Computation in Galois Fields," PhD thesis, University of Essen, Germany, 1994

[15] Paar, C. : "A New Architecture for a Parallel Finite Field Multiplier with Low Complexity Based on Composite Fields," *IEEE Transactions on Computers*, vol. 45, no. 7, pp. 856-861, July 1996.

[16] Paar, C., Cleischmann, V.B. , and Roelse, P. : "Efficient Multiplier Schemes for Galois Fields $GF(2^{4n})$, " *IEEE Transactions on Computers*, vol. 47, no. 2, pp. 162-170, Feb. 1998.

[17] Elia, M., Leone, M., and Visentin, C. : "Low Complexity Bit-parallel Multipliers for $GF(2^m)$ with Generator Polynomial $x^m + x^k + 1$," *IEE Electronics Letters*, vol. 35, no.7, pp.551-552, 1999.

[18] Leone, M. : "A New Low Complexity Parallel Multiplier for a Class of Finite Fields," *Proc. Cryptographic Hardware and Embedded Systems (CHES 2001)*, LNCS 2162, pp. 160-170, 2001.

[19] Jung, M., Madlener, F., Ernst,M., and Huss, S. : "A Reconfigurable Coprocessor for Finite Field Multiplication in $GF(2^n)$," *Proc. IEEE Workshop Heterogeneous reconfigurable Systems on Chip*, 2002.

[20] Ernst, M., Jung,M., Madlener,F., Huss, S., and Blumel, R. : "A Reconfigurable System on Chip Implementation for Elliptic Curve Cryptography over $GF(2^n)$," *Proc. Cryptographic Hardware and Embedded Systems (CHES 2002)*, LNCS 2523, pp. 381-399, 2003.

[21] Grabbe, C., Bednara, M., Shokrollahi, J., Teich, J., and Gathen, J. V. Z. : "FPGA Designs of Parallel High Performance $GF(2^{233})$ Multipliers," *Proc. Int'l Symposium on Circuits and Systems (ISCAS 2003)*, vol. II, pp. 268-271, 2003.

[22] Weimerskirch, A., and Paar, C. : "Generalizations of the Karatsuba Algorithm for Efficient Implementations," 2003, $http://www.crypto.ruhr-uni-bochum.de/imperia/md/content/texte/kaweb.pdf$.

[23] Rodríguez-Henríquez, F., and Koç, Ç. K. : "On Fully Parallel Karatsuba Multipliers for $GF(2^m)$," *Proc. Int'l Conf. Computer Science and Technology (CST 2003)*, pp. 405-410, 2003.

[24] Erdem, S. S. , and Koç, Ç. K. : "A Less Recursive Variant of Karatsuba-Ofman Algorithm for Multiplying Operands of Size a Power of Two," *Proc. 16th IEEE Symposium on Computer Arithmetic (Arith-16 2003)*, pp. 28-35, 2003.

[25] Sunar, B. : "A Generalized Method for Constructing Subquadratic Complexity $GF(2^k)$ Multipliers," *IEEE Transactions on Computers*, vol. 53, no. 9, pp. 1097-1105, Sept. 2004.

[26] Chang,N. S. , Kim, C. H. , Park, Y. H. , and Lim, J. : "A Non-Redundant and Efficient Architecture for Karatsuba-Ofman Algorithm," *Proc. 8th International Conf. on Information Security (ISC 2005)*, LNCS 3650, pp. 288-299, 2005.

[27] Montgomery, P. L. : "Five, Six, and Seven-Term Karatsuba-Like Formulae," *IEEE Transactions on Computers*, vol. 54, no. 3, pp. 362-369, Mar. 2005.

[28] Fan H., and Hasan, M. A. : "Comments on "Five, Six, and Seven-Term Karatsuba-Like Formulae"," *IEEE Transactions on Computers*, vol. 56, no. 5, pp. 716-717, May 2007.

[29] Dyka. Z., and Langendoerfer, P. : "Area Efficient Hardware Implementation of Elliptic Curve Cryptography by Iteratively Applying Karatsuba's Method," *Proc. Conf. on Design, Automation and Test in Europe 2005*, pp. 70-75, 2005.

[30] Chang, K. Y. , Hong, D., and Cho, H. S. : "Low Complexity Bit-Parallel Multiplier for $GF(2^m)$ Defined by All-One Polynomials Using Redundant Representation" *IEEE Transactions on Computers*, vol. 54, no. 12, pp. 1628-1630, Dec. 2005.

[31] Cheng, L. S. , Miri, A. , and Yeap, T. H. : "Improved FPGA Implementations of Parallel Karatsuba Multiplication over $GF(2^n)$," *Proc. 23rd Biennial Symposium on Communications*, 2006.

[32] Gathen, J. V. Z., and Shokrollahi, J. : " Efficient FPGA-based Karatsuba Multipliers for Polynomials over F_2," *Proc. 12th Workshop on Selected Areas in Cryptography (SAC 2005)*, LNCS 3897 pp.359-369, 2006.

[33] Gathen, J. V. Z., and Shokrollahi, J. : "Fast arithmetic for polynomials over F_2 in hardware," *Proc. IEEE Workshop on Information Theory*, pp.107-111, 2006.

[34] Peter, P., and Langendorfer, P. : "An Efficient Polynomial Multiplier in $GF(2^m)$ and its Application to ECC Designs," *Proc. Conf. on Design, Automation and Test in Europe 2007*, pp. 1253-1258, 2007.