This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2015.2456023, IEEE Transactions on Computers

High-Speed Hybrid-Double Multiplication Architectures Using New Serial-Out Bit-Level Mastrovito Multipliers

Ebrahim A. Hasan Abdulrahman, and Arash Reyhani-Masoleh, *Member, IEEE*

Abstract—The Serial-out bit-level multiplication scheme is characterized by an important latency feature. It has an ability to sequentially generate an output bit of the multiplication result in each clock cycle. However, the computational complexity of the existing serial-out bit-level multipliers in $GF(2^m)$ using normal basis representation, limits its usefulness in many applications; hence, an optimized serial-out bit-level multiplier using polynomial basis representation is needed. In this paper, we propose new serial-out bit-level Mastrovito multiplier schemes. We show that in terms of the time complexities, the proposed multiplier schemes outperform the existing serial-out bit-level schemes available in the literature. In addition, using the proposed multiplier schemes, we present new hybrid-double multiplication architectures. To the best of our knowledge, this is the first time such a hybrid multiplier structure using the polynomial basis is proposed. Prototypes of the presented serial-out bit-level schemes and the proposed hybrid-double multiplication architectures (10 schemes in total) are implemented over both $GF(2^{163})$ and $GF(2^{233})$, and experimental results are presented.

Index Terms—serial-out, polynomial basis, bit-level multiplier, Mastrovito multiplier, hybrid-double multiplication

1 Introduction

Finite field arithmetic has been widely applied in applications of different fields like error-control coding, cryptography, and digital signal processing [1], [2], [3], [4]. The arithmetic operations in the finite fields over characteristic two $GF(2^m)$ have gained widespread use in public-key cryptography such as point multiplication in elliptic curve cryptography [5], [6], and exponentiation-based cryptosystems [7], [8]. The finite field $GF(2^m)$ has 2^m elements and each of its elements can be represented by its m binary coordinates based on the choice of field-generating polynomial. For such a representation, the addition is relatively straight-forward by bit-wise XORing of the corresponding coordinates of two field elements. On the other hand, the multiplication operation requires larger and slower hardware. Other complex and time-consuming operations such as exponentiation, and division/inversion are implemented by the iterative application of the multiplication operations. Much of the ongoing research in this area is focused on finding new architectures to implement the arithmetic multiplication operation more efficiently (see for example [9], [10], [11]).

Finite field multipliers with different properties are obtained by choosing different representations of the field elements. With the advantages of low design complexity, simplicity, regularity, and modularity in architecture, the standard or polynomial basis (PB) representation, is extensively used for cryptographic applications [12], [13]. In the PB, a multiplier requires a polynomial multiplication followed by a modular reduction. In practice, these two steps can be combined into a single step by using the so-called Mastrovito matrix [14], [15]. The properties and complexities of the PB multipliers depend heavily on the choice of a field-generating polynomial. In this paper, we first consider an irreducible polynomial with ω, $\omega \geq 3$, non-zero terms (denoted by ω-nomials). We then obtain a further optimized structure for the special irreducible trinomial ($\omega = 3$).

The implementation of finite field multipliers can be categorized, in terms of their structures, into three groups of parallel-level, digit-level and bit-level types. The bit-level multiplier scheme, which processes one bit of input per clock cycle, is area-efficient and suitable for resource-constrained and low-weighted devices. The bit-level type multiplication algorithms, when the PB is used are classified as least significant bit first (LSB-first), and most significant bit first (MSB-first) schemes [16].

The bit-level multiplier can be further categorized into two types of either parallel or serial output. In the traditional parallel-out bit-level (POBL) multipliers [16], all of the output bits of the multiplication (from the first bit to the last bit) are generated at the end of the last clock cycle. Serial-out bit-level (SOBL) multipliers, on the other hand, generate an output bit of the product sequentially, after a certain number of clock cycles. A multiplication scheme based on serial-out architecture, i.e., SOBL, has certain advantages as compared to the traditional parallel-out architecture. For instance, combining a SOBL with a traditional LSB-first POBL one, would make fast exponentiation and inversion possible [17], [18]. The

- *Ebrahim A. Hasan Abdulrahman is with the Faculity of Information Technology, Department of Computer Engineering, The University of Bahrain, Sakheer, Bahrain (E-mail: eabdulrahman@uob.edu.bh).*
- *Arash Reyhani-Masoleh is with the Department of Electrical and Computer Engineering, Western University, London, Ontario, Canada (E-mail: areyhani@uwo.ca).*

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2015.2456023, IEEE Transactions on Computers

author of [19], has proposed a SOBL multiplication architecture that is constructed by the trinomials and the ω-nomials irreducible polynomials in $GF(2^m)$ using PB representation. In this paper, alternative schemes for the serial-out multiplication in the PB over $GF(2^m)$ for both trinomial and ω-nomial irreducible polynomial are developed. We summarize our contributions as follows:

- We have proposed a new scheme for the SOBL multiplication architecture in the PB over $GF(2^m)$ for the ω-nomials, then we further optimized it for the irreducible trinomials. Both schemes have lower critical path delay compared to previously published results.
- In order to investigate the applicability of the proposed SOBL schemes, we employed the proposed two SOBL schemes, and the SOBL scheme proposed in [19], to present, to our knowledge, the first approach for hybrid-double multiplication architecture in the PB over $GF(2^m)$.
- We extended the traditional POBL multiplier schemes presented in [16] to propose two new LSB-first/MSB-first POBL double multiplication architectures, which perform two multiplications together after $2m$ clock cycles.
- To obtain the actual implementation results, all the proposed schemes, i.e., 2 SOBL multipliers, 3 hybrid-double multiplication architectures, 2 double multiplication architectures, and the counterpart ones, i.e., LSB-first POBL [16], MSB-first POBL [16], and SOBL scheme proposed in [19] are coded in VHDL (10 schemes in total), and implemented on ASIC technology over both $GF(2^{163})$ and $GF(2^{233})$.

The organization of this paper is as follows. Notation and mathematical background are given in Section 2. In Section 3, the formula for a new SOBL multiplication is presented. Section 4 is the core of our paper, in which a novel architecture for the SOBL multiplier for both the trinomial and the ω-nomial irreducible polynomial are presented. In Section 5, new double multiplication architectures using PB are proposed and discussed. In Section 6, the proposed architectures and the previously reported ones are compared in terms of area, delay and I/O loading complexities. In Section 7, the performance of the proposed multiplier schemes are investigated by implementing each multiplier and the counterpart multipliers as well as the double multiplication architectures on ASIC technology. Finally, the conclusion is presented in Section 8.

2 PRELIMINARIES

The binary extension field $GF(2^m)$ can be viewed as an m-dimensional vector space defined over $GF(2)$ [1]. A set of m linearly independent vectors (elements of $GF(2^m)$) is chosen to serve as the basis of representation. An explicit choice for a basis is the ordered set $\{\alpha^{m-1}, \cdots, \alpha^2, \alpha, 1\}$, where $\alpha \in GF(2^m)$ and is a root of an irreducible polynomial $P(x)$. This basis is called the polynomial basis (PB). Each element is represented by a polynomial of degree $m-1$, whose coefficients are the binary digits 0 or 1. All arithmetic operations are performed modulo 2.

A straightforward $GF(2^m)$ multiplication computations consists of two parts, the product of two field elements, followed by a modular reduction [20], [21]. Suppose $A = (a_{m-1}, \cdots, a_1, a_0)$, $B = (b_{m-1}, \cdots, b_1, b_0)$ are two arbitrary field elements, i.e., $A, B \in GF(2^m)$, then to obtain the field multiplication of A and B, AB is computed first; it is then followed by the modular reduction, i.e., $C \triangleq AB \bmod P(\alpha)$.

In [14], [15], Mastrovito has proposed an efficient dedicated parallel multiplication method that combines the two parts of the product and the modular reduction into a single step. He showed that the coordinates of C are obtained from the *matrix-by-vector* product of

$$\mathbf{c} = [c_{m-1}, \cdots, c_1, c_0]^T = \mathbf{M} \cdot \mathbf{b}, \tag{1}$$

where T denotes the transposition; the column vector $\mathbf{b} = [b_{m-1}, \cdots, b_1, b_0]^T$ contains the coordinates of the multiplier $B = (b_{m-1}, \cdots, b_1, b_0) \in GF(2^m)$, and $\mathbf{M}$ is an $m \times m$ binary matrix whose entries depend on the coordinates of $A \in GF(2^m)$. This equation was implicitly used in [22], [23], and [24] to derive the parallel-level multiplier and is now used in this work to design a new SOBL multiplier.

Sunar and Koç [22] have studied the Mastrovito matrix $\mathbf{M}$, and have presented a formulation for the Mastrovito algorithm using the irreducible trinomials. Halbutoğullari and Koç in [23] have presented a new architecture for the Mastrovito multiplication and have also shown that the coefficient of the product AB can be obtained from the *matrix-by-vector* product of

$$\mathbf{d} \triangleq [d_{2m-2}, \cdots, d_m, d_{m-1}, \cdots, d_0]^T = \mathbf{Z} \cdot \mathbf{b},$$

where $\mathbf{Z}$ is a $2m-1 \times m$ binary matrix whose entries are

$$\mathbf{Z} \triangleq \begin{pmatrix} a_0 & 0 & \cdots & 0 & 0 \\ a_1 & a_0 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{m-2} & a_{m-3} & \cdots & a_0 & 0 \\ a_{m-1} & a_{m-2} & \cdots & a_1 & a_0 \\ 0 & a_{m-1} & \cdots & a_2 & a_1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & a_{m-1} & a_{m-2} \\ 0 & 0 & \cdots & 0 & a_{m-1} \end{pmatrix}. \tag{2}$$

In [24], Zhang and Parhi have proposed the use of a parallel-level Mastrovito multiplier based on a systematic design approach for the technique proposed in [23].

2.1 Notations

Let us now introduce the following notations, which will be used in this paper: Column vectors are represented by small boldfaced characters. Matrices are represented by capital boldfaced characters, and to represent the entries

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2015.2456023, IEEE Transactions on Computers

of a matrix, we use the common notation used in the literature such as in [22], [23], [24], [25], and [19]. These notations are summarized in TABLE 1.

TABLE 1: List of notations.

Symbol	Description
$\mathbf{b}$, $\mathbf{b}^T$	Column and row vectors, respectively.
$\mathbf{M}(i, :)$	The i^{th} row of the matrix $\mathbf{M}$.
$\mathbf{M}(:, j)$	The j^{th} column of the matrix $\mathbf{M}$.
$\mathbf{M}(i: j)$	An entry with position (i,j) of the matrix $\mathbf{M}$.
$[v_j, \cdots, v_i]$	The range of bits in the vector $\mathbf{v}$ from position i to position j, $j > i$.
$\langle r_j, \cdots, r_i \rangle$	The range of bits in the register $\langle R \rangle$ from position i to position j, $j > i$.
$\mathbf{M}[\downarrow n]$	A down shift of the matrix $\mathbf{M}$ by n positions, emptied positions after the shifts are filled by zeros.
$\mathbf{M}(j, :)[\rightarrow 1]$	A right shift of the j^{th} row of the matrix $\mathbf{M}$ by 1 position, emptied positions after the shifts are filled by zeros.
$\mathbf{v}[f_0, \downarrow 1]$	A down shift of the vector $\mathbf{v}$ by one-bit with cell f_0 fed in its upper-most bit, i.e., for the vector $\mathbf{v}$ of length l-bits $\mathbf{v}[f_0, \downarrow 1] = [f_0, \overbrace{0, \cdots, 0}^{l-1}]^T + \mathbf{v}[\downarrow 1]$.
$e_i \| \mathbf{v}^T$	The process of concatenating an element e_i and a vector $\mathbf{v}$.

2.2 Reduction Process

Let us first define an irreducible polynomial with ω non-zero terms, i.e., [19]

$$P(x) \triangleq x^m + \sum_{i=1}^{\omega-1} x^{t_i}, \tag{3}$$

where $\frac{m}{2} > t_1 > t_2 > \cdots > t_{\omega-2} > t_{\omega-1} = 0$. Then from (3), we define two new sets: $\mathcal{T}$ is a set of degrees of nonzero terms in (3), and $\mathcal{N}$ consists of $\omega - 1$ elements, which are the differences between m and the others contains the non-zero terms in (3), i.e.,

$$\mathcal{T} \triangleq \{0, t_1, \cdots, t_{\omega-2}\}, \text{ and } \mathcal{N} \triangleq \{0, \Delta_1, \cdots, \Delta_{\omega-2}\},$$

where $\Delta_1 = m - t_{\omega-2}$, $\Delta_2 = m - t_{\omega-3}$, $\cdots$, $\Delta_{\omega-2} = m - t_1$.

Note that the Mastrovito matrix $\mathbf{M}$, which is shown in (1) can be obtained by reducing the matrix $\mathbf{Z}$ in (2) using the generating polynomial (3). It is shown in [26], that the entries of the matrix $\mathbf{M}$ can be obtained as

$$\mathbf{M} = (\mathbf{L} + \mathbf{Q} \cdot \mathbf{U}), \tag{4}$$

where $\mathbf{L}$ is an $m \times m$ lower triangular Toeplitz matrix, which is defined as the first m rows of the matrix $\mathbf{Z}$; $\mathbf{U}$ is an $(m-1) \times m$ upper triangular Toeplitz matrix, which is defined as the last $(m-1)$ rows of $\mathbf{Z}$, i.e.,

$$\mathbf{L} \triangleq \begin{pmatrix} a_0 & 0 & 0 & 0 & \cdots & 0 \\ a_1 & a_0 & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \ddots & & \vdots \\ a_{m-2} & a_{m-3} & \cdots & a_1 & a_0 & 0 \\ a_{m-1} & a_{m-2} & \cdots & a_2 & a_1 & a_0 \end{pmatrix},$$
$$\mathbf{U} \triangleq \begin{pmatrix} 0 & a_{m-1} & a_{m-2} & \cdots & a_1 \\ 0 & 0 & a_{m-1} & \cdots & a_2 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & a_{m-1} & a_{m-2} \\ 0 & 0 & \cdots & 0 & a_{m-1} \end{pmatrix}, \tag{5}$$

and $\mathbf{Q}$ is a reduction matrix, which is formalized in [24], [26], and [25] as

$$\mathbf{Q} = \sum_{n \in \mathcal{N}} \hat{\mathbf{Q}}[\rightarrow n], \tag{6}$$

where

$$\hat{\mathbf{Q}} = \sum_{t \in \mathcal{T}} \mathbf{I}_{m \times (m-1)} [\downarrow t], \tag{7}$$

where $\mathbf{I}_{m \times (m-1)}$ represents an $m \times (m-1)$ identity matrix.

Then, using (6) and (7) the matrix $\mathbf{M}$ in (4) can be written as [24]

$$\mathbf{M} = \mathbf{L} + \mathbf{S} + \sum_{t \in \mathcal{T} - \{0\}} \mathbf{S}[\downarrow t], \tag{8}$$

where the matrix $\mathbf{S}$ is an $m \times m$ upper triangular Toeplitz matrix with the following form:

$$\mathbf{S} \triangleq \begin{pmatrix} 0 & s_{m-1} & s_{m-2} & \cdots & s_1 \\ 0 & 0 & s_{m-1} & \cdots & s_2 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 & s_{m-1} \\ 0 & 0 & \cdots & 0 & 0 \end{pmatrix}, \tag{9}$$

where the row 0 of $\mathbf{S}$, i.e., $\mathbf{S}(0, :)$ can be computed as [24]

$$\mathbf{S}(0, :) = [0, s_{m-1}, \cdots, s_1] = \sum_{n \in \mathcal{N}} \mathbf{U}(0, :)[\rightarrow n]. \tag{10}$$

3 Proposed Serial-out Bit-Level Mastrovito Multiplication Algorithm

From (4) and (8), one can define a matrix $\mathbf{P}$ as

$$\mathbf{P} = \mathbf{Q} \cdot \mathbf{U} = \mathbf{S} + \sum_{t \in \mathcal{T} - \{0\}} \mathbf{S}[\downarrow t]. \tag{11}$$

In (11), the rows produced due to the reductions corresponding to the x^{t_i} terms in (3) are identical to the rows produced at the first reduction iteration. Thus, we can store the elements of row $\mathbf{S}(0, :)$, so that they can be added later to obtain the rows t_i, $1 \leq i \leq \omega - 2$, of the matrix $\mathbf{P}$, i.e., $\mathbf{P}(t_i, :)$, for $t_i \in \mathcal{T} - \{0\}$. Then, the rows $\mathbf{P}(j, :)$, for $0 \leq j \leq m - 1$ can be obtained as

$$\mathbf{P}(j, :) = \begin{cases} \mathbf{S}(0, :), & \text{for } j = 0, \\ \mathbf{P}(j-1, :)[\rightarrow 1], & \text{for } 0 < j \ \& \ j \neq t_i, \\ \mathbf{P}(j-1, :)[\rightarrow 1] + \mathbf{S}(0, :), & \text{for } j = t_i, \end{cases} \tag{12}$$

for $1 \leq i \leq \omega - 2$.

From the Toeplitz matrix $\mathbf{L}$, which is shown in (5), one can see that the rows $\mathbf{L}(j, :)$, for $0 \leq j \leq m - 1$ can be obtained as

$$\mathbf{L}(j, :) = \begin{cases} [a_0, \underbrace{0, \cdots, 0}_{m-1}], & \text{for } j = 0, \\ \mathbf{L}(j-1, :)[a_j, \rightarrow 1], & \text{for } 0 < j \leq m-1. \end{cases} \tag{13}$$

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2015.2456023, IEEE Transactions on Computers

From (12) and (13), the row j of the matrix $\mathbf{M}$ in (4), i.e., $\mathbf{M}(j, :)$, for $0 \leq j \leq m-1$, is obtained as

$$\mathbf{M}(j, :) = \begin{cases} \mathbf{L}(0, :) + \mathbf{S}(0, :), & j = 0, \\ \mathbf{M}(j-1, :)[a_j, \rightarrow 1], & 0 < j \;\&\; j \neq t_i, \\ \mathbf{M}(j-1, :)[a_j, \rightarrow 1] + \mathbf{S}(0, :), & j = t_i, \end{cases} \quad (14)$$

for $1 \leq i \leq \omega - 2$.

From (10) and (13), one can see that the row 0 of the matrix $\mathbf{M}$ in (14) can be obtained as

$$\mathbf{M}(0, :) = \mathbf{L}(0, :) + \mathbf{S}(0, :) = [a_0, s_{m-1}, s_{m-2}, \cdots, s_1]. \quad (15)$$

After calculating $\mathbf{M}(j, :)$ and based on (1), one can serially obtain c_j, for $0 \leq j \leq m-1$ as

$$c_j = \mathbf{M}(j, :) \cdot \mathbf{b}. \quad (16)$$

3.1 Proposed SOBL Multiplication Algorithm for ω-nomials

From (10), (14), (15), and (16), we propose the following algorithm, which outlines the process of serially generating the coordinates of C starting from c_0 to ending c_{m-1} for the multiplication of the two field elements A and B.

Algorithm 1 Proposed Serial-Out Bit-Level Mastrovito Multiplier for ω-nomials $x^m + x^{t_1} + \cdots + x^{t_{\omega-2}} + 1$

Input : The parameters of the ω-nomial irreducible polynomial: $m, t_1, \cdots, t_{\omega-2}$, $A = (a_{m-1}, \cdots, a_0)$, $B = (b_{m-1}, \cdots, b_0) \in GF(2^m)$.
Output : c_j, where $C = (c_{m-1}, \cdots, c_0) = AB \mod P(\alpha)$.
/* *Set signal vectors* $\mathbf{s}^T$, $\mathbf{y}^T$, *and* $\mathbf{z}^T$ *of length* $m-1$, $m-1$, *and* m *bits, respectively* */
Initialize : $\mathbf{y}^T = [\, y_{m-2}, \cdots, y_0] = (a_{m-1}, \cdots, a_1)$;
$\mathbf{z}^T = [\, z_{m-1}, \cdots, z_0] = (b_{m-1}, \cdots, b_0)$;
$\mathbf{s}^T = [\, s_{m-1}, \cdots, s_1] = (a_{m-1}, \cdots, a_1)$.
/* *Compute* $\mathbf{s}^T$ = $\mathbf{S}(0, :)$ */
Step 1 : **For** $i = 1$ to $\omega - 2$ **do**
Step 1.1 : $\Delta_i = m - t_{\omega-1-i}$;
Step 1.2 : $\mathbf{s}^T = [s_{m-1}, \cdots, s_1] + [\overbrace{0, \cdots, 0}^{\Delta_i}, a_{m-1}, \cdots, a_{\Delta_i+1}]$;
Step 2 : **End For**
/* *Set a signal vector* $\mathbf{w}^T$ *of length* $m-1$ *bits, and initialize it with* $\mathbf{S}(0, :)$, *and set a signal vector* $\mathbf{x}^T$ *of length* m *bits, and initialize it with* $\mathbf{M}(0, :)$ */
Step 3 : $\mathbf{w}^T \leftarrow \mathbf{s}^T$; $\mathbf{x}^T \leftarrow a_0 \big|\big| \mathbf{s}^T$;
/* *Processes of the loop started in Step 4 are computed in parallel* */
Step 4 : **For** $j = 0$ to $m - 1$ **do**
/* *Compute the inner product :* $c_j = \mathbf{M}(j, :) \cdot \mathbf{b}$ */
Step 4.1 : Output $c_j = \mathbf{x}^T \bullet \mathbf{z}$;
/* *Update* $\mathbf{x}^T$ *with* $\mathbf{M}$(j+1, :) */
Step 4.2 : **If** $j \neq t_i - 1$ **Then**
/* $\mathbf{M}$(j+1, :)= $\mathbf{M}(j, :)[a_{j+1}, \rightarrow 1]$ */
Step 4.2.1 : $\mathbf{x}^T \leftarrow [y_0, x_{m-1}, \cdots, x_1]$;
Step 4.3 : **Else** /* $j = t_i - 1$ */
/* $\mathbf{M}$(j+1, :)= $\mathbf{M}(j, :)[a_{j+1}, \rightarrow 1] + \mathbf{S}(0, :)$ */
Step 4.3.1 : $\mathbf{x}^T \leftarrow [y_0, x_{m-1} + w_{m-2}, \cdots, x_1 + w_0]$;
Step 4.4 : **End If**
Step 4.5 : $\mathbf{y}^T \leftarrow [y_0, y_{m-2}, \cdots, y_1]$;
Step 5 : **End For**

Algorithm 1 is indeed a bit-level algorithmic version of the architecture of the parallel-level Mastrovito PB multiplier proposed in [24]. In Algorithm 1, the coordinates of the row vector $\mathbf{s}^T$ represent the entry of the first row of the matrix $\mathbf{S}$, i.e., $\mathbf{S}(0, :)$. These coordinates are obtained as presented in (10). From the Toeplitz matrix $\mathbf{S}$ shown in (9), one can see that the entry $\mathbf{S}(0 : m-1)$ is zero; hence, it is neglected in Algorithm 1. The row vector $\mathbf{s}^T$, is initialized with the coordinates from 1 to $m-1$ of the multiplicand A, i.e., $\mathbf{s}^T = [s_{m-1}, \cdots, s_1] = [a_{m-1}, \cdots, a_1]$. Then, the elements of $\mathbf{s}^T$ are accumulated in accordance with (10) to produce the desired $\mathbf{S}(0, :)$ after a total of $\omega - 2$ loop iterations. Hence, at each *for* loop iteration, i.e., in Step 1.2, coordinates from $\Delta_i + 1$ to $m - 1$, for $1 \leq i \leq \omega - 2$, of the multiplicand A are added with entries of the previous iteration's $\mathbf{s}^T$ vector.

The following lemma proves the correctness of vector $\mathbf{s}^T$ contents in Algorithm 1.

Lemma 1 *Let A be an arbitrary element in $GF(2^m)$ and $\mathbf{s}^T$ be a row vector of length $m-1$ that is initialized with the following entries $\mathbf{s}^T = [s_{m-1}, \cdots, s_1] = [a_{m-1}, \cdots, a_1]$. Then, the entries of the vector $\mathbf{s}^T$ at the end of the for loop at Step 1 of Algorithm 1 become $\mathbf{S}(0, :)$.*

Proof: Since the vector $\mathbf{s}^T$ is initialized with the row 0 of the matrix $\mathbf{U}$ in (5), the recursive call to the *for* loop in Step 1 accumulates $\mathbf{s}^T$ in accordance with $\mathbf{U}(0, :)[\rightarrow \Delta_i]$. Then, the final retuned vector (after a total of $\omega - 2$ loop iterations) satisfies $\mathbf{S}(0, :)$ as in (10). □

As shown in the initialization step, the coordinates of the multiplier B are stored in the row vector $\mathbf{z}^T$. Also the coordinates from 1 to $m-1$ of the multiplicand A are stored in the row vector $\mathbf{y}^T$, which will be used to obtain the rows j, for $1 \leq j \leq m-1$, of the matrix $\mathbf{L}$ as stated in (13). In Step 3, the operation $\mathbf{x}^T \leftarrow a_0 \big|\big| \mathbf{s}^T$, represents the concatenation of a_0 and the row vector $\mathbf{s}^T$; hence, $\mathbf{M}(0, :)$ that is shown in (15), is generated and stored in $\mathbf{x}^T$. The vector $\mathbf{s}^T$ is also stored in $\mathbf{w}^T$, in order to be added later for obtaining the rows $\mathbf{M}(t_i, :)$, $1 \leq i \leq \omega - 2$, as seen in (14).

The operation $\mathbf{x}^T \bullet \mathbf{z}$ in Step 4.1, represents the inner products of the coordinates of both the row vector $\mathbf{x}^T$ and the column vector $\mathbf{z}$, i.e., $\mathbf{x}^T \bullet \mathbf{z} = \sum_{i=0}^{m-1} x_i z_i$. It is noteworthy to mention that at the end of the iteration j of the loop started in Step 4, the output c_j is computed and at the same iteration the row $j+1$ of the matrix $\mathbf{M}$, i.e., $\mathbf{M}(j+1, :)$ would be generated and stored in $\mathbf{x}^T$. Hence, it would be ready for use in the next iteration. The following lemma proves that the contents of $\mathbf{x}^T$ at the end of j iteration become the row $\mathbf{M}(j+1, :)$ as seen in (14).

Lemma 2 *Let A be an arbitrary element in $GF(2^m)$, $\boldsymbol{y}^T$ be a row vector of length $m-1$ that is initialized with the following entries $\boldsymbol{y}^T = [y_{m-2}, \cdots, y_0] = [a_{m-1}, \cdots, a_1]$, $\boldsymbol{w}^T$ be a row vector of length $m-1$ that is initialized with $\mathbf{S}(0, :)$, and $\boldsymbol{x}^T$ be a row vector of length m that is initialized with row 0 of matrix $\boldsymbol{M}$. Then, the coordinates of $\boldsymbol{x}^T$ in the for loop at Step 4 of Algorithm 1 returns the correct value of the next row of the matrix $\boldsymbol{M}$ in (4).*

Proof: The *for* loop in Step 4 of Algorithm 1 has two

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2015.2456023, IEEE Transactions on Computers

conditional cases, for $j \neq t_i$, for this case, the *for* loop recursively computes

$$\mathbf{x}^T \leftarrow [y_0, x_{m-1}, \cdots, x_1], \quad \mathbf{y}^T \leftarrow [y_0, y_{m-2}, \cdots, y_1],$$

and for $j = t_i$, for this case, the *for* loop recursively computes

$$\mathbf{x}^T \leftarrow [y_0, x_{m-1} + w_{m-2}, \cdots, x_1 + w_0],$$
$$\mathbf{y}^T \leftarrow [y_0, y_{m-2}, \cdots, y_1],$$

by induction, each recursive call to the *for* loop in Step 4 of Algorithm 1, returns the next row of matrix $\mathbf{M}$ as in (14). □

The inner product generated in Step 4.1 and the bit additions of Step 4.3.1 can be performed independently and in parallel. Therefore, the computation time required for obtaining each bit of the output result (c_j), is proportional to the longest delay that is the delay of the inner product generated in Step 4.1.

4 Multiplier Architectures

In this section, an approach to the architecture design of the SOBL multiplier for both the ω-nomials and the irreducible trinomials is presented in detail. Both architectures are capable of generating an output bit with a total of one computational clock cycle. We remark that the bit-level structure multiplier is considered as an iterative architecture. Thus, for any bit-level (or digit-level) multiplier, a main control unit that generates a counter is required to generate the load, start, complete, and other control signals. In our approach, additional control signals are needed in computation of the multiplication product, which can also be generated from the main control unit. However, in order to provide a complete and in-depth view of the components involved in our approach, a binary counter that generates the necessary control signals for the computation of the multiplication product is included in our architecture. In our model, a series carry synchronous counter is used, which is implemented with a register for every bit and an AND gate for every bit except the first and last bit. The carry-in to carry-out delay in the series carry synchronous counter is $(\lceil \log_2 m \rceil - 2)T_A$, where T_A denotes the delay of the 2-input AND gate. We further remark that the loop iterations of the Algorithm 1 are mapped into hardware clock counter that are also denoted by j.

4.1 Multiplier Architecture for ω-nomials

The architecture for the ω-nomials (irreducible polynomials with ω non-zero terms) is depicted in Fig. 1(a). It is composed of circuits S and CSC, a binary counter, an $\mathbf{IP}_m$ block, and four registers $\langle W \rangle$, $\langle X \rangle$, $\langle Y \rangle$, and $\langle Z \rangle$ that are of length $m-1$, m, t_1, and m-bits, respectively. The circuit S maps the implementation of the loop started in Step 1 of Algorithm 1. The detailed implementation of S is shown in Fig. 2. In this figure, an oval-shape enclosure indicates a binary tree of XOR gates. It is noted that the output signal $\mathbf{s}$, which is generated by the circuit S, is equal to that of corresponding row 0 of the matrix $\mathbf{S}$, i.e., $\mathbf{S}(0, :)$. Let us consider the binary extension field $GF(2^{163})$ generated by the irreducible pentanomial $P(x) = x^{163} + x^7 + x^6 + x^3 + 1$. Given an arbitrary field element $A \in GF(2^{163})$, the coordinates of $\mathbf{s}$, are computed as

$$s_i = \begin{cases} a_i + a_{160+i} + a_{157+i} + a_{156+i}, & 1 \leq i \leq 2, \\ a_i + a_{157+i} + a_{156+i}, & 3 \leq i \leq 5, \\ a_i + a_{156+i}, & i = 6, \\ a_i, & 7 \leq i \leq 162, \end{cases} \tag{17}$$

for $i = 1, 2, \cdots 162$. Equation (17), can be realized by an architecture of 6 binary tree of the XOR gates. In general, the number of the XOR gates for computing $\mathbf{s}$, i.e., $\mathbf{s}_{\text{xor}}$ is

$$\mathbf{s}_{\text{xor}} = \sum_{i=1}^{\omega-2} (t_i - 1), \tag{18}$$

and the time delay of the longest path between the inputs and outputs ($\mathbf{s}_{\text{time}}$) is $\mathbf{s}_{\text{time}} = \lceil \log_2(\omega - 1) \rceil T_X$, where T_X denotes the delay of the 2-input XOR gate. As a result, the total XOR gates for this example becomes $\mathbf{s}_{\text{xor}} = 13$ and the delay becomes $\mathbf{s}_{\text{time}} = 2T_X$.

The register $\langle W \rangle$ is initialized with the contents of $\mathbf{s}$, i.e., $\langle w_{m-2}, \cdots, w_0 \rangle = [s_{m-1}, \cdots, s_1]$; hence, the operation $\mathbf{w}^T \leftarrow \mathbf{s}^T$, in Step 3 of Algorithm 1 is considered in this architecture. The output bits obtained from the circuit S, are concatenated with the element a_0, and the result is loaded to $\langle X \rangle$, i.e., $\langle x_{m-1}, \cdots, x_0 \rangle = [a_0, s_{m-1}, \cdots, s_1]$. This indicates that the operation $\mathbf{x}^T \leftarrow a_0 \| \mathbf{s}^T$, in Step 3 of Algorithm 1, is also presented in our architecture.

As also shown in the initialization step of Algorithm 1, the register $\langle Z \rangle$ is initialized with the coordinates of the multiplier B and its contents remain unchanged during each clock cycle until the end of multiplication process. Also, the coordinates from 1 to t_1 of the multiplicand A are initially fed into the register $\langle Y \rangle$, i.e., $\langle y_{t_1-1}, \cdots, y_1, y_0 \rangle = [a_{t_1}, \cdots, a_2, a_1]$.

It is worth noting that in this architecture, the row j, $0 < j \leq m-1$ & $j \neq t_i$, of the matrix $\mathbf{M}$ in (14) is obtained as

$$\mathbf{M}(j, :) = \begin{cases} \mathbf{M}(j-1, :)[y_0, \rightarrow 1], & \text{for } 0 < j \leq t_1 - 1, \\ \mathbf{M}(j-1, :)[w_{t_1}, \rightarrow 1], & \text{for } t_1 < j \leq m-1, \end{cases}$$

where y_0 and w_{t_1} are the coordinates of $\langle Y \rangle$ and $\langle W \rangle$ registers, respectively.

In TABLE 2, we show how the control signals Ctrl1 and Ctrl2 in Fig. 1(a) coordinate the contents of $\langle W \rangle$, $\langle X \rangle$, and $\langle Y \rangle$ registers. As shown in this table, if $j \leq t_1 - 1$, the contents of $\langle W \rangle$ remain unchanged, i.e., $\langle W \rangle = \mathbf{S}(0, :)$, whereas, the contents of $\langle Y \rangle$ are right cyclic shifted and hence, it maps the implementation of Step 4.5 of Algorithm 1. The contents of $\langle X \rangle$ during j, for $0 \leq j \leq t_1 - 1$ are updated as follows. If $j \neq t_i - 1$, then, $\langle X \rangle$ is updated by the right shift (RS) of its coordinates with $\langle y_0 \rangle$ fed at

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2015.2456023, IEEE Transactions on Computers

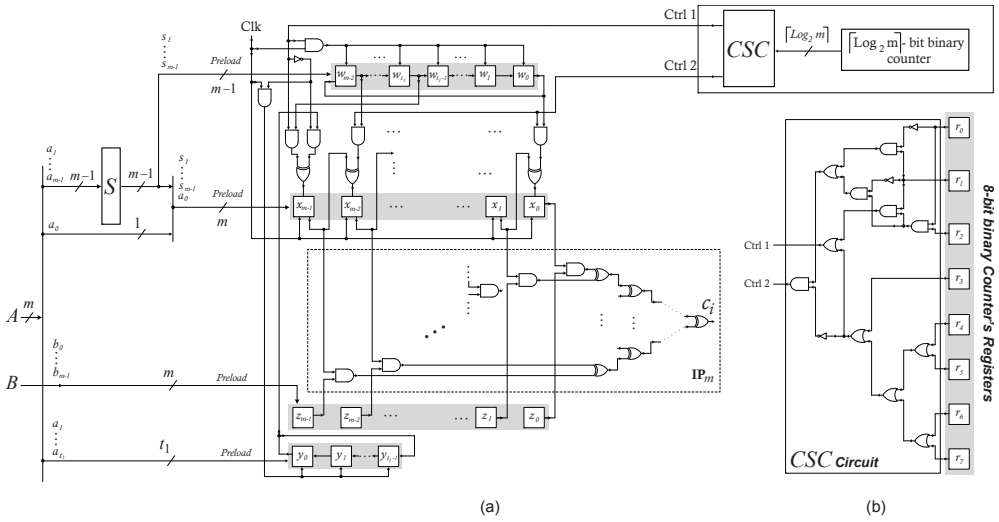

Fig. 1: The proposed serial-out bit-level (SOBL) Mastrovito multiplier architecture for the ω-nomial. (a) The high-level architecture. (b) The implementation of the control signal circuit (CSC) that generates the signals Ctrl1 and Ctrl2 from the 8-bit binary counter's registers for the $GF(2^{163})$ field constructed by $P(x) = x^{163} + x^7 + x^6 + x^3 + 1$.

TABLE 2: The operations of the control signals Ctrl1, and Ctrl2 in Fig. 1(a).

$j^{\dagger}$	Ctrl1	Ctrl2	$\langle W \rangle$	$\langle X \rangle$	$\langle Y \rangle$
$0 \leq j < t_1 - 1$ & $j \neq t_i - 1^{\dagger\dagger}$	0	0	clock is disabled	$\langle X \rangle = \langle y_0, x_{m-1}, \cdots, x_1 \rangle$	$\langle Y \rangle = \langle y_0, y_{t_1-1}, \cdots, y_1 \rangle$
$j = t_i - 1^{\dagger\dagger}$	0	1	clock is disabled	$\langle X \rangle = \langle y_0, x_{m-1} + w_{m-2}, \cdots, x_1 + w_0 \rangle$	$\langle Y \rangle = \langle y_0, y_{t_1-1}, \cdots, y_1 \rangle$
$t_1 - 1 < j \leq m - 1$	1	0	$\langle W \rangle = \langle w_0, w_{m-2}, \cdots, w_1 \rangle$	$\langle X \rangle = \langle w_{t_1}, x_{m-1}, \cdots, x_1 \rangle$	clock is disabled
$P(x) = x^{163} + x^7 + x^6 + x^3 + 1$					
$j = 0, 1, 3, 4$	0	0	clock is disabled	$\langle X \rangle = \langle y_0, x_{162}, \cdots, x_1 \rangle$	$\langle Y \rangle = \langle y_0, y_6, \cdots, y_1 \rangle$
$j = 2, 5, 6$	0	1	clock is disabled	$\langle X \rangle = \langle y_0, x_{162} + w_{161}, \cdots, x_1 + w_0 \rangle$	$\langle Y \rangle = \langle y_0, y_6, \cdots, y_1 \rangle$
$j = 7, 8, \cdots, 162$	1	0	$\langle W \rangle = \langle w_0, w_{161}, \cdots, w_1 \rangle$	$\langle X \rangle = \langle w_7, x_{162}, \cdots, x_1 \rangle$	clock is disabled

† j represents the hardware clock counter. †† For $1 \leq i \leq \omega - 2$.

the MSB. This maps the implementation of Step 4.2.1 of Algorithm 1. If $j = t_i - 1$ (t_i is obtained in (3)), then, $\langle X \rangle$ is updated by XORing the coordinates of $\langle W \rangle$ with the RS of its coordinates, and $\langle y_0 \rangle$ being fed into the MSB of $\langle X \rangle$. This maps the implementation of Step 4.3.1 of Algorithm 1. If $j > t_1 - 1$, observing this conditional case, one can see that the above mentioned condition, i.e., $j = t_i - 1$, will never occur again, hence, the contents of $\langle W \rangle$, i.e., $\mathbf{S}(0, :)$ are no longer needed. This gives us the freedom of using and changing the contents of $\langle W \rangle$. Hence, the contents of $\langle W \rangle$ are right cyclic shifted, i.e., $\langle w_{m-2}, \cdots, w_0 \rangle = \langle w_0, w_{m-2}, \cdots, w_1 \rangle$. The register $\langle X \rangle$ is then updated by the RS of its coordinates with $\langle w_{t_1} \rangle$ being fed into the MSB of $\langle X \rangle$. Fig. 1(b) illustrates the control signal circuit (CSC) that generates the signals Ctrl1 and Ctrl2 from the 8-bit binary counter's registers for the $GF(2^{163})$ field constructed by $P(x) = x^{163} + x^7 + x^6 + x^3 + 1$. From this figure, one can see that an additional cost of 6 OR gates, 5 AND gates, and 3 NOT gates (14 gates in total with an area complexity of 0.00925 KGate), is needed over the gate costs of a traditional 8-bit counter.

The module $\mathbf{IP}_m$ that is shown in Fig. 1(a), maps the implementation of the operation $c_j = \mathbf{x}^T \bullet \mathbf{z}$ in Step 4.1. This module, computes the output bit result $c_j = \mathbf{M}(j, :) \cdot \mathbf{b}$. It does so by performing the inner product (IP) of its two input vectors; it first generates the product in parallel using m AND gates and then, by adding (modulo 2) the generated partial products using a binary XOR tree. The architecture of the $\mathbf{IP}_m$ block implements

$$c_i = \sum_{i=0}^{m-1} x_i z_i = [x_0, \cdots, x_{m-1}] \times [z_0, \cdots, z_{m-1}]^T,$$

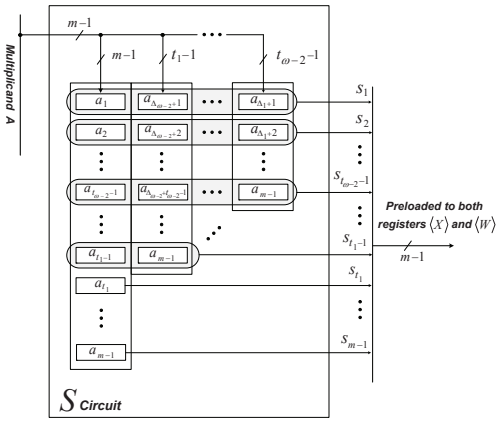

Fig. 2: The implementation of the circuit S in Fig. 1(a) that generates $\mathbf{S}(0, :)$.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2015.2456023, IEEE Transactions on Computers

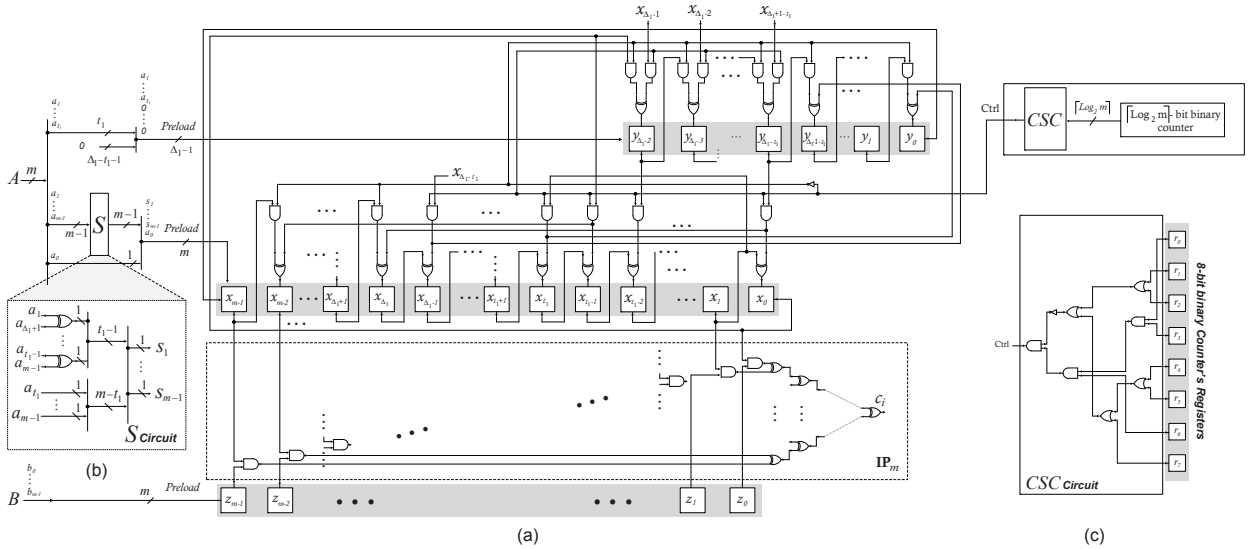

Fig. 3: The proposed Mastrovito serial-out bit-level (SOBL) multiplier architecture for the irreducible trinomial. (a) The high-level architecture. (b) The implementation of the circuit S. (c) The implementation of the control signal circuit (CSC) that generates the signal Ctrl from the 8-bit binary counter's registers for the $GF(2^{233})$ field constructed by $P(x) = x^{233} + x^{74} + 1$.

which requires $m-1$ XOR gates to accumulate the partial products. The depth of the binary XOR tree is given as $\lceil \log_2 m \rceil$ and, hence, the total delay of the $\mathbf{IP}_m$ module ($[\mathbf{IP}_m]_{\text{time}}$) is

$$[\mathbf{IP}_m]_{\text{time}} = T_A + \lceil \log_2 m \rceil T_X. \tag{19}$$

Proposition 1 *For the finite field $GF(2^{163})$ generated by the irreducible pentanomial $P(x) = x^{163} + x^7 + x^6 + x^3 + 1$, the proposed SOBL PB multiplier architecture (Fig. 1(a)) requires 503 1-bit registers, 333 2-input AND gates, 6 2-input OR gates, 4 NOT gates, and 336 2-input XOR gates.*

Proof: The number of 1-bit registers includes the ones in the $\langle X \rangle$ register, i.e., m, the register $\langle Z \rangle$, i.e., m, the register $\langle W \rangle$, i.e., $m-1$, the register $\langle Y \rangle$, i.e., t_1 and the register $\langle R \rangle$ in the binary counter, i.e., $\lceil \log_2 m \rceil$. Thus, the multiplier requires $3m + t_1 + 7 = 503$ 1-bit registers. The $\mathbf{IP}_m$ block requires m AND gates, a single AND gate for clock enabling the $\langle W \rangle$ register, $m+1$ AND gates for the connection between $\langle W \rangle$ and $\langle X \rangle$ registers and 5 AND gates for the CSC are also required. Therefore, the multiplier requires $2m + 7 = 333$ 2-input AND gates. The CSC circuit requires 6 OR gates and 3 NOT gates, a single NOT gate for complementing the signal Ctrl is also required. Therefore, the multiplier requires 6 OR gates and 4 NOT gates. The number of the XOR gates is obtained by adding those for the $\mathbf{IP}_m$, the updating signal for the register $\langle X \rangle$, as well as the S circuit, which are $m-1$, m, and (18), respectively. As a result, the number of the XOR gates required in the SOBL multiplier architecture generated by $P(x) = x^{163} + x^7 + x^6 + x^3 + 1$ is $2m - 2 + \sum_{i=1}^{\omega-2}(t_i - 1) = 336$ and the proof is complete. □

4.2 Multiplier Architecture for Trinomials

The proposed SOBL multiplier architecture that is illustrated in Fig. 1(a), can be further optimized for the irreducible trinomial, which is a special case of (3), i.e., $P(x) \triangleq x^m + x^{t_1} + 1$. The sets $\mathcal{T}$ and $\mathcal{N}$ for the irreducible trinomial, have $\{0, t_1\}$ and $\{0, \Delta_1 = m - t_1\}$ sets, respectively. This optimization can be achieved as shown in Fig. 3(a).

The architecture in this figure, is composed of circuits S and CSC, a binary counter, an $\mathbf{IP}_m$ block, and three registers $\langle X \rangle$, $\langle Y \rangle$, and $\langle Z \rangle$. The register $\langle Y \rangle$ in this figure, is reduced to $\Delta_1 - 1$ bits. Initially, the coordinates from 1 to t_1 of the multiplicand A are fed into $\langle Y \rangle$ in the locations from 0 to $t_1 - 1$, i.e., $\langle y_{t_1-1}, \cdots, y_0 \rangle = [a_{t_1}, \cdots, a_1]$. The contents of $\langle Y \rangle$ are postponed by $m - 2t_1 - 1$, zeros (cleared) at its left-most $m - 2t_1 - 1$ bits, i.e., $\langle y_{\Delta_1-2}, \cdots, y_{t_1} \rangle = [\underbrace{0, 0, \cdots, 0}_{m-2t_1-1}]$.

The register $\langle Z \rangle$, and the module $\mathbf{IP}_m$ remain unchanged as in the proposed ω-nomial SOBL architecture, which is presented in Subsection 4.1 (Fig. 1(a)). The S circuit is implemented as shown in Fig. 3(b). As seen in this figure, it is composed of $t_1 - 1$ parallel XORs. The output bits obtained from the circuit S, are concatenated with the element a_0. This concatenation result is loaded to $\langle X \rangle$, i.e., $\langle x_{m-1}, \cdots, x_0 \rangle = [a_0, s_{m-1}, \cdots, s_1]$. During both clock periods $0 \le j \le t_1 - 2$ and $t_1 \le j \le m - 1$, the contents of both registers $\langle X \rangle$ and $\langle Y \rangle$ are right shifted. The right-most bit (LSB) of $\langle X \rangle$ is fed into the MSB of the register $\langle Y \rangle$, i.e., $\langle y_{\Delta_1-2} \rangle \leftarrow \langle x_0 \rangle$, and similarly, the LSB of $\langle Y \rangle$ is fed into the MSB of $\langle X \rangle$, i.e., $\langle x_{m-1} \rangle \leftarrow \langle y_0 \rangle$.

At the clock cycle $t_1 - 1$, both registers $\langle X \rangle$ and $\langle Y \rangle$ are updated with the proper contents as described in the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2015.2456023, IEEE Transactions on Computers

following:

$$\langle x_{t_1-2}, \cdots, x_0\rangle \leftarrow \langle x_{t_1-1}+y_{\Delta_1-2}, \cdots, x_1+y_{\Delta_1-t_1}\rangle,$$
$$\langle x_{m-1}, \cdots, x_{t_1-1}\rangle \leftarrow \langle y_0, y_{\Delta_2}, \cdots, y_{\Delta_1-t_1}, x_{\Delta_1-t_1}+x_{\Delta_1}, \cdots, x_0+x_{t_1}\rangle,$$
$$\langle y_{\Delta_1-1}, \cdots, y_0\rangle \leftarrow \langle x_{\Delta_1-1}, \cdots, x_1\rangle.$$

Fig. 3(c) illustrates the control signal circuit (CSC) that generates the signal Ctrl from the 8-bit binary counter's registers for the $GF(2^{233})$ field constructed by $P(x) = x^{233} + x^{74} + 1$. From this figure, one can see that an additional cost of 4 OR gates, 3 AND gates, and 1 NOT gate (8 gates in total with an area complexity of 0.00525 KGate), is needed over the gate costs of a traditional 8-bit binary counter.

Proposition 2 *For the finite field $GF(2^{233})$ generated by the irreducible trinomial $x^{233} + x^{74} + 1$, the proposed SOBL PB multiplier architecture (Fig. 3(a)) requires 632 1-bit registers, 699 2-input AND gates, 4 2-input OR gates, 2 NOT gates, and 696 2-input XOR gates.*

Proof: The number of 1-bit registers includes the ones in the $\langle X\rangle$ register, i.e., m, the register $\langle Z\rangle$, i.e., m, the register $\langle Y\rangle$, i.e., $\Delta_1 - 1 = m - t_1 - 1$ and the register $\langle R\rangle$ in the binary counter, i.e., $\lceil \log_2 m \rceil$. Thus, the multiplier requires $3m - t_1 + 7 = 632$ 1-bit registers. The $\mathbf{IP}_m$ block requires m AND gates, $2m - 3$ AND gates for the connection between $\langle X\rangle$ and $\langle Y\rangle$ registers and 3 AND gates for the CSC circuit are also required. Therefore, the multiplier requires $3m = 699$ 2-input AND gates. The CSC circuit requires 4 OR gates and a single NOT gate, a single NOT gate for complementing the signal Ctrl is also required. Therefore, the multiplier requires 4 OR gates and 2 NOT gates. The number of the XOR gates is obtained by adding those for the $\mathbf{IP}_m$, the updating signals for $\langle X\rangle$ and $\langle Y\rangle$, as well as the S circuit, which are $m - 1$, $m - 1$, Δ_1, and $t_1 - 1$, respectively. As a result, the number of the XOR gates required in the SOBL multiplier architecture generated by the irreducible trinomial $x^{233}+x^{74}+1$ is $3m-3 = 696$ and the proof is complete. □

The critical path delay, which is the longest path from the registers to the output c_i, is one of the main factors that determines the time complexity. It determines the maximum operating frequency. By properly implementing the proposed SOBL architectures, i.e., Fig. 1(a) and Fig. 3(a), one can see that the critical path delay of both architectures is equal to the total delay of the $\mathbf{IP}_m$ module, which is shown in (19).

5 Architectures for Double Multiplication

In this section, we first extend the traditional parallel-out bit-level (POBL) multiplier schemes presented in [16] to propose new POBL double multiplication architectures. We then, propose new hybrid-double multiplication architectures using PB over $GF(2^m)$. Note that all the presented architectures can be easily modified to extend their structure into the digit-level. However, for the sake of simplicity, in this work we did not investigate on the techniques for the digit-level structures.

5.1 New Architectures for LSB-first/MSB-first POBL Double Multiplications

Beth and Gollman in [16] proposed two types of bit-level multiplier schemes, namely LSB-first and MSB-first, multipliers. Let A and B be two arbitrary elements of $GF(2^m)$ and C be their multiplication, i.e., $C = AB$. Then, the LSB-first POBL multiplier is obtained as follows [16]

$$C = b_{m-1}\big((A\alpha^{m-1}) \bmod P(\alpha)\big) + \cdots + b_0\big(A \bmod P(\alpha)\big),$$

and the MSB-first POBL multiplier is obtained as follows

$$C = \Bigg(\cdots\Big((b_{m-1}A)\alpha \bmod P(\alpha) + b_{m-2}A\Big)\alpha \bmod P(\alpha) + \cdots + b_1A\Bigg)\alpha \bmod P(\alpha) + b_0A.$$

Let D and $E \in GF(2^m)$ such that $E = CD \bmod P(\alpha)$. A combination of two consecutive single multiplications $C = AB$, and $E = CD$ produces the following double multiplication involving three operands:

$$E = ABD. \tag{20}$$

A double multiplier that computes (20) can be achieved by extending the schemes of the traditional POBL to the schemes presented in Figs. 4(a) and 4(b). In these figures, the register $\langle Y\rangle$ is initialized as follows, for the LSB-first double multiplier, i.e., Fig. 4(a), $\langle y_{2m-1}, \cdots, y_m\rangle = D$, and $\langle y_{m-1}, \cdots, y_0\rangle = A$, and for the MSB-first double multiplier, i.e., Fig. 4(b), $\langle y_{2m-1}, \cdots, y_m\rangle = A$, and $\langle y_{m-1}, \cdots, y_0\rangle = D$. In both architectures, the register $\langle X\rangle$ is initialized with B and the register $\langle Z\rangle$ is initially cleared. Also, the α module multiplies the input by α and reduces the results by $P(x)$. This is done at cost of $\omega - 2$ 2-input XOR gates. The dotted block, i.e., $\bigodot$, in both figures, denotes bit-wise AND operation between the LSB (or MSB) bit of $\langle Y\rangle$ and the contents of $\langle X\rangle$ and is performed using m 2-input AND gates. The adder block, i.e., $\bigoplus$, denotes bit-wise XOR gates and is implemented using m 2-input XOR gates. After m clock cycles, the contents of $\langle Z\rangle$ that become the coordinates of the product $C = AB$, are loaded to $\langle X\rangle$. Eventually, at clock $2m$, the contents of $\langle Z\rangle$ become the coordinates of the product $E = CD$.

The MSB-first double multiplier scheme shown in Fig. 4(a) as compared to the LSB-first double multiplier scheme shown in Fig. 4(b), has longer critical path delay. Since in the MSB-first double multiplier scheme, the α module must also be considered in the delay path. However, the hardware overhead gates due to the parallel I/O data transfer to $\langle X\rangle$ register in the LSB-first double multiplier requires a 3-to-1 multiplexer of size m bits. As

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2015.2456023, IEEE Transactions on Computers

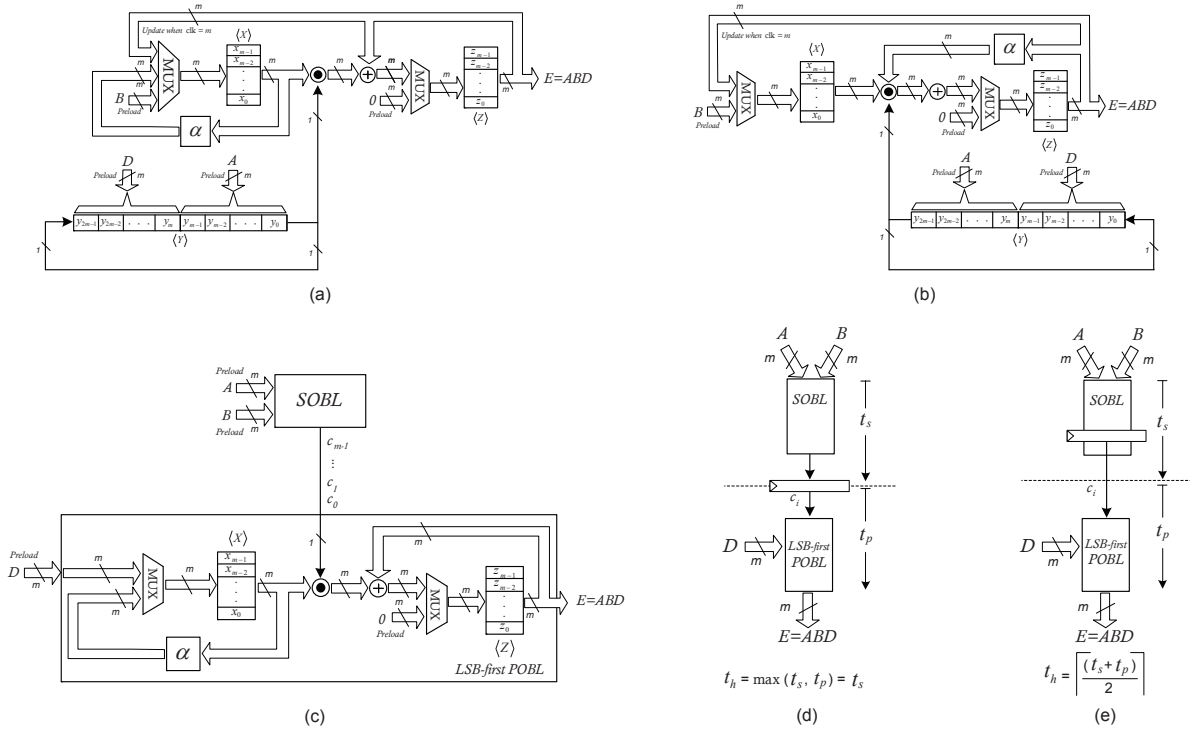

Fig. 4: The proposed double-multiplication architectures. (a) The proposed LSB-first POBL double multiplication architecture that extends the POBL schemes presented in [16]. (b) The proposed MSB-first POBL double multiplication architecture that extends the POBL schemes presented in [16]. (c) The hybrid-double multiplication structure is developed by connecting the output of the SOBL multiplier into the input of the POBL multiplier. (d) The critical-path delay of the hybrid-double multiplication (t_h). (e) Reducing the delay by inserting registers at the $\mathbf{IP}_m$ block inside the SOBL multiplier.

a result, the LSB-first double multiplier has higher area complexity.

5.2 Hybrid-Double Multiplication

Recently, hybrid-double multiplier was proposed in $GF(2^m)$ using normal basis representation [17], [18]. This hybrid-double multiplier is achieved by combining and interleaving a SOBL Gaussian normal basis multiplier that is implemented based on [27], and a POBL normal bases multiplier that is based on [16]. Note that a traditional POBL multiplier such as Beth and Gollmann approach [16] by itself cannot create a hybrid-double multiplier component; however, combining a SOBL multiplier with a traditional POBL one would allow to develop a hybrid-double multiplier.

A multiplier operates using the PB representation, in compared to the normal bases, has lower hardware requirements and easy-to-derive structure based on the defining irreducible polynomial for the field $P(x)$ [30]. In the following we employ the proposed two SOBL schemes, and the SOBL scheme proposed in [19], to present, for the first time, hybrid-double multiplication architectures using PB over $GF(2^m)$.

The SOBL polynomial basis multiplication scheme proposed in [19] generates every bit of the multiplication in each clock cycle. Thus, it can be combined with the traditional POBL multiplier (such as Beth and Gollmann approach in [16]) to produce the hybrid-double multiplication scheme. The structure of the hybrid-double multiplication is illustrated in Fig. 4(c). In this figure, the SOBL multiplier generates every bit of the multiplication, i.e., the output bit result of the product $C = AB$, in each clock cycle, whereas the POBL multiplier computes all output coordinates in parallel after m clock cycles. As one can see from Fig. 4(c), all bits of the operands A, B, and D are initially available, while the coordinates of the partial product C should be available in serial fashion starting from the LSB, i.e., c_0.

The structure of the hybrid-double multiplication as illustrated in Fig. 4(c), allows performing two multiplications simultaneously, where the results are available in parallel after $m + 1$ clock cycles assuming that one clock cycle is required to load the output of the SOBL multiplier (stored in the register) to the input of the LSB-first SOBL multiplier. The critical path delay of the hybrid-double multiplication (t_h) is equal to the maximum of delays between the LSB-first POBL (t_s) and the SOBL (t_p) multipliers, i.e., $t_h = \max\{t_s, t_p\}$. Based on the information provided in TABLE. 3, i.e., $t_s > t_p$, one can see that $t_h = t_s$. Thus, to speed up the multiplication, one can balance the latency of the two multipliers at the cost of a few additional registers. Let us divide the $\mathbf{IP}_m$ block by inserting registers at stage ε, then, the total

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2015.2456023, IEEE Transactions on Computers

TABLE 3: Comparison of the Proposed SOBL Multipliers (Fig. 1(a) and Fig. 3(a)) in Terms of Times Complexities for the Irreducible ω-nomial and the Irreducible Trinomial.

Type of Multiplier Scheme	Output Structure	Latency [cycle]		Critical Path Delay †
		Bit-Latency	Total-Latency	
$P(x) = x^m + \sum_{i=1}^{\omega-1} x^{t_i}$, $\frac{m}{2} > t_1 > t_2 > \cdots > t_{\omega-2} > t_{\omega-1} = 0$				
LSB-first [16]	Parallel	m	m	$T_A + T_X$
MSB-first [16]	Parallel	m	m	$T_A + T_X$
SOBL [28] ††	Serial	m	$2m$	$T_A + \lceil \log_2(m-1) \rceil T_X$
SOBL [19] †††	Serial	1	m	$T_A + \max(T_1, T_2)$
Proposed SOBL Fig. 1(a)	Serial	1	m	$T_A + \lceil \log_2 m \rceil T_X$
$P(x) = x^{163} + x^7 + x^6 + x^3 + 1$				
LSB-first [16]	Parallel	163	163	$T_A + T_X$
MSB-first [16]	Parallel	163	163	$T_A + T_X$
SOBL [28]	Serial	163	326	$T_A + 8\ T_X$
SOBL [19]	Serial	1	163	$T_A + 11\ T_X$
Proposed SOBL Fig. 1(a)	Serial	1	163	$T_A + 8\ T_X$
$P(x) = x^m + x^{t_1} + 1$, and $1 \leq t_1 < \frac{m}{2}$				
LSB-first [16]	Parallel	m	m	$T_A + T_X$
MSB-first [16]	Parallel	m	m	$T_A + T_X$
SOBL [28]	Serial	m	$2m$	$T_A + \lceil \log_2(m-1) \rceil T_X$
SOBL [19]	Serial	1	m	$T_A + (2 + \lceil \log_2 m \rceil) T_X$
Proposed SOBL Fig. 3(a)	Serial	1	m	$T_A + \lceil \log_2 m \rceil T_X$
$P(x) = x^{233} + x^{74} + 1$				
LSB-first [16]	Parallel	233	233	$T_A + T_X$
MSB-first [16]	Parallel	233	233	$T_A + T_X$
SOBL [28]	Serial	233	466	$T_A + 8\ T_X$
SOBL [19]	Serial	1	233	$T_A + 10\ T_X$
Proposed SOBL Fig. 3(a)	Serial	1	233	$T_A + 8\ T_X$

† The critical path delay of the multiplier schemes is obtained in terms of the delay of two-input XOR gate (T_X) and the delay of two-input AND gate (T_A).

†† The complexity results of [28] are obtained from [29].

††† $T_1 = (1 + \lceil \log_2(\omega - 1) \rceil + \lceil \log_2(m) \rceil) T_X$, $T_2 = (1 + \lceil \log_2(m-1) \rceil + \lceil \log_2(\omega - 2) \rceil) T_X$.

number of required registers υ is $\upsilon = \left\lceil \frac{m}{2^\varepsilon} \right\rceil$ register bits. It is noted that, if the position of ε were to be properly chosen, then, the total propagation delay of the hybrid-double multiplication architecture, as depicted in Fig. 4(e), would be reduced to about $\left\lceil \frac{t_s + t_p}{2} \right\rceil$.

6 Comparison

Let us define bit-latency and total-latency as the number of clock cycles needed for the first bit of the output to be available, and for the entire multiplication, respectively. Thus, one can see that the bit-latency of the proposed SOBL multipliers is one, and that the total-latency requires m clock cycles.

TABLE 3 and TABLE 4 show the comparison of the proposed SOBL multiplier with other efficient POBL and SOBL multipliers in terms of area and time complexities for the irreducible ω-nomials and the trinomials. It can be seen from both tables that the complexity of the SOBL multiplier schemes are higher than that using POBL multiplier schemes. However, in many applications such as the hybrid-double multiplication architecture a SOBL multiplier would be desirable because of its ability to sequentially generate an output bit of the final multiplication result in each clock cycle with the latency of one cycle. TABLE 3 also shows that in terms of delay complexities, the proposed two SOBL multiplier schemes, i.e., Fig. 1(a) and Fig. 3(a), outperform the previous published SOBL ones. As an example, for the binary extension fields $GF(2^{163})$ and $GF(2^{233})$ that are recommended by NIST [31] and SECG [32], the critical path delay of the SOBL multiplier that is proposed in [19] over those two finite fields are $T_A + 11T_X$, and $T_A + 10T_X$, respectively. whereas in proposed two SOBL multiplier schemes, the critical path delays over both finite fields are $T_A + 8T_X$.

In addition to the core multiplier component, the bit-level multiplier processor has to embed some other functionality to operate properly. For instance, a controller component that allows controlling the I/O communication signals, and generates the control signals is required. Also, to minimize the total latency, the data I/O has to be transferred in parallel (at cost of 1 clock cycle). The parallel I/O overhead (time and extra hardware) cannot be considered negligible. Figs. 5(a) and 5(b), illustrate the hardware overhead gates due to the parallel I/O data transfer. The circuit that is depicted in Fig. 5(a) enables a bit register to be initially cleared (when *load* signal $= 1$) or updated with the *update* signal (when *load* signal $= 0$). The circuit in Fig. 5(b) enables a bit register to switch between two inputs based on the *load* signal. Note that no extra gate is required when a bit register hold the same data as at the initialization (as required in the $\langle Z \rangle$ register in both Fig. 1(a), and Fig. 3(a)). The corresponding loading overhead gates in the proposed

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2015.2456023, IEEE Transactions on Computers

TABLE 4: Comparison of the Proposed SOBL Multipliers (Fig. 1(a) and Fig. 3(a)) in Terms of Space Complexities for the Irreducible ω-nomial and the Irreducible Trinomial.

Type of Multiplier Scheme	Area Cost			
	Total AND Gates	Total XOR Gates	Total 1-bit Reg.	Additional Costs
$P(x) = x^m + \sum_{i=1}^{\omega-1} x^{t_i}$, $\frac{m}{2} > t_1 > t_2 > \cdots > t_{\omega-2} > t_{\omega-1} = 0$				
LSB-first [16]	m	$m+\omega-2$	$3m$	–
MSB-first [16]	m	$m+\omega-2$	$3m$	–
SOBL [28] †	$3m-1$	$3m-2$	$4m+1$	–
SOBL [19] ††	$2m-1$	$2m+\omega+\gamma-4$	$3m+t_1-1$	–
Proposed SOBL Fig. 1(a)	$2m+2$	$2m+\gamma-2$	$3m+t_1-1$	$\lceil \log_2 m \rceil$–binary counter and CSC circuit †††
$P(x) = x^{163} + x^7 + x^6 + x^3 + 1$				
LSB-first [16]	163	166	489	–
MSB-first [16]	163	166	489	–
SOBL [28]	488	487	653	–
SOBL [19]	325	340	495	–
Proposed SOBL Fig. 1(a)	333	336	503	6 OR gates and 4 NOT gates
$P(x) = x^m + x^{t_1} + 1$, and $1 \leq t_1 < \frac{m}{2}$				
LSB-first [16]	m	$m+1$	$3m$	–
MSB-first [16]	m	$m+1$	$3m$	–
SOBL [28]	$3m-1$	$3m-2$	$4m+1$	–
SOBL [19]	$2m-1$	$2m+t_1-2$	$3m+t_1-1$	–
Proposed SOBL Fig. 3(a)	$3m-3$	$3m-3$	$3m-t_1-1$	$\lceil \log_2 m \rceil$–binary counter and CSC circuit †††
$P(x) = x^{233} + x^{74} + 1$				
LSB-first [16]	233	234	699	–
MSB-first [16]	233	234	699	–
SOBL [28]	698	697	933	–
SOBL [19]	465	538	772	–
Proposed SOBL Fig. 3(a)	699	696	632	4 OR gates and 2 NOT gates

† The complexity results of [28] are obtained from [29].
†† $\gamma = \sum_{i=1}^{\omega-2}(t_i - 1)$.
††† The complexity of the binary counter can be ignored by using the counter of the main control unit.

multiplier schemes are provided in TABLE 5. In this table, we compare the proposed multiplier schemes with the related bit-level multipliers when having the same parallel I/O communication format.

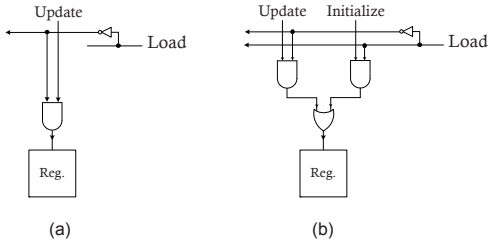

Fig. 5: Hardware overhead gates due to the parallel I/O data transfer. (a) The circuit that enables a register to be cleared or updated. (b) The circuit that enables a register to be switched between two inputs (MUX).

7 ASIC Implementation

In this section, We implement the presented schemes in the previous sections and the counterpart ones (10 schemes in total) to evaluate their area, time, and power requirements. For each scheme, we have two implementations, one with basic controller, and one with considering the full controllers that initialize and terminate the computation as part of the multiplier scheme (a complete serial-multiplier circuit). The proposed multiplier schemes are modeled in VHDL and synthesized for the binary extension fields $GF(2^{163})$ and $GF(2^{233})$ that are recommended by NIST and SECG. The 65-nm Complementary Metal-Oxide-Semiconductor (CMOS) library has been chosen for the synthesis on the ASIC technology. All architectures have been synthesized using Synopsys® Design Vision® which is a GUI for Synopsys® Design Compiler® tools [33]. The correctness of the architectures is verified by Xilinx® ISE™ Simulator (ISim).

The same default configurations have been used for each synthesis approach, i.e., the same supply voltage, test-bench, etc. The map effort for optimizations is set to medium (i.e., default). The power consumption readings have been conducted under 666 MHz frequency for all designs. The fast bit-level multipliers described in [16] and [19] are also modeled in VHDL and synthesized in the same framework as the proposed multipliers to facilitate quantitative performance comparison. We note that the power compiler in Synopsys® Design Compiler® tools uses the power characterization specified in the target library and switching activity to estimate power dissipation [33]. For each multiplier scheme, the area complexities are normalized to the complexity of a two-input NAND gate. It is noted that the area of a NAND gate in the utilized CMOS library for the drive strength of two is 2.08 μm^2. The total area is the sum of the combinational area (CA) and the non-combinational area (Non-CA). The timing (ns) for the critical-path delays

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2015.2456023, IEEE Transactions on Computers

TABLE 5: Comparison of the Proposed Multiplier Schemes (Fig. 1(a) and Fig. 3(a)) with the Related Bit-Level Multiplier Schemes when having the same Parallel I/O Data Transfer Format.

Type of Multiplier Scheme	Total Reg. [bit]	Never Changed Reg. † [bit]	Initially Cleared Reg. †† [bit]	Loaded and Updated Reg. ††† [bit]	Total Parallel I/O Hardware Overhead	
					Total AND Gates	Total OR Gates
LSB-first [16]	$3m$	–	m	$2m$	$5m$	$2m$
MSB-first [16]	$3m$	m	m	m	$3m$	m
SOBL [19]	$3m+t_1-1$	m	$m+t_1-1$	m	$3m+t_1-1$	m
Proposed SOBL Fig. 1(a)	$3m+t_1-1$	m	–	$2m+t_1-1$	$4m+2t_1-2$	$2m+t_1-1$
Proposed SOBL Fig. 3(a)	$3m-t_1-1$	m	$m-2t_1-1$	$m+t_1$	$3m-1$	$m+t_1$
$P(x)=x^{163}+x^7+x^6+x^3+1$						
LSB-first [16]	489	–	163	326	815	326
MSB-first [16]	489	163	163	163	489	163
SOBL [19]	495	163	169	163	495	163
Proposed SOBL Fig. 1(a)	495	163	–	332	664	332
$P(x)=x^{233}+x^{74}+1$						
LSB-first [16]	699	–	233	466	1165	466
MSB-first [16]	699	233	233	233	699	233
SOBL [19]	772	233	306	233	772	233
Proposed SOBL Fig. 1(a)	772	233	–	539	1078	539
Proposed SOBL Fig. 3(a)	624	233	84	307	698	307

† Bit registers with free I/O data transfer.
†† Bit registers with a single AND gate for the I/O data transfer.
††† Bit registers with a multiplexer for the I/O data transfer.

(CPD) and the dynamic power (mW) are also obtained for all the designs. The reported ASIC results of the implementations of the multipliers over $GF(2^{163})$ and $GF(2^{233})$ are listed in TABLE 6. In this table, the total time required for each multiplier is computed by multiplying the number of clock cycles, i.e., m, by the critical-path delay. It can be seen from the table that for the POBL schemes, the computation time required to obtain the first output bit and the total time required for the multiplication are equal, whereas, in the SOBL schemes, the computation time required to obtain the first output bit is equal to the critical-path delay. Also the controller has longer critical-path delay than the delay of the actual POBL schemes (the core multiplier component). From the table, one can see that the area complexity of the proposed ω-nomial SOBL scheme that is depicted in Fig. 1(a), i.e., the one that uses Ctrl1 and Ctrl2 signals, is increased around 8-11% as compared to the one proposed in [19], while the critical-path delay is decreased by 14% w.r.t the one in [19]. Also from this table, one can see that the proposed trinomial SOBL scheme that is depicted in Fig. 2(a) has lower time and area complexity as compared to the one in [19]. Further, when considering the controllers as part of the multiplier in the finite field over $GF(2^{233})$, the SOBL multipliers are the most dynamic power efficient schemes.

Also, the proposed double multiplication architectures are implemented and the area, time, and power consumption are reported for both $GF(2^{163})$ and $GF(2^{233})$ in TABLE 7. In this table, the total time of the multiplication is computed as follows. For the POBL double-multiplication architectures, we multiply the total number of clock cycles, i.e., $2m$, by the critical-path delay. For the hybrid-double multiplication architectures, we multiply the total number of clock cycles, i.e., $m+1$, by the critical-path delay. Also, for the POBL double-multiplication architectures, the throughput (TPT) of the multiplication is obtained by multiplying the number of bits per cycle, i.e. $\frac{m}{2m}$, by the speed, whereas, the TPT in the hybrid-double multiplication architectures, is obtained by multiplying the number of bits per cycle, i.e. $\frac{m}{m+1}$, by the speed. It is shown in TABLE 7, that by employing the proposed SOBL schemes in the hybrid-double multiplication architectures, the total time complexity reduces, and the throughput improves, w.r.t. the other double multiplication architectures.

8 Conclusions

We have presented new hardware schemes for the serial-out bit-level (SOBL) multiplier in PB representation over $GF(2^m)$ for both the ω-nomial and the irreducible trinomial. Compared to previously published results in terms of time complexities, the work presented here outperform the existing SOBL multiplier schemes. We have also extended the traditional POBL multiplier schemes to new POBL double multiplication architectures, which perform two multiplications after $2m$ clock cycles. Then, we proposed three hybrid-double multiplication architectures in PB over $GF(2^m)$. These hybrid multiplier structures perform two multiplications with latency comparable to the latency of a single multiplication, i.e., after $m+1$ clock cycles. We have obtained the space and time complexities of the presented multipliers and have compared them with their counterparts. For the practical purposes, all the 10 schemes presented in this work have been implemented in ASIC technology over both $GF(2^{163})$ and $GF(2^{233})$, and the area, timing, power consumption, and energy results have been presented.

Acknowledgments

The authors would like to thank the anonymous reviewers for their constructive comments. The work of A. Reyhani-Masoleh was supported by the Natural Sciences

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2015.2456023, IEEE Transactions on Computers

TABLE 6: Comparison of Bit-Level polynomial basis multipliers on an ASIC implementation (post synthesis) over both $GF(2^{163})$ and $GF(2^{233})$ using 65-nm CMOS standard technology.

Type of Multiplier	Type of Scheme	Area [KGate] †			CPD $[ns]$	Speed $[MHz]$	Bit-Time $[ns]$	Total-Time $[ns]$	Dynamic Power $[mW]$ ††
		CA	Non-CA	Total					
$P(x) = x^{163} + x^7 + x^6 + x^3 + 1$ (Without the main controller)									
LSB-first [16]	POBL	1.49	1.84	3.33	0.3	3333	48.9	48.9	6.653
MSB-first [16]	POBL	1.16	1.84	3	0.32	3125	52.16	52.16	5.76
SOBL [19]	SOBL	1.63	1.9	3.53	0.86	1162	0.86	140.18	4.996
Proposed Fig. 1(a)	SOBL	1.99	1.96	3.95	0.75	1333	0.75	122.25	6.338
$P(x) = x^{163} + x^7 + x^6 + x^3 + 1$ (With the main controller)									
LSB-first [16]	POBL	1.58	1.89	3.47	0.41	2439	66.83	66.83	6.748
MSB-first [16]	POBL	1.23	1.89	3.12	0.43	2325	70.09	70.09	5.816
SOBL [19]	SOBL	1.67	1.96	3.63	0.86	1162	0.86	140.18	5.168
Proposed Fig. 1(a)	SOBL	1.99	1.96	3.95	0.75	1333	0.75	122.25	6.338
$P(x) = x^{233} + x^{74} + 1$ (Without the main controller)									
LSB-first [16]	POBL	2.11	2.62	4.73	0.31	3225	72.23	72.23	9.498
MSB-first [16]	POBL	1.65	2.62	4.27	0.32	3125	74.56	74.56	8.108
SOBL [19]	SOBL	2.55	2.95	5.5	0.83	1204	0.83	193.39	7.848
Proposed Fig. 1(a)	SOBL	2.77	3.01	5.78	0.74	1351	0.74	172.42	9.07
Proposed Fig. 3(a)	SOBL	2.43	2.39	4.82	0.73	1369	0.73	170.09	8.158
$P(x) = x^{233} + x^{74} + 1$ (With the main controller)									
LSB-first [16]	POBL	2.22	2.67	4.89	0.4	2500	93.2	93.2	9.625
MSB-first [16]	POBL	1.79	2.67	4.46	0.41	2439	95.53	95.53	8.297
SOBL [19]	SOBL	2.59	3.01	5.6	0.83	1204	0.83	193.39	8.037
Proposed Fig. 1(a)	SOBL	2.77	3.01	5.78	0.74	1351	0.74	172.42	9.07
Proposed Fig. 3(a)	SOBL	2.43	2.39	4.82	0.73	1369	0.73	170.09	8.158

† KGate is the area equivalence in terms of number of NAND gates $\times 10^3$ (estimated area of one NAND gate is $2.08\ \mu m^2$).
†† The power consumption readings were conducted under 666 MHz frequency for all the designs.

TABLE 7: ASIC synthesis results for the proposed double multiplication architectures (Fig. 4(a), Fig. 4(b), Fig.4(d), and Fig.4(e)) for the polynomial basis over both $GF(2^{163})$ and $GF(2^{233})$ using 65-nm CMOS standard technology.

Type of Architecture	Type of Multiplier used	Area [KGate] †			CPD $[ns]$	Speed [MHz]	Total Time $[ns]$	TPT †† $[Mbps]$	TPT/Area $[Kbps/Gate]$	Dynamic Power ††† $[mW]$	Energy †††† $[m.J/Gbit]$
		CA	Non-CA	Total							
$P(x) = x^{163} + x^7 + x^6 + x^3 + 1$ (Without the main controller)											
LSB-first double Fig. 4(a)	POBL [16]	2.00	2.45	4.45	0.41	2439	133.7	1219	274	7.76	6.36
MSB-first double Fig. 4(b)	POBL [16]	1.88	2.45	4.33	0.32	3125	104.3	1562	361	7.68	4.91
Hybrid-double Fig. 4(d)	SOBL [19]	2.75	3.08	5.83	0.87	1149	142.7	1142	196	9.408	8.23
Hybrid-double Fig. 4(e)	SOBL Fig. 1(a)	3.01	3.17	6.18	0.62	1613	101.7	1603	260	11.01	6.87
$P(x) = x^{163} + x^7 + x^6 + x^3 + 1$ (With the main controller)											
LSB-first double Fig. 4(a)	POBL [16]	2.05	2.51	4.56	0.48	2083	156.5	1041	229	8.907	8.55
MSB-first double Fig. 4(b)	POBL [16]	1.97	2.51	4.48	0.45	2174	150.0	1087	243	8.22	7.56
Hybrid-double Fig. 4(d)	SOBL [19]	2.79	3.13	5.92	0.87	1149	142.7	1142	193	9.506	8.32
Hybrid-double Fig. 4(e)	SOBL Fig. 1(a)	3.01	3.17	6.18	0.62	1613	101.7	1603	260	11.01	6.87
$P(x) = x^{233} + x^{74} + 1$ (Basic Controller)											
LSB-first double Fig. 4(a)	POBL [16]	2.84	3.5	6.34	0.42	2380	195.72	1190	188	11.15	9.37
MSB-first double Fig. 4(b)	POBL [16]	2.66	3.5	6.16	0.33	3030	153.78	1515	246	10.99	7.25
Hybrid-double Fig. 4(d)	SOBL [19]	4.14	4.64	8.78	0.8	1250	187.2	1245	142	14.11	11.34
Hybrid-double Fig. 4(e)	SOBL Fig. 1(a)	4.36	4.75	9.11	0.61	1640	142.74	1632	179	15.64	9.58
Hybrid-double Fig. 4(e)	SOBL Fig. 3(a)	4.02	4.20	8.22	0.57	1754	133.38	1747	213	14.15	8.1
$P(x) = x^{233} + x^{74} + 1$ (With the main controller)											
LSB-first double Fig. 4(a)	POBL [16]	2.89	3.56	6.45	0.52	1923	242.32	961	149	12.76	13.27
MSB-first double Fig. 4(b)	POBL [16]	2.73	3.56	6.29	0.45	2222	209.7	1111	177	11.70	10.53
Hybrid-double Fig. 4(d)	SOBL [19]	4.19	4.69	8.88	0.79	1265	184.86	1260	142	14.26	11.31
Hybrid-double Fig. 4(e)	SOBL Fig. 1(a)	4.36	4.75	9.11	0.61	1640	142.74	1632	179	15.64	9.58
Hybrid-double Fig. 4(e)	SOBL Fig. 3(a)	4.02	4.20	8.22	0.57	1754	133.38	1747	213	14.15	8.1

† KGate is the area equivalence in terms of number of NAND gates $\times 10^3$ (estimated area of one NAND gate is $2.08\ \mu m^2$).
†† TPT is the throughput and is equal to the number of bits per cycle times the speed.
††† The power consumption readings were conducted under 666 MHz frequency for all the designs.
†††† Obtained by $\frac{\text{dynamic power}}{\text{throughput}}$.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2015.2456023, IEEE Transactions on Computers

and Engineering Research Council (NSERC) of Canada. The authors would like to thank Canadian Microelectronics Corporation (CMC) Microsystems for providing the required infrastructure and CAD tools that have been used in this work.

REFERENCES

[1] R. Lidl, and H. Niederreiter, *Introduction to Finite Fields and Their Applications*. 2nd Ed., Cambridge Univ. Press, Cambridge, UK, Aug. 1994.

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

[3] A. J Menezes, I. F. Blake, X. Gao, R. C. Mullin, S. A. Vanstone, and T. Yaghoobian, *Applications of Finite Fields*. Kluwer Academic Publishers, Boston, MA, 1993.

[4] R. E. Blahut, *Fast Algorithms for Digital Signal Processing*. 1st Ed., Addison-Wesley, Reading, MA, Sept. 1985.

[5] V. S. Miller, "Use of Elliptic Curves in Cryptography," *In Proc. of Advances in Cryptology-CRYPTO'85*, LNCS, 1986, vol. 218, pp. 417-426.

[6] N. Koblitz, "Elliptic Curve Cryptosystems," *Mathematics of Computation*, vol. 48, no. 177, pp. 203-209, Jan. 1987.

[7] T. Elgamal, "A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms," *IEEE Trans. Inf. Theory*, vol. 31, no. 4, pp. 469-472, Jul. 1985.

[8] W. Diffie, and M. Hellman, "New Directions in Cryptography," *IEEE Trans. Inf. Theory*, vol. 22, no. 6, pp. 644-654, Nov. 1976.

[9] M. A. Hasan, A. H. Namin, and C. Negre, "Toeplitz Matrix Approach for Binary Field Multiplication Using Quadrinomials," *IEEE Trans. VLSI Systems*, vol. 20, no. 3, pp. 449-458, Mar. 2012.

[10] H. Wu, "Bit-Parallel Polynomial Basis Multiplier for New Classes of Finite Fields," *IEEE Trans. Computers*, vol. 57, no. 8, pp. 1023-1031, Aug. 2008.

[11] A. Hariri, and A. Reyhani-Masoleh, "Bit-Serial and Bit-Parallel Montgomery Multiplication and Squaring over $GF(2^m)$," *IEEE Trans. Computers*, vol. 58, no. 10, pp. 1332-1345, Oct. 2009.

[12] I.S. Hsu, T. K. Truong, L. J. Deutsch, and I. S Reed, "A Comparison of VLSI Architecture of Finite Field Multipliers Using Dual, Normal, or Stnadard Basis," *IEEE Trans. Computers*, vol. 37, no. 6, pp. 735-739, Jun. 1988.

[13] D. Hankerson, A. Menezes, and S. Vanstone, *Guide to Elliptic Curve Cryptography*. New York: Springer-Verlag, 2004.

[14] E. D. Mastrovito, "VLSI Designs for Multiplication over Finite Field $GF(2^m)$," *Proc. Sixth Symp. Applied Algebra, Algebraic Algorithms, and Error Correcting Codes (AAECC-6)*, pp. 297-309, Jul. 1988.

[15] E. D. Mastrovito, "VLSI Architectures for Computation in Galois Fields," PhD thesis, Linköping Univ., Linköping, Sweden 1991.

[16] T. Beth, and D. Gollmann, "Algorithm Engineering for Public Key Algorithms," *IEEE J. Selected Areas in Communications*, vol. 7, no. 4, pp. 458-466, May 1989.

[17] R. Azarderakhsh, and A. Reyhani-Masoleh, "Low-Complexity Multiplier Architectures for Single and Hybrid-Double Multiplications in Gaussian Normal Bases," *IEEE Trans. Computers*, vol. 62, no. 4, pp. 744-757, Jan. 2012.

[18] R. Azarderakhsh, K. Järvinen, and V. Dimitrov, "Fast Inversion in $GF(2^m)$ with Normal Basis Using Hybrid-Double Multipliers," *IEEE Trans. Computers*, in process.

[19] A. Reyhani-Masoleh, "A New Bit-Serial Architecture for Field Multiplication Using Polynomial Bases," *In Proc. of CHES 2008*, Aug. 2008, LNCS 5154, pp. 300-314.

[20] H. Wu, "Bit-Parallel Finite Field Multiplier and Squarer Using Polynomial Basis," *IEEE Trans. Computers*, vol. 51, no. 7, pp. 750-758, Jul. 2002.

[21] F. Rodriguez-Henriguez, and Ç. K. Koç, "Parallel Multipliers Based on Special Irreducible Pentanomials," *IEEE Trans. Computers*, vol. 52, no. 12, pp. 1535-1542, Dec. 2003.

[22] B. Sunar, and Ç. K. Koç, "Mastrovito Multiplier for All Trinomials," *IEEE Trans. Computers*, vol. 48, no. 5, pp. 522-527, May 1999.

[23] A. Halbuoğullari, and Ç. K. Koç, "Mastrovito Multiplier for General Irreducible Polynomial," *IEEE Trans. Computers*, vol. 49, no. 5, pp. 503-518, May 2000.

[24] T. Zhang, and K. K. Parhi, "Systematic Design of Original and Modified Mastrovito Multipliers for General Irreducible Polynomials," *IEEE Trans. Computers*, vol. 50, no. 7, pp. 734-748, Jul. 2001.

[25] S. S. Erdem, T. Yanik, and Ç. K Koç, "Polynomial Basis multiplication over $GF(2^m)$," *Acta Applicandae Mathematicae*, vol. 93, no. 1, pp. 33-55, Sep. 2006.

[26] A. Reyhani-Masoleh, and M. A. Hasan, "Low Complexity Bit Parallel Architectures for Polynomial Basis Multiplication over $GF(2^m)$," *IEEE Trans. Computers*, vol. 53, no. 8, pp. 945-959, Aug. 2004.

[27] A. Reyhani-Masoleh, "Efficient Algorithms and Architectures for Field Multiplication Using Gaussian Normal Bases," *IEEE Trans. Computers*, vol. 55, no. 1, pp. 34-47, Jan. 2006.

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

[29] L. Song, and K. K. Parhi, "Efficient Finite Field Serial/Parallel Multiplication," *In Proc. of Int. Conf. Application Specific Syst., Architectures and Processors (ASAP)*, Chicago, IL, Aug. 1996, pp. 72-82.

[30] R. Katti, and J. Brennan "Low Complexity Multiplication in a Finite Field Using Ring Representation," *IEEE Trans. Computers*, vol. 52, no. 4, pp. 418-427, Apr. 2003.

[31] *Digital Signature Standard (DSS)*, Fed. Information Processing Standard, Nat'l Inst. of Standards and Technology Std. FIPS PUB 186-3, June 2009.

[32] *Recommended Elliptic Curve Domain Parameters*, Standards for Efficient Cryptography, Certicom Research Std. SEC 2, Sept. 2000.

[33] Synopsys, Inc. [Online]. Available: http://www.synopsys.com

Eberahim A. Hasan Abdulrahman received the BSc degree in computer science and engineering from Qatar University, Doha, Qatar, in 2002, with the first rank, the MSc degree in information technology (networking) from James Cook University, Townsville, QLD, Australia, in 2005, and the PhD degree in electrical and computer engineering from Western University, London, ON, Canada, in 2013. In February 2002, he joined the Department of Computer Engineering, University of Bahrain as a Graduate Teaching and Research Assistant, where he was awarded a master and a Ph.D. scholarship. He is currently an assistant professor at the University of Bahrain.

Arash Reyhani-Masoleh Arash Reyhani-Masoleh received the BSc degree in electrical and electronic engineering from Iran University of Science and Technology in 1989, the MSc degree in electrical and electronic engineering from the University of Tehran in 1991, both with the first rank, and the PhD degree in electrical and computer engineering from the University of Waterloo in 2001. From 1991 to 1997, he was with the Department of Electrical Engineering, Iran University of Science and Technology. From June 2001 to September 2004, he was with the Center for Applied Cryptographic Research, University of Waterloo, where he was awarded a Natural Sciences and Engineering Research Council of Canada (NSERC) Postdoctoral Fellowship in 2002. In October 2004, he joined the Department of Electrical and Computer Engineering, Western University, London, Canada, where he is currently a tenured associate professor. His current research interests include fault-tolerant computing, algorithms and VLSI architectures for computations in finite fields, cryptography, and error-control coding. He has been a two-time recipient of NSERC Discovery Accelerator Supplement (DAS) award in 2010 and 2015. Currently, he serves as an associate editor for Integration, the VLSI Journal (Elsevier). He is a member of the IEEE and the IEEE Computer Society.