Ain Shams Engineering Journal 14 (2023) 102188

Contents lists available at ScienceDirect

Ain Shams Engineering Journal

journal homepage: www.sciencedirect.com

Low-complexity systolic array structure for field multiplication in resource-constrained IoT nodes

Atef Ibrahim *

Computer Engineering Department, College of Computer Engineering and Sciences, Prince Sattam Bin Abdulaziz University, Alkharj, Saudi Arabia
ECE Department, University of Victoria, Victoria, BC, Canada

ARTICLE INFO

Article history:
Received 27 July 2022
Revised 7 January 2023
Accepted 19 January 2023
Available online 17 February 2023

Keywords:
Serial multipliers
Finite field arithmetic
Crypto-processors
IoT security
Systolic arrays
Embedded computing systems

ABSTRACT

Security and privacy issues with Internet of Things (IoT) network make it difficult to use IoT technology. Cryptographic protocols can be put in place on IoT edge nodes to address security flaws. Since the edge nodes have few resources, it is challenging to implement these protocols on them. The key operation in these protocols is finite-field multiplication, and how well it is carried out has a big effect on how well they perform. Therefore, we present in this work a novel irreducible All-One Polynomial (AOP)-based low-complexity bit-serial systolic implementation for multiplication in the binary-extended field. The shown multiplier structure features regular cell architectures and local communication connections between the cells, making it more appropriate for VLSI implementation. When compared to competing multipliers, the complexity analysis of the suggested multiplier reveals a significant savings in area and area-time product. As a result, it is more ideal for cryptographic systems that set additional constraints on area.

1. Introduction and related work

The IoT is presently having a significant influence on our everyday lives. Healthcare, transportation, entertainment, commercial appliances, agriculture, and housing are just a few of the industries they can be used in. Gathering data and sending it to the cloud for further analysis and decision-making is the main goal of the IoT network. The information collected by IoT devices needs to be protected at all IoT network tiers because the bulk of IoT applications are sensitive. Due to their limited resource availability, the majority of IoT edge nodes have trouble adapting security procedures. As a result, several efforts have been made to resolve this complex problem. On IoT edge nodes with limited resources, there are numerous security methods that can be used. Among many other cryptographic techniques, Elliptic Curve-Cryptography (ECC) is optimized for use on these nodes. Most optimized algorithms are based on finite-field arithmetic operations, particularly finite-field multiplication. Additionally, all other field operations, such as inversion, division, and exponentiation, are derived from finite-field multiplication. As a result, there is now a lot of interest in promoting the use of compact but highly potent cryptographic algorithms.

Many crucial applications of computer algebra depend on computation in the Galois field GF(2^m), including public-key cryptography and error-correcting codes [1,2]. The computing strategy used in these applications relies heavily on multiplication over GF(2^m). In the literature, there is a lot of discussion on how to minimize the space and delay overhead of this operation because it is quite expensive in terms of area and delay complexity [3–16].

The components of the GF(2^m) have a variety of base representations, including polynomial basis (PB), normal basis (NB), and dual basis (DB). The best option for a basis is PB because it offers simple, typical hardware constructions without the need for base conversion. [17]. Multiplication in GF(2^m) is much more complicated than addition and subtraction because it needs two stages to perform: polynomial multiplication and modulo reduction using an irreducible polynomial. Irreducible polynomials come in a variety of forms, including All-One Polynomials (AOP), ESPs (Equally Spaced Polynomials), trinomials, and pentanomials. In spite of the fact that the AOP class of irreducible polynomials provides a significantly efficient implementation of finite field multiplication, they are not as frequently researched as irreducible trinomials or pentanomials. As a result, the multiplier construction suggested in this work is based on AOP.

* Address: Computer Engineering Department, College of Computer Engineering and Sciences, Prince Sattam Bin Abdulaziz University, 16278 Alkharj, Saudi Arabia.
E-mail addresses: attif_ali2002@yahoo.com, aa.mohamed@psau.edu.sa, atef@ece.uvic.ca

https://doi.org/10.1016/j.asej.2023.102188
2090-4479/

The literature describes many hardware architectures, such as bit-parallel, bit-serial, and digit-serial multipliers, that implement PB multiplication in GF(2^m). Despite their enormous space complexity, bit-parallel multiplier designs accomplish field multiplication in a fixed number of iterations regardless of field size m [18]. Systolic bit-parallel multiplier topologies based on systolic arrays provide the advantage of greatly increasing multiplier throughput at the expense of increasing hardware complexity and latency. [14–16,19–22].

Bit-parallel architectures are not a better option in cryptography applications with restricted resources because of their high hardware needs. Because they enable a good speed/space tradeoff, bit-serial multiplier layouts are suited for cryptographic applications with greater space and power limitations. [5,6,10–13,23,24]. Nevertheless, they are not performance scalable and necessitate a minimum of m time steps to generate the multiplication result. Digit-serial multiplier architectures feature a little gain in hardware resources and performance scalability over bit-serial multiplier layouts. [25–29]. Because of this, they are better suited for cryptography algorithms with fixed word sizes of data.

The AOP-Based multiplication algorithm over GF(2^m) is implemented in this work using a new bit-serial systolic array construction. Using the derived dependency graph (DG) of the multiplication method, we investigated the bit-serial multiplier construction by selecting the appropriate scheduling and projection vectors. The bit-serial systolic array structure is obtained by mapping the DG nodes to the associated processing element (PE) and assigning the proper time to the DG nodes, respectively, using the chosen scheduling and projection vectors. The proposed bit-serial multiplier has much reduced area and area-time complexity than the effective bit-serial multiplier constructions that have been originally described in the literature. This makes it more appropriate for cryptographic techniques that have limited space.

The following shows how the paper is organized: The adopted AOP-based multiplication algorithm over $GF(2^m)$ is briefly explained in Section 2. The bit-serial systolic array multiplier structure has been examined and is provided in Section 3. The recommended multiplier layout and the observed effective bit-serial architectures are examined for their spatial and time complexity in Section 4. This work is concluded in Section 5.

2. Development of the finite field multiplication algorithm

Assume that the irreducible polynomial of degree $n, \Theta(\lambda)$, creates the finite field over the binary extension field, $GF(2^n)$. The polynomial representation of $\Theta(\lambda)$ can be described as follows:

$$\Theta(\lambda) = 1 + \theta_1\lambda^1 + \cdots + \theta_i\lambda^i + \cdots + \theta_{n-1}\lambda^{n-1} + \lambda^n \tag{1}$$

where $\theta_i \in GF(2)$. Let η be the irreducible polynomial $\Theta(\lambda)$'s root. This leads to the collection of polynomial bases $\{1, \eta, \eta^2, \eta^3, \cdots, \eta^{n-1}\}$ can be used to describe the field's constituent elements.

Consider two field elements in $GF(2^n)$ named A and B. They can be formulated as follows in the degree $n-1$ polynomial form:

$$A = a_0 + a_1\eta^1 + \cdots + a_i\eta^i + \cdots + +a_{n-1}\eta^{n-1} \tag{2}$$

$$B = b_0 + h_1\eta^1 + \cdots + b_i\eta^i + \cdots + +b_{n-1}\eta^{n-1} \tag{3}$$

with $a_i, b_i \in GF(2)$.

It is possible to multiply A and B over $GF(2^n)$ as follows:

$$P = A \cdot B \bmod \Theta(\lambda) \tag{4}$$

The following is a recurrence relation of multiplication that can be obtained by expanding Eq. (4):

$$P = b_0 \cdot A + \left[\sum_{i=1}^{n-1} b_i \cdot \eta^{i-1} \cdot F\right] \bmod \Theta(\lambda) \tag{5}$$

Since $F = \eta A$ is a polynomial of degree n, it could be written as follows:

$$F = \sum_{i=0}^{n} f_i \cdot \eta^i \tag{6}$$

where $f_0 = 0$ and $f_i = a_{i-1}$ for $i = 1, 2, \cdots, n$.

By multiplying the extended polynomial of (6) by η, we can get

$$\eta F = f_0\eta + f_1\eta^2 + \cdots + f_{n-1}\eta^n + f_n\eta^{n+1} \tag{7}$$

This results in $\Theta(\lambda) = 0$ because η is a root of $\Theta(\lambda)$. As a result, we can derive the following formula from Eq. (1).

$$\eta^n = 1 + \theta_1\eta + \theta_2\eta^2 + \cdots + \theta_{n-1}\eta^{n-1} \tag{8}$$

We can express Eq. (8) as follows when the algebraic $\Theta(\lambda)$ is an AOP:

$$\eta^n = 1 + \eta + \eta^2 + \cdots + \eta^{n-1} \tag{9}$$

When multiplying both sides of Eq. (9) by η, we can get:

$$\eta^{n+1} = 1 \tag{10}$$

We can reduce ηK to a polynomial (F^1) of degree n by replacing from (10) in (7) as shown below.

$$F^1 = f_n + f_0\eta + f_1\eta^2 + \cdots + f_{n-1}\eta^n \tag{11}$$

From Eq. (11), we can see that the partially-reduced polynomial F^1 of the polynomial ηF is created by the cyclic-shift-left of polynomial F. Similar to this, cyclic-shifting left on polynomial F^1 will result in the partially reduced polynomial F^2 of polynomial $\eta^2 F$. Typically, cyclic-shift-left of polynomial F^{i-1} will result in the partially reduced polynomial F^i of polynomial $\eta^i F$. This cyclic shift to the left can be expressed mathematically as:

$$F^i = CSL(F^{i-1}), \quad 0 \leqslant i \leqslant n-1 \tag{12}$$

where CSL stands for the cyclic-shift-left operation and $F^{-1} = (0\&A)$. We may use Eq. (12) to formulate Eq. (5) as:

$$P = b_0 \cdot A + \left[\sum_{i=1}^{n-1} b_i \cdot F^{i-1}\right] \bmod \Theta(\lambda) \tag{13}$$

with $F^0 = F = \eta A$.

Eq. (13) can also be expressed as:

$$P = C \bmod \Theta(\lambda) \tag{14}$$

where C is the summation of all polynomials of degree n, which are expressed as:

$$C = \sum_{i=0}^{n-1} b_i \cdot F^{i-1} \tag{15}$$

with $F^{-1} = (0\&A)$.

The polynomial in Eq. (15) can be modelled as follows:

$$C = c_0 + c_1\eta^1 + c_2\eta^2 + \cdots + c_{n-1}\eta^{n-1} + c_n\eta^n \tag{16}$$

The polynomial $C \bmod \Theta(\lambda)$ (polynomial of degree $n-1$) is simplified to the following expression by substituting the expansion obtained from Eq. (9) for η^n in Eq. (16):

$$\begin{aligned} P = C \bmod \Theta(\lambda) &= (c_0 \oplus c_n) + (c_1 \oplus c_n)\eta^1 \\ &+ (c_2 \oplus c_n)\eta^2 + \cdots + (c_{n-1} \oplus c_n)\eta^{n-1} \end{aligned} \tag{17}$$

If j is assumed to be any polynomial's bit position in a binary string, then we can state Eqs. (12) and (15) in their corresponding bit-level forms, as illustrated in Eqs. (18) and (19):

$$\begin{aligned} f_{j+1}^{i} &= f_{j}^{i-1} \\ f_{0}^{i} &= f_{n+1}^{i} \end{aligned} \tag{18}$$

$$c_j^i = c_j^{i-1} + b_i \cdot c_j^{i-1} \tag{19}$$

with $f_n^{-1} = 0, c_j^{-1} = 0, 0 \leqslant i \leqslant n-1$, and $0 \leqslant j \leqslant n$.

Additionally, the product polynomial D's reduced form, which is shown in Eq. (17), can be expressed as follows at the bit-level:

$$p_j = c_j^{n-1} + c_n^{n-1} \tag{20}$$

where $0 \leqslant j \leqslant n-1$.

2.1. Dependency Graph

The two iterative equations, Eqs. (18) and (19), represent the iterative part of the AOP-based field multiplication method. The two indices i and j control the number of iterations. The two-dimensional integer domain $\mathbb{D}$ can be used to create a dependence graph (DG) by using the method described in the reference [30]. The DG is displayed in Fig. 1 for the example $n = 5$. The vertices of the DG denote the operations designated by Eqs. (18) and (19). The signals of c_j^i are represented by the vertical lines in accordance with the construction criteria of reference [30]. Horizontal lines are used to depict signals b_i. The slanted lines serve as a representation for the signals f_{j+1}^i. The f_{n+1}^i signal is generated from the nodes in the final column and allocated to the nodes within the first column. The vertical and diagonal inputs to the nodes in the top row correspond to the algorithm inputs $c_j^{-1}, f_j^{-1} = a_j$. The expected outcome bits $p_j, 0 \leqslant j \leqslant n-1$ are formed by combining the resultant signals $c_j^{n-1}, 0 \leqslant j \leqslant n-1$, from the bottom row with the most significant signal c_n^{n-1} using XOR gates (blue vertices shown in Fig. 1), as indicated in the algorithm's reduction stage, Eq. (20).

3. Suggested bit-serial systolic array construction

In order to create the bit-serial systolic array layout, we must select the appropriate scheduling and projection vectors to apply to the DG. By employing the procedure already described in [16,30–37], the scheduling vector $\mathbf{S}$ and projection vector $\mathbf{P}$ that arise from the bit-serial systolic design should be $\mathbf{S} = [2 \ -1]$ and $\mathbf{P} = [1 \ 0]^T$, respectively. By applying the $\mathbf{S}$ and $\mathbf{P}$ vectors to the DG nodes $\mathbf{p}(i,j)$, we can get the scheduling function $t(\mathbf{p})$ and projection function $PE(\mathbf{p})$, that are utilized to give each DG node a time value and project each DG node to the associated PE in the systolic array, respectively.

$$t(\mathbf{p}) = n + 2i - j \tag{21}$$

$$PE(\mathbf{p}) = i \tag{22}$$

After employing the scheduling function $t(\mathbf{p})$ on the DG, the obtained node timing is shown in Fig. 2. As we can see, the provided time values result in the successive application of the inputs and successive generation of the DG products. One bit at a time, the input bits c_j^{-1} and f_j^{-1} are applied and the resulting output bits c_j^{n-1} are presented. The output is produced beginning with the MSB, c_n^{n-1}, at time $2n-2$ and ending with the LSB, c_0^{n-1}, at time $3n-2$. The serial-in serial-out (SISO) systolic array structure is depicted in Fig. 3 as a result of applying the projection function $PE(\mathbf{p})$ to the DG. It requires $3n-2$ clock cycles to produce the final output and consists of n unique PEs. Features of the PEs logic are shown in Fig. 5. (see Fig. 4)

By investigating the SISO systolic array, we notice that each input signal, b_i, is assigned to the corresponding PE, and the intermediate signals of f_{j+1}^i and c_j^i are pipelined between the adjacent

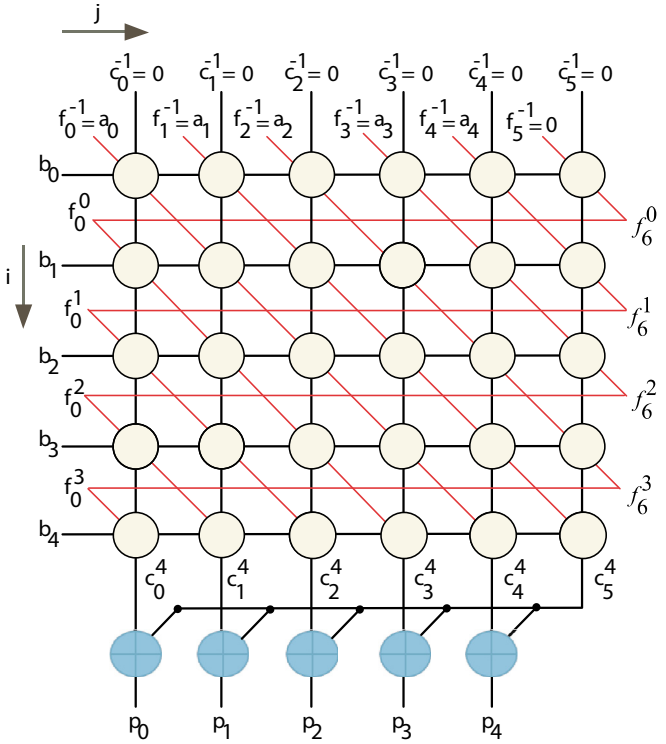

Fig. 1. DG for the AOP-Based multiplication algorithm for $m = 5$.

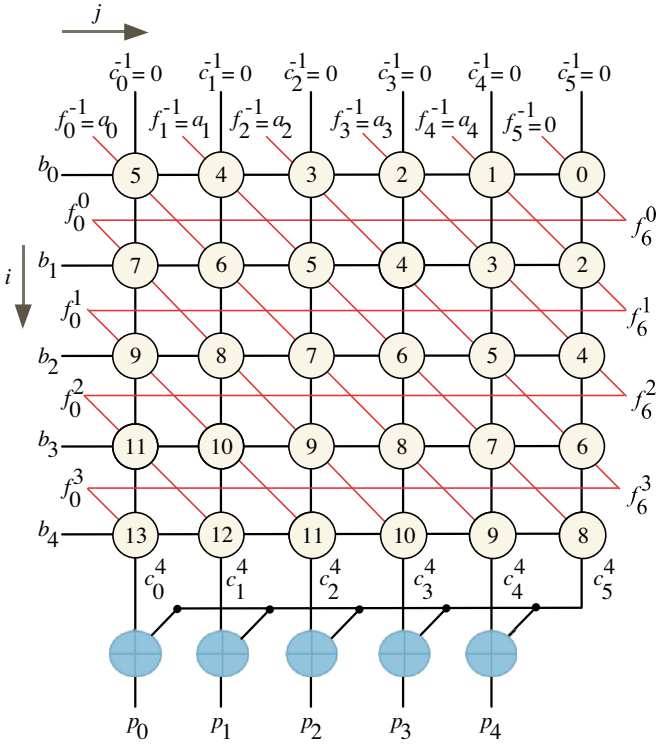

Fig. 2. Node timing for $m = 5$.

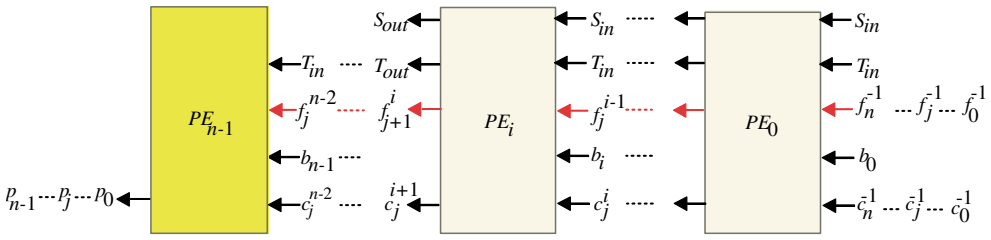

Fig. 3. Suggested SISO systolic multiplier.

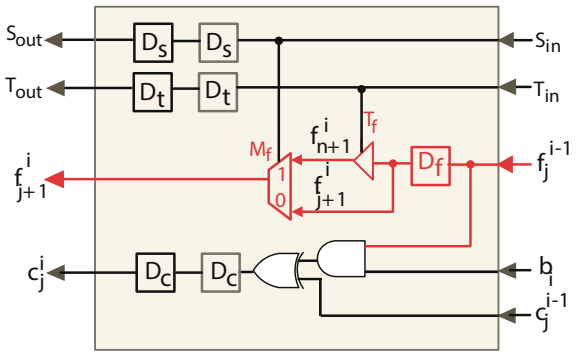

Fig. 4. PE_i logic circuit. The square blocks represent D-Latches.

PEs. The Tri-state buffer in Fig. 5 is controlled by control signal T_{in} to keep the MSB of F^i, f^i_{n+1}, to be transmitted to the next PE at the appropriate moment based on the select control signal S_{in}. Additionally, before being assigned to the following PE, these control signals are pipelined through D-Latches and delayed by two clock cycles. D-latch should be used to postpone the signal f^i_{j+1} by one clock cycle so that it can be applied to the following PE at the appropriate moment. By going across two D-latches, the signal c^i_j can be delayed by two clock cycles to ensure that it reaches the next PE at the appropriate moment. Before the SISO systolic array begins to operate, it is important to note that all D-Latches must be cleared. The SISO systolic array's functionality can be summed up as follows:

1. At the initial time instance ($t = 0$), the D-latch (D_f), shown in Fig. 5, is enabled to pass the input signal f^{-1}_n to be hold in the first PE (PE_0) as f^0_{n+1}. Holding this signal is performed by activating control signal T_{in} ($T_{in} = 1$) and deactivating control signal S_{in} ($S_{in} = 0$) to enable and disable tri-state buffer T_f and Multiplexer M_f, respectively. Input signals b_0, and c^{-1}_n are also applied to the first PE (PE_0) at this time instance to update the intermediate signal c^0_n using the AND-XOR connection shown in Fig. 5.
2. At the first time instance ($t = 1$), control signal T_{in} deactivates, $T_{in} = 0$, to disable the tri-state buffer T_f to prevent writing the input signal f^{-1}_{n-1} on the previously kept one, f^0_{n+1}, in the first PE (PE_0). Control signal S_{in} of Multiplexer M_f still deactivates, ($S_{in} = 0$), through this time instance to pass input signal f^{-1}_{n-1} to the next PE (PE_1). Also, through this time instance, input signals f^{-1}_{n-1} and c^{-1}_{n-1} are used in the first PE (PE_0) to compute the intermediate signal c^0_{n-1}.
3. When the second time instance ($t = 2$) occurs, the second PE (PE_1)) begins operating normally like PE_0 to retain the Most Significant Bit (MSB) of F^0 (f^0_n) and refresh the intermediary signals f^1_{n+1} and c^1_n. The first PE(PE_0), which is currently modifying the intermediary signals, f^0_{n-1}, and c^0_{n-2}, is also operating during this time instance.
4. As a general case, at time instances ($t = 2i$), $2 \leqslant i \leqslant n-1$, the PEs ($PE_i$) resume work normally like PE_0 and PE_1 to retain the MSB of F^i (i.e., f^i_n) and refresh the intermediary signals of f^i_{n+1} and c^i_n. Furthermore, during this time interval, the preceding PE (PE_{i-1}) is attempting to update the intermediary signals f^{i-1}_{n-1}, and c^{i-1}_{n-2}.
5. Throughout time instances ($t > 2i$), $0 \leqslant i \leqslant n-1$, the PEs ($PE_i$) refresh the intermediary signals of f^i_{j+1} and $c^i_j, 0 \leqslant j \leqslant n-1$.
6. At time instances $t = n + 2i + 2, 0 \leqslant i \leqslant n-2$, control signal S_{in} should be activated $S_{in} = 1$ to enable Multiplexer M_f to pass the preserved signal f^i_{n+1} to the next PE at the proper time.
7. At time instance ($t = 2n-2$), the D-latch (D_c) in the last PE (PE_{n-1}), shown in Fig. 5, is enabled to pass the computed signal c^{n-1}_n to be hold in the last PE (PE_{n-1}). Holding this signal is performed by activating control signal T_{in} ($T_{in} = 1$) to enable tri-state buffer T_f.
8. At time instances ($t > 2n-2$), control signal T_{in} deactivates ($T_{in} = 0$) to disable the tri-state buffer T_f to prevent writing the computed signals $c^{n-1}_j, 0 \leqslant j \leqslant n-1$, on the previously kept one, c^{n-1}_n, in the last PE (PE_{n-1}). At these time instances, the preserved signal c^{n-1}_n and signals $c^{n-1}_j, 0 \leqslant j \leqslant n-1$, are logically combined in sequence using the XOR gate shown in Fig. 5 to produce the final product $p_j, 0 \leqslant j \leqslant n-1$.
9. The final product bits of $p_j, 0 \leqslant j \leqslant n-1$ are produced in serial starting with the Most Significant Bit (MSB) p_{n-1} at time instant $2n-1$ and ending with Least Significant Bit (LSB) p_0 at time instant $3n-2$.

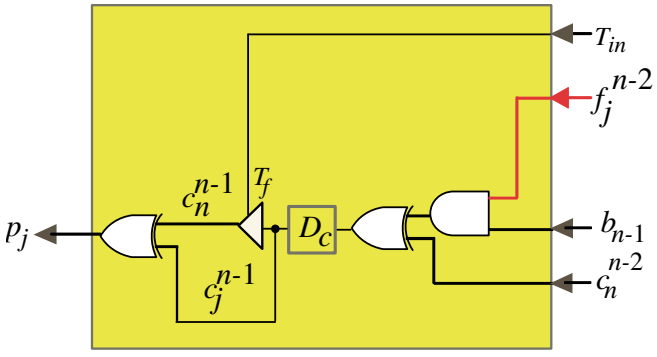

Fig. 5. PE_{n-1} logic circuit.

4. Results and discussions

This part addresses the comparison between the proposed SISO systolic multiplier layout with the rival bit-serial systolic and non-systolic multiplier architectures that have already been published

Table 1
Comparison of the area and delay complexity of the provided Multiplier layout to alternative bit-serial systolic multipliers.

Design	Type	TSB	AND	XOR	MUX	Latches	Latency	CPD
Song [5]	Systolic	0	$3n$	$3n$	$3n$	$14n$	$3n$	$\Pi_A + \Pi_X + \Pi_M$
Fenn [6]	Systolic	0	$4n-1$	$2n-1$	n	$12n-7$	$3n-2$	$\Pi_A + \Pi_X$
Choi [11]	Systolic	0	$3n$	$3n$	$3n$	$14n$	$3n-1$	$\Pi_A + T_X + \Pi_M$
Masoleh [38]	Non-Systolic	0	$R1^{(1)}$	$R2^{(1)}$	0	$R3^{(1)}$	n	$\Pi_A + (2 + \lceil \log_2(n) \rceil)\Pi_X$
Masoleh [12]	Non-Systolic	0	$R4^{(2)}$	$R5^{(2)}$	0	$R6^{(2)}$	n	$\Pi_A + \lceil \log_2(n) \rceil \Pi_X$
Ibrahim [13]	Semi-Systolic	0	$6w^{(3)}$	$6w+2$	$2w$	$27w-3$	$3w+n+1$	$\Pi_A + \Pi_X + \Pi_M$
Ibrahim [16]	Systolic	$2n$	$3n$	$3n$	n	$9n$	$3n-2$	$\Pi_A + \Pi_X + \Pi_M$
Ibrahim [39]	Systolic	n	$2n$	$2n$	0	$8n$	$3n-2$	$\Pi_A + \Pi_X + \Pi_{tri}$
Proposed	Systolic	n	n	$n+1$	$n-1$	$7n-6$	$3n-2$	$\Pi_A + \upsilon_X$

(1) $R1 = 5n+3, R2 = 5n+2\upsilon-4, R3 = 19n+4\upsilon-4$, with υ is indeed the power of the second term of the trinomial, $(x^n + x^\upsilon + 1)$.
(2) $R4 = 7n+6, R5 = 7n+6, R6 = 18n-2\upsilon-2$,
(3) $w = \lceil n/2 \rceil$.

Table 2
Costs of serial structures in terms of space and time for $n = 233$ and $t = 74$.

Design	A [*Kgates*]	T [*ns*]	AT	%A	%AT
Song [5]	18.0	25.1	451.8	56.6	62.0
Fenn [6]	17.3	21.9	378.9	54.9	54.9
Choi [11]	18.0	25.0	450.0	56.7	62.0
Masoleh [38]	42.0	32.0	1344.0	81.4	87.3
Masoleh [12]	41.0	26.0	1066.0	80.9	83.9
Ibrahim [13]	16.0	21.0	357.0	51.3	49.2
Ibrahim [16]	12.6	24.8	312.5	38.1	45.3
Ibrahim [39]	9.4	22.2	208.7	17.0	18.1
Proposed	7.8	21.9	170.8	-	-

[5,6,11–13,16,38,39]. The outcomes are compiled in Table 1 in regards to the total gate counts, latency, and critical path delay (CPD). The symbols $\Pi_{tri}, \Pi_A, \Pi_X, \Pi_M$ stand for the latencies of the logic components of the Tri-State Buffers, the two-input AND gates, two-input XOR gates, and two-to-one MUXs, respectively.

Before analysing the area and delay complexity findings from Table 1, it is important to know that only a few of the multiplier constructions indicated in [12,38] are systolic or semi-systolic. The architectures provided in [12,38] are built on two different kinds of irreducible polynomials: trinomials and ω-nomials (irreducible polynomials with ω non-zero terms). Since trinomial-based solutions outperform ω-nomial solutions, we choose to compare them with the offered multiplier construction. The created multiplier layout has a substantially less count of gates (n AND gates, $n+1$ XOR gates, $n-1$ MUXes, and $7n-6$ D-Latches) and more n Tri-State buffers (n) than other multiplier layouts, according to the estimated findings shown in Table 1. Only one other multiplier, the Fenn [6] multiplier, contains almost as many XOR gates ($2n-1$) as the created multiplier. Also, the design offered by [39] has zero MUXs and the same number of the Tri-state buffers. Additionally, we recognize that the non-systolic multiplier constructions cited by [12,38] have drastically reduced latency than the one that was recommended but have a substantially higher Critical Path Delay (CPD). Additionally, the multiplier of Fenn [6] has a similar latency and similar CPD like the one that was recommended. Ibrahim's multiplier structure [13] has latency of approximately ($\approx 2.5n+1$) and CPD that is larger than the CPD of the proposed design by more MUX delay. Despite Ibrahim's multiplier construction have lower latency than the proposed multiplier architecture, the suggested multiplier layout has a somewhat shorter total computation time. This is attributed to the lower CPD of the offered design outperforms the lower latency of the design of Ibrahim [13] as will be verified later using real implementation.

We assessed the overall area (A), the entire processing time (T), and the area-time (AT) of the compared structures for $n = 233$ and $t = 74$, as shown in Table 2, to support our observations of Table 1. To assess the area (total gate count) and delay of the crucial logic components, we employed NanGate (15 nm, 0.8 V) Open Cell Library. While the delay is evaluated for each basic component separately, the area of each basic component (S) is calculated as a function of the two-input NAND gate. We can summarize the assessed results for each component as follows:

1. Tri-State Buffer: $S_{tri} = 0.8, \Pi_{tri} = 7.9ps$.
2. Two-input AND gate: $S_A = 1.2, \Pi_A = 11.3ps$
3. Two-input XOR gate: $S_X = 2.5, \Pi_X = 12.7ps$,
4. 2-to-1 MUX: $S_M = 2.5, \Pi_M = 12.4ps$,
5. D-Latch: $S_{Latch} = 2.8, \Pi_{Latch} = 16.6ps$.

According to the results listed in Table 2, the offered SISO systolic array multiplier construction substantially outperforms the recently reported proficient bit-serial multiplier constructions in aspects of area (A) and area-time (AT) product by average values of at least 17.4% and 18.14%, respectively. As a result, the findings obtained support the idea that the suggested architecture is appropriate for cryptosystems used in IoT applications that place greater restrictions on space complexity due to their limited resources.

5. Summary and conclusion

The AOP-based multiplication technique over the binary-extended field was performed in this paper using a new SISO systolic array design. The suggested architecture lends itself better to VLSI implementations owing to its regular processing element builds and localized interconnections. Furthermore, opposed to the multiplier architectures of its existing rivals, it has the distinction of lowering area overhead with a respectable delay. As a result, it could really work effectively in cryptosystems utilized in IoT applications that place greater restrictions on space complexity.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This study is supported via funding from Prince Sattam bin Abdulaziz University project number (PSAU/2023/R/1444)

References

[1] Blahut RE, Katz J, van Oorschot PC, Vanstone SA. Handbook of Applied Cryptography. Boca Raton, FL: CRC Press; 1996.
[2] Blahut RE. Theory and Practice of Error Control Codes. Reading, MA: Addison-Wesley; 1983.
[3] Haa J-C, Moon S-J. A common-multiplicand method to the montgomery algorithm for speeding up exponentiation. Inform Process Lett 1998;66(2):105–7.
[4] Lee K-J, Yoo K-Y. Linear systolic multiplier/squarer for fast exponentiation. Inform Process Lett 2000;76:105–11.
[5] Song L, Parhi KK. Efficient finite field serial/parallel multiplication. In: Proceedings of the IEEE 1996 international conference on application-specific architectures and processors; 1996. p. 72–82.
[6] Fenn STJ, Taylor D, Benaissa M. A dual basis bit serial systolic multiplier for gf (2^m). Integr. VLSI J 1995;18:139–49.
[7] C.-W. Chiou, C.-Y. Lee, A.-W. Deng, J.-M. Lin, Concurrent error detection in montgomery multiplication over GF(2^m), IEICE Transactions on Fundamentals of Electronics Communications and Computer Sciences E89-A (2) (2006) 566–574.
[8] Huang WT, Chang C, Chiou C, Chou F. Concurrent error detection and correction in a polynomial basis multiplier over GF(2^m). IET Inf Secur 2010;4.
[9] Kim KW, Jeon JC. Polynomial basis multiplier using cellular systolic architecture. IETE J Res 2014;60(2):194–9.
[10] Kim KW, Jeon JC. A semi-systolic montgomery multiplier over GF(2^m). IEICE Electron Exp 2015;12(21):1–6.
[11] Choi S, Lee K. Efficient systolic modular multiplier/squarer for fast exponentiation over GF(2^m). IEICE Electron Exp 2015;12(11):1–6.
[12] Abdulrahman EA, Reyhani-Masoleh Ah. High-speed hybrid-double multiplication architectures using new serial-out bit-level mastrovito multipliers. IEEE Trans Comput 2016;65(6):1734–47.
[13] Ibrahim A. Novel bit-serial semi-systolic array structure for simultaneously computing field multiplication and squaring. IEICE Electron Exp 2019;16(23):20190600.
[14] Kim KW, Lee JD. Efficient unified semi-systolic arrays for multiplication and squaring over GF(2^m). IEICE Electron Exp 2017;14(12):1–10.
[15] Kim KW, Kim SH. Efficient bit-parallel systolic architecture for multiplication and squaring over GF(2^m). IEICE Electron Exp 2018;15(2):1–6.
[16] Ibrahim A. Efficient parallel and serial systolic structures for multiplication and squaring over GF(2^m). Canad J Electr Comput Eng 2019;42(2):114–20.
[17] Hsu IS, Truong TK, Deutsch LJ, Reed I. A comparison of vlsi architecture of finite field multipliers using dual, normal, or standard bases. IEEE Trans Comput 1988;37(6):735–9.
[18] Wu H. Bit-parallel finite field multiplier and squarer using polynomial basis. IEEE Trans Comput 2002;51(7):750–8.
[19] Lee C-Y. Low-latency bit-pararallel systolic multiplier for irreducible $x^m + x^n + 1$ with GCD$(m, n) = 1$. IEICE Trans Fund Elect, Commun Comp Sci 2008;55(3):828–37.
[20] Y.-R. Ting, E.-H. Lu, Y.-C. u, Ringed bit-parallel systolic multipliers over a class of fields GF(2^m), Integration VLSI J. 38 (4) (2005) 371–384.
[21] Fournaris AP, Koufopavlou O. Versatile multiplier architectures in gf(2^k) fields using the montgomery multiplication algorithm. Integration VLSI J. 2008;41(3):571–8.
[22] K.-W. Kim, H.-H. Lee, S.-H. Kim, Efficient combined algorithm for multiplication and squaring for fast exponentiation over finite fields GF(2^m), in: Proc. 7th International Conference on Emerging Databases, LNEE 461, 2017, pp. 50–57.
[23] Kitsos P, Theodoridis G, Koufopavlou O. An efficient reconfigurable multiplier architecture for galois field GF(2^m). Elsevier Sci Microelectron 2003;34:975–80.
[24] Selimis GN, Fournaris AP, Michail HE, Koufopavlou O. Improved throughput bit-serial multiplier for GF(2^m) fields. Integration VLSI J 2009;42:371–84.
[25] Guo J-H, Wang C-L. Digit-serial systolic multiplier for finite fields GF(2^m). IEEE Proc Comput Digital Tech 1998;145(2):143–8.
[26] M. Hutter, J. Grobschald, G.-A. Kamenje, A versatile and scalable digit-serial/parallel multiplier architecture for finite fields gf(2^m), in: Proc. 2003 4^{th} International Conference on InformationTechnology: Coding and Computing (ITTCC2003), 2003, pp. 692–700.
[27] Song L, Parhi KK. Low-energy digit serial/parallel finite field multipliers. J VLSI Signal Process Syst 1998;19(2):149–66.
[28] Kim CH, Hong CP, Kwon S. A digit-serial multiplier for finite field GF(2^m). IEEE Trans Very Large Scale Integr (VLSI) Sys 2005;13(4):476–83.
[29] A. Hariri, A. Reyhani-Masoleh, Digit-serial structures for the shifted polynomial basis multiplication over binary extension fields, in: Proc. LNCS Intl Workshop Arithmetic of Finite Fields (WAIFI), 2008, pp. 103–116.
[30] Gebali F. Algorithms and Parallel Computers. New York, USA: John Wiley; 2011.
[31] Ibrahim A, Gebali F. Low power semi-systolic architectures for polynomial-basis multiplication over GF(2^m) using progressive multiplier reduction. J Signal Process Syst 2016;82(3):331–43.
[32] Ibrahim A, Gebali F, Al-Somani T. Systolic array architectures for Sunar Koc optimal normal basis type ii multiplier. IEEE Trans Very Large Scale Integr VLSI Syst 2015;23(10):2090–102.
[33] Ibrahim A, Gebali F. Scalable and unified digit-serial processor array architecture for multiplication and inversion over GF(2^m). IEEE Trans Circ Syst I Regul Pap 2017;22(11):2894–906.
[34] Ibrahim A, Elsimary H, Gebali F. New systolic array architecture for finite field division. IEICE Electron Exp 2018;15(11):1–11.
[35] Ibrahim A. Scalable digit-serial processor array architecture for finite field division. Microelectron J 2019;85:83–91.
[36] Ibrahim A, Alsomani T, Gebali F. Unified systolic array architecture for field multiplication and inversion over GF(2^m). Comput Electr Eng J-Elsevier 2017;61:104–15.
[37] Ibrahim A, Alsomani T, Gebali F. New systolic array architecture for finite field inversion. IEEE Canad J Electr Comput Eng 2017;40(1):23–30.
[38] A. Reyhani-Masoleh, A new bit-serial architecture for field multiplication using polynomial bases, in: Proceedings of the 7^{th} International Workshop Cryptographic Hardware Embedded Systems (CHES 2008), 2008, pp. 330–314.
[39] Ibrahim A. Low-space bit-serial systolic array architecture for interleaved multiplication over GF(2^m). IET Comput Digital Tech 2021;15(3):223–9.

Atef Ibrahim received his B.Sc. degree in Electronics from Mansoura University and M.Sc. degree in Electronics and Electrical Communications from Cairo University. He obtained his PhD degree in Electronics and Electrical Communications from Cairo University jointly with the University of Victoria, Canada. Dr. Ibrahim is a professor of computer engineering at Prince Sattam Bin Abdulaziz University, KSA. Also, he is an adjunct professor at the University of Victoria. His research interests include Computer Arithmetic, Cryptography, Bio-Computing, Embedded Systems Design, and Digital VLSI design.