UVicSPACE: Research & Learning Repository

University of Victoria

Faculty of Engineering

Faculty Publications

Word-serial unified and scalable semi-systolic processor for field multiplication and squaring

Atef Ibrahim

February 2021

© 2021 Atef Ibrahim. This is an open access article distributed under the terms of the Creative Commons Attribution License. https://creativecommons.org/licenses/by-nc-nd/4.0/

This article was originally published at:

https://doi.org/10.1016/j.aej.2020.10.058

Citation for this paper:

Ibrahim, A. (2021). Word-serial unified and scalable semi-systolic processor for field multiplication and squaring. *Alexandria Engineering Journal,* 60(1), 1379-1388. https://doi.org/10.1016/j.aej.2020.10.058.

Alexandria Engineering Journal (2021) 60, 1379–1388

HOSTED BY

Alexandria University

Alexandria Engineering Journal

www.elsevier.com/locate/aej
www.sciencedirect.com

Word-serial unified and scalable semi-systolic processor for field multiplication and squaring

Atef Ibrahim *

Department of Computer Engineering, College of Computer Engineering and Sciences, Prince Sattam Bin Abdulaziz University, Al-Kharj 11942, Saudi Arabia
ECE Department, University of Victoria, Victoria, BC, Canada

Received 19 July 2020; revised 17 September 2020; accepted 23 October 2020
Available online 13 November 2020

KEYWORDS

Word-serial systolic/semi-systolic arrays;
Cryptographic processors;
Finite-field arithmetic;
Resource-constrained embedded applications;
Hardware security

Abstract This paper exhibits a word-serial unified and scalable semi-systolic processor core for concurrently executing both multiplication and squaring operations over GF(2^k). The processor is extracted by applying a chosen non-linear scheduling and projection functions to the dependency graph of the adopted bipartite multiplication-squaring algorithm. It has the advantage of sharing the data-path resources between the two operations leading to considerable savings in both space and power resources. Also, the processor's scalability nature provides the designer with higher flexibility to manage the processor size as well as its execution time. The acquired ASIC synthesis results of the explored word-serial multiplier-squarer architecture and the reported competing word-serial multiplier architectures indicate that the developed design significantly outperforms the competing ones in terms of area and consumed energy at the word-size of 32-bits. Therefore, the explored architecture is more suited for realizing cryptographic primitives in all resource-constrained embedded applications operating at this word-size.

© 2020 The Author. Published by Elsevier B.V. on behalf of Faculty of Engineering, Alexandria University. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

1. Introduction

Modern cryptography substantially depends on finite-field arithmetic operations such as addition, subtraction, multiplication, inversion, division, and exponentiation. There are two categories of cryptography: symmetric-key cryptography and public-key cryptography. In symmetric-key cryptography, encryption and decryption processes use the same secret key, while they use different keys in public-key cryptography. RSA cryptography [1] and epileptic curve cryptography (ECC) [2] are two important cryptographic techniques based on the public-key cryptography principle. They extensively use finite-field arithmetic operations to realize both the encryption and decryption processes.

Addition and subtraction operations can be easily realized using the logical XOR gate. The highly complicated operations such as inversion, division, and exponentiation are substantially achieved using recursive multiplication. Therefore, finite-field multiplication is considered the fundamental part of these complex operations and hence all the cryptographic techniques.

Modular exponentiation is an essential part of several cryptographic techniques, especially RSA cryptography. There are

* Address: Department of Computer Engineering, College of Computer Engineering and Sciences, Prince Sattam Bin Abdulaziz University, Al-Kharj 11942, Saudi Arabia.
E-mail address: atef@ece.uvic.ca.
Peer review under responsibility of Faculty of Engineering, Alexandria University.

https://doi.org/10.1016/j.aej.2020.10.058
1110-0168 © 2020 The Author. Published by Elsevier B.V. on behalf of Faculty of Engineering, Alexandria University.
This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

two binary algorithm techniques used to perform this operation. In the first technique, the algorithm scans the exponent bits starting from the rightmost bit (the least significant bit). In contrast, the second technique scans the exponent bits starting from the leftmost bit (the most significant bit). Both algorithm techniques are proceeded using a sequence of finite-field multiplication and squaring operations. The first algorithm technique can perform both multiplication and squaring operations concurrently to reduce the computation time. Therefore, many trials in the literature have merged both operations in a unified hardware structure to minimize the utilized space and increase the computation performance [3–5]. Unfortunately, all the developed merged structures mainly targeted the high-performance applications and neglected the resource-constrained embedded applications.

Scalable systolic/semi-systolic processors can be considered the optimal hardware structure for resource-constrained embedded applications. They achieve a trad-off between area and delay complexities. Therefore, they can combine the merits of the bit-serial and bit-parallel systolic/semi-systolic processors. The systolic/semi-systolic nature of these architectures makes them more efficient in VLSI implementation due to their regularity, modularity, and local interconnectivity between their processing elements. Bit-serial systolic/semi-systolic processors typically have low area-complexity and high delay-complexity, making them not suitable for high-speed applications. Bit-parallel systolic/semi-systolic processors usually have high area-complexity and low delay-complexity, making them not ideal for applications imposes restrictions on the area. On the other hand, the scalable systolic/semi-systolic processors allow flexibility to control the area and delay complexities to fit the design to the fixed embedded processor space.

2. Related work

There are various multiplier architectures over GF(2^k) realized in the literature [3,4,6,7] but they have high hardware and delay complexities. Therefore, they are not suitable for utilization in resource-constrained embedded applications. Word-serial multiplier architectures are the most suitable ones that can target these types of applications. This is due to that they have a trade-off between hardware and delay complexities. Therefore, the designer can easily manage these hardware structures to control their size and execution time. There are four categories of word-serial multiplier structures: serial-in/ serial-output [8–12], parallel-in/ serial-output [13], serial-in/ parallel-output [14–17], and scalable structures [18–24].

This paper presents an efficient word-serial unified and scalable semi-systolic processor core that performs both field multiplication and squaring operations simultaneously over GF (2^k). The combined structure provides the advantage of sharing hardware resources leading to more savings in hardware complexity and consumed power. The scalability offers the merit of adapting the processor size and its execution time to suit all the resource-constrained embedded applications. Also, the semi-systolic structures of the processor core make it more suitable for VLSI implementation. The processor core is extracted by applying a chosen non-linear scheduling and projection functions, based on the approach discussed in [25–30], to the dependency graph of the adopted bipartite multiplication-squaring algorithm.

The arrangement of the article is as follows. Section 3 presents a brief description of polynomial-based bipartite multiplication-squaring algorithm in $GF(2^k)$ and develops the corresponding bit-level representation. Section 4 displays the extraction process of the algorithm dependency graph (DG). Section 5 explains the exploration process of the word-serial unified and scalable semi-systolic processor core and provides its hardware details. Section 6 compares the performance of the proposed design to the competent ones in terms of area, delay, and consumed energy. Section 7 summarizes and concludes this work.

3. Polynomial-based bipartite multiplication-squaring algorithm over $GF(2^k)$

Suppose $H(\alpha)$ be the polynomial generator of the binary extension field $GF(2^k)$ and polynomials $A(\alpha)$ and $B(\alpha)$ be any arbitrary polynomial elements inside this field. The representation of these polynomials in $GF(2^k)$ can be as follows:

$$A(\alpha) = \sum_{i=0}^{k-1} a_i \alpha^i \tag{1}$$

$$B(\alpha) = \sum_{i=0}^{k-1} b_i \alpha^i \tag{2}$$

$$H(\alpha) = \sum_{i=0}^{k} h_i \alpha^i \tag{3}$$

where $a_i, b_i, h_i \in \mathrm{GF}(2)$ and k is the filed size.

Since α is a root of $H(\alpha)$, $\alpha^k \bmod H(\alpha)$ and $\alpha^{k+1} \bmod H(\alpha)$ can be represented as follows:

$$\alpha^k \bmod H(\alpha) = \sum_{j=0}^{k-1} h_j \alpha^j \tag{4}$$

$$\begin{aligned} \alpha^{k+1} \bmod H(\alpha) &= \sum_{j=1}^{k-1} \left(h_{k-1}h_j + h_{j-1}\right)\alpha^j + h_{k-1}h_0 \\ &\cong H'(\alpha) = \sum_{j=0}^{k-1} h'_j \alpha^j \end{aligned} \tag{5}$$

Consider that $H'(\alpha)$ is available in advance and suppose that $f = \lfloor k/2 \rfloor, g = \lceil k/2 \rceil$. We can define the polynomial multiplication and squaring over $GF(2^k)$ as follows:

$$\begin{aligned} P(\alpha) &= A(\alpha)B(\alpha) \bmod H(\alpha) \\ &= \sum_{i=0}^{k-1} b_i A(\alpha)\alpha^i \bmod H(\alpha) \\ &= \left(\sum_{i=0}^{g-1} b_{2i} A(\alpha)\alpha^{2i} + \alpha \sum_{i=0}^{f-1} b_{2i+1} A(\alpha)\alpha^{2i}\right) \bmod H(\alpha) \end{aligned} \tag{6}$$

$$\begin{aligned} S(\alpha) &= A(\alpha)A(\alpha) \bmod H(\alpha) \\ &= \sum_{i=0}^{k-1} a_i A(\alpha)\alpha^i \bmod H(\alpha) \\ &= \left(\sum_{i=0}^{g-1} a_{2i} A(\alpha)\alpha^{2i} + \alpha \sum_{i=0}^{f-1} a_{2i+1} A(\alpha)\alpha^{2i}\right) \bmod H(\alpha) \end{aligned} \tag{7}$$

$P(\alpha)$ and $S(\alpha)$ can be divided into two parts as follows:

$$P(\alpha) = (C(\alpha) + \alpha D(\alpha)) \bmod H(\alpha) \tag{8}$$

$$S(\alpha) = (Q(\alpha) + \alpha R(\alpha)) \bmod H(\alpha) \tag{9}$$

where,

$$C(\alpha) = \sum_{i=0}^{g-1} b_{2i} A(\alpha)\alpha^{2i} \bmod H(\alpha) \tag{10}$$

$$D(\alpha) = \sum_{i=0}^{f-1} b_{2i+1} A(\alpha)\alpha^{2i} \bmod H(\alpha) \tag{11}$$

$$Q(\alpha) = \sum_{i=0}^{g-1} a_{2i} A(\alpha)\alpha^{2i} \bmod H(\alpha) \tag{12}$$

$$R(\alpha) = \sum_{i=0}^{f-1} a_{2i+1} A(\alpha)\alpha^{2i} \bmod H(\alpha) \tag{13}$$

Algorithm 1 is the bipartite unified algorithm recommended by Kim [4,31] to concurrently computes the products $P(\alpha)$ and $S(\alpha)$. From now on, we will replace the polynomials $A(\alpha), C(\alpha), D(\alpha), Q(\alpha)$, and $R(\alpha)$ with the variables A, C, D, Q, and R, respectively. At iteration i, the partial results of these variables are represented as A^i, C^i, D^i, Q^i, and R^i. $b_{2i-2}, b_{2i-1}, a_{2i-2}$, and a_{2i-1} depicts the $(2i-2)^{th}$ and $(2i-1)^{th}$ bits of the input variables B and A, respectively. At the initialization step ($i = 0$), the algorithm assigns zero values to variables C, D, Q, and R. Through the i^{th} iterations of the algorithm, the for loop updates the intermediate results A^i, C^i, D^i, Q^i, and R^i of variables A, C, D, Q, and R, respectively, as shown in steps 2-to-6. After the final iteration of the for-loop, the post-processing steps 8 and 9 computes the products P and S, respectively.

To extract the data dependency graph of Algorithm 1, and hence exploring the hardware structure of the multiplier-squarer, we should represent Algorithm 1 in the bit-level form. The developed bit-level representation of Algorithm 1 is shown in Algorithm 2. In this algorithm, the elements $a_j^i, c_j^i, d_j^i, q_j^i$ and r_j^i represent the j^{th} bit of variables A, C, D, Q and R at the i^{th} iteration, respectively. The algorithm replaces the i for loop in Algorithm 1 by two for loops: the outer i for loop and the inner j for loop to compute, bit-by-bit, the intermediate partial results of variables A^i, C^i, D^i, Q^i, R^i . The last two steps, Steps 8 and 9 of Algorithm 2, are replaced by the post-processing for loop shown at the end of Algorithm 2 to be executed bit-by-bit. According to Step 2 in Algorithm 1, the value of A^{i-1} is multiplied by α^2. Thus, it should be shifted lift, before reduction, by two bits through each iteration of the outer for loop of Algorithm 2. Due to shifting left of A^{i-1} by two positions, the initial value of operand A, A^0, should be padded by two zero bits at the right, as shown in Algorithm 2. Also, through each iteration of the outer for loop, the least significant bits, a_{-1}^{i-1} and a_{-2}^{i-1}, of variable A^{i-1} should be assigned zero values as shown in Step 2 of Algorithm 2. Since the final values of variables D^f and R^f are multiplied by α, as shown in Steps 8 and 9 of Algorithm 1, they should be shifted left by 1-bit. Thus, their initial values, D^0 and R^0 should be padded by zero bit at the right and also their final least significant bits, d_{-1}^f and r_{-1}^f, should be forced to have zero value as shown in Step 11 of Algorithm 2.

Algorithm 1. Polynomial-based bipartite multiplication and squaring algorithm in GF(2^k) [4,31].

Input: $A, B \in \mathrm{GF}(2^k)$, $H, H', g = \lceil k/2 \rceil$, and $f = \lfloor k/2 \rfloor$
Mult. Output: $P = A.\ B \mod H$
Square Output: $S = A.\ A \mod H$
Initialization:
$A^{(0)} \leftarrow A,\ B \leftarrow B,\ C^0 \leftarrow 0,\ D^0 \leftarrow 0,\ Q^0 \leftarrow 0,\ R^0 \leftarrow 0,\ H \leftarrow H,\ H' \leftarrow H'$
Algorithm:
1: **for** $1 \leqslant i \leqslant g$ **do**
2: $A^i = A^{i-1}.\ \alpha^2 \mod H$
3: $C^i \leftarrow C^{i-1} + b_{2i-2}A^{i-1}$
4: $D^i \leftarrow D^{i-1} + b_{2i-1}A^{i-1}$
5: $Q^i \leftarrow Q^{i-1} + a_{2i-2}A^{i-1}$
6: $R^i \leftarrow R^{i-1} + a_{2i-1}A^{i-1}$
7: **end for**
8: $P \leftarrow \left(C^g + \alpha D^f\right) \bmod H$
9: $S \leftarrow \left(Q^g + \alpha R^f\right) \bmod H$

Algorithm 2. Bit-level form of Algorithm 1.

Input: $A, B \in \mathrm{GF}(2^k)$, $H, H', g = \lceil k/2 \rceil$, and $f = \lfloor k/2 \rfloor$
Mult. Output: $P = A.\ B \mod H$
Square Output: $S = A.\ A \mod H$
Initialization:
$A^0 = \left(a_{k-1}^0 \cdots a_1^0 a_0^0 a_{-1}^0 a_{-2}^0\right) \leftarrow (a_{k-1} \cdots a_1 a_0 00)$
$B \leftarrow (b_{k-1} \cdots b_1 b_0)$
$C^0 = \left(c_{k-1}^0 \cdots c_1^0 c_0^0\right) \leftarrow (0 \cdots 00)$
$D^0 = \left(d_{k-1}^0 \cdots d_1^0 d_0^0 d_{-1}^0\right) \leftarrow (0 \cdots 000)$
$Q^0 = \left(q_{k-1}^0 \cdots q_1^0 q_0^0\right) \leftarrow (0 \cdots 00)$
$R^0 = \left(r_{k-1}^0 \cdots r_1^0 r_0^0 r_{-1}^0\right) \leftarrow (0 \cdots 000)$
$H \leftarrow (h_{m-1} \cdots h_1 h_0)$
$H' \leftarrow \left(h'_{m-1} \cdots h'_1 h'_0\right)$
Algorithm:
1: **for** $1 \leqslant i \leqslant g$ **do**
2: $a_{-1}^{i-1} \leftarrow 0, a_{-2}^{i-1} \leftarrow 0$
3: **for** $0 \leqslant j \leqslant k-1$ **do**
4: $a_j^i = a_{j-2}^{i-1} + a_{k-2}^{i-1}h_j + a_{k-1}^{i-1}h'_j$
5: $c_j^i = c_j^{i-1} + b_{2i-2}a_j^{i-1}$
6: $d_j^i = d_j^{i-1} + b_{2i-1}a_j^{i-1}$
7: $q_j^i = q_j^{i-1} + a_{2i-2}a_j^{i-1}$
8: $r_j^i = r_j^{i-1} + a_{2i-1}a_j^{i-1}$
9: **end for**
10: **end for**
11: $d_{-1}^f \leftarrow 0, r_{-1}^f \leftarrow 0$
12: **for** $0 \leqslant j \leqslant k-1$ **do**
13: $p_j = c_j^g + d_{k-1}^f h_j + d_{j-1}^f$
14: $s_j = q_j^g + r_{k-1}^f h_j + r_{j-1}^f$
15: **end for**

4. Algorithm dependency graph

Fig. 1 indicates the extracted dependency graph (DG) from the bit-level algorithm, Algorithm 2, for $k = 5$. The DG is represented in the 2D space with the row index i and column index j. The light red nodes (circles) of the DG represent the operation steps 4–8 of Algorithm 2, while light blue nodes represent the operation steps 13 and 15 of the same algorithm. The upper g rows of the DG compute the partial bits of the variables A, C, D, Q, R according to steps 4–8 of Algorithm 2. The last row computes the resulting bits of the output products P and S according to steps 13 and 15 of Algorithm 2.

The inputs at the top of the DG are the initial bits $a_j^0, c_j^0, d_j^0, q_j^0, r_j^0, h_j$ and h'_j of variables A, C, D, Q, R, H, H', respectively. In the upper g rows, the vertical lines represent the intermediate bit values of $c_j^i, d_j^i, q_j^i, r_j^i, h_j$, and h'_j, while the slanted red lines represent the intermediate bit values of a_j^i. Also, in the upper g rows of the DG, the resulted intermediate bit values of $a_{k-2}^{i-1}, a_{k-1}^{i-1}$ as well as the input bits of $a_{2i-2}, a_{2i-1}, b_{2i-2}, b_{2i-1}$ are represented by the horizontal lines. The produced bit values $c_j^g, d_j^f, q_j^g, r_j^f$ from the upper g rows beside the broadcasted bits of h_j are used as inputs to the last row of the DG to produce the final bit values p_j and s_j of the output products P and R, respectively, as indicated in Fig. 1.

The unified and scalable processor core that concurrently performs both the multiplication and squaring operations can be extracted from the DG by choosing a proper non-linear scheduling and projection functions, as explained in [25]. The scheduling function assigns a time value to each node (circle) of the DG. In contrast, the projection function maps several DG nodes to a corresponding processing element (PE) in the systolic/semi-systolic array block of the processor core.

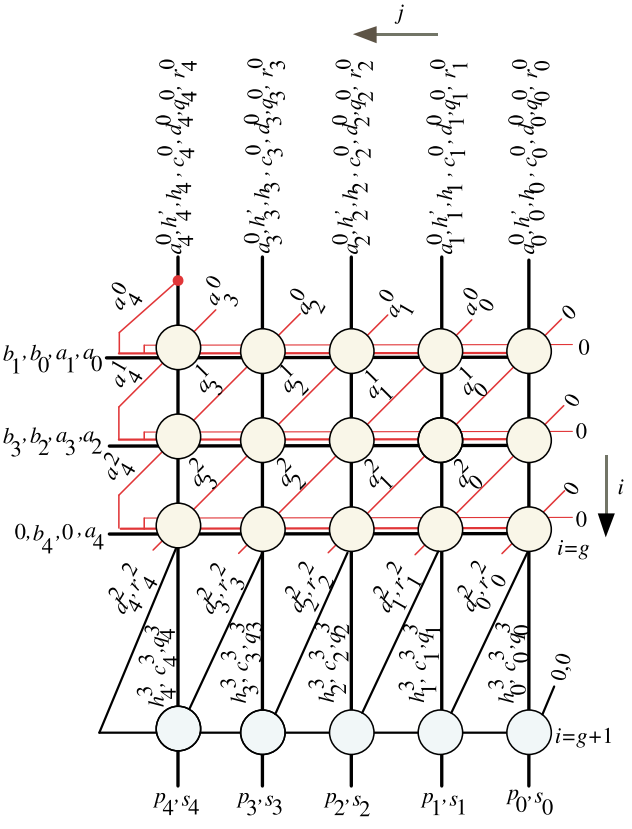

Fig. 1 DG of the unified bipartite algorithm for $k = 5$.

5. Proposed unified and scalable word-serial multiplier-squarer processor core

By following the approach discussed in [25], we can choose the following non-linear scheduling function to partition the DG space, composed of k columns, into l equitemporal zones.

$$F(\mathbf{N}) = (i-1)\left\lceil \frac{k}{l} \right\rceil + \left\lfloor \frac{k-1-j}{l} \right\rfloor + 1 \tag{14}$$

where $F(\mathbf{N})$ is the time assigned to a node $\mathbf{N}(i,j)$ in the DG, $1 \leqslant i \leqslant k$ and $-\gamma \leqslant j \leqslant k-1$. Where γ is the number of the added extra columns at the rightmost side of the DG as will be discussed below.

Fig. 2 shows the equitemporal zones (the light green zones) resulted from applying the scheduling function of Eq. (14) to the DG. This figure indicates the node timing or scheduling time for the case when $k = 5$ and $l = 3$. The time index inside each zone represents the execution time of the constituting processing nodes. When the number of the DG columns k is not a multiple integer of l, γ extra columns should be added to the rightmost side of the DG. The value of γ can be calculated as $\gamma = l\lceil \frac{k}{l} \rceil - k$. The added γ columns lead to right padding the variables A, H, H', C, D, Q, and R by γ zeros. For the case when $k = 5$ and $l = 3, \gamma$ will equal to one and thus only one more column should be added at the rightmost side of the DG as shown in Fig. 2. In this case, the input variables A, H, H' should be expressed as:

$$A = [\, a_k \quad \cdots \quad a_1 \quad a_0 \quad a_{-1} \quad a_{-2} \quad 0 \,] \tag{15}$$

$$H = [\, h_{k-1} \quad \cdots \quad h_3 \quad h_2 \quad h_1 \quad h_0 \quad 0 \,] \tag{16}$$

$$H' = [\, h'_{k-1} \quad \cdots \quad h'_3 \quad h'_2 \quad h'_1 \quad h'_0 \quad 0 \,] \tag{17}$$

The chosen non-linear scheduling function, Eq. (14), has the advantage of making us able to control processor workload (number of processing elements working at the same time) per time instance. Also, it has the advantage of managing the total number of time instances needed to perform the whole computation of the multiplier-squarer. The workload in our case is equal to l and the total number of time instances required to perform the whole computation can be determined by the following formula.

$$\#\text{Time Instances} = (g+1)\left\lceil \frac{k}{l} \right\rceil \tag{18}$$

By observing the node timing in Fig. 2, we notice that only l nodes are working at any given time. Thus, we can follow the approach discussed in [25] to extract the following non-linear projection function to map a node $\mathbf{N}(i,j) \in \mathbb{D}$ of Fig. 2 to a node $\overline{\mathbf{N}}(o,m)$ in the systolic/semi-systolic array space:

$$\overline{\mathbf{N}}(o,m) = \mathbf{N}_{siso}\ \mathbf{N}(i,j) \tag{19}$$

$$o = i \tag{20}$$

$$m = k - 1 - j \bmod l \tag{21}$$

$$\mathbf{N}_{siso} = [\,1 \quad . \bmod l\ ] \tag{22}$$

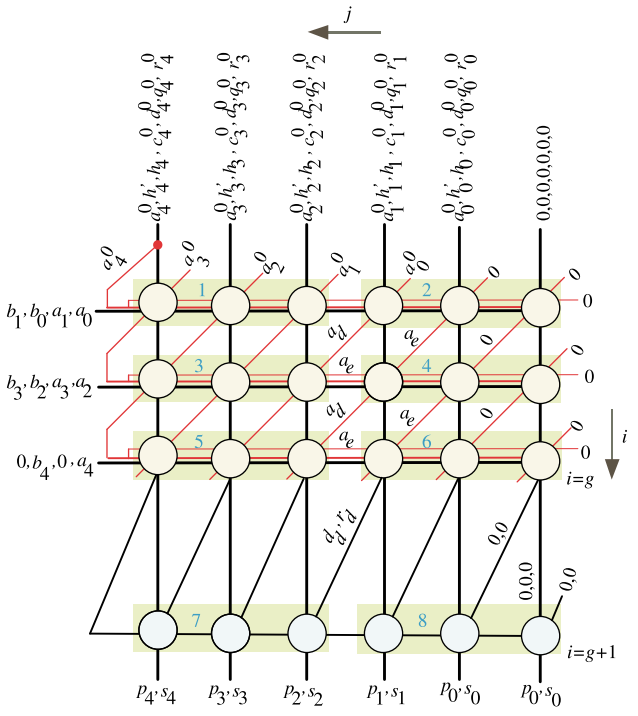

Fig. 2 Scheduling time for the combined multiplication-squaring operation for the case when $k = 5$ and $l = 3$.

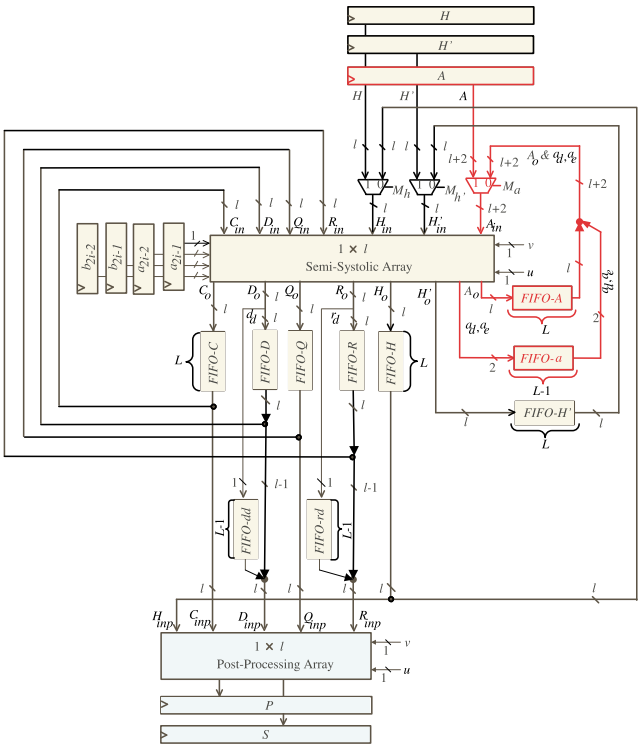

Fig. 3 Scalable word-serial semi-systolic multiplier-squarer processor core.

where "dot" is a place holder for the argument [25].

Fig. 3 displays the resulted scalable word-serial semi-systolic multiplier-squarer processor core after applying the previously extracted projection function, Eq. (19), to the nodes of Fig. 2. The resulted processor core composes of the main semi-systolic array block and the post-processing array block besides some FIFO buffers and I/O registers as well as three 2-to-1 Multiplexers (MUXes). Both the main semi-systolic array and the post-processing array blocks consist of PEs arranged in one-dimensional array of one row and l columns. The MUXes are used to select between the input words of variables A, H, H' and their intermediate word values. The FIFO buffers are used to sequentially fed the computed intermediate words of C, D, Q, R, A, H and H' to the inputs of the semi-systolic array block through the different computation cycles. As we notice, the FIFO of variable A is divided into two FIFO buffers: FIFO-A and FIFO-a. This is attributed to that the intermediate results of bits a_d, a_e, shown in Fig. 2, are fed to their neighbour nodes after delayed by $L-1$ time instances, $L = \lceil \frac{k}{l} \rceil$, while the remaining bits are fed to their neighbour nodes after delayed by L time instances. Also, for the same reason, the FIFOs of variables D and R are divided into two FIFO buffers: (FIFO-D, FIFO-dd) and (FIFO-R, FIFO-rd). All FIFO buffers displayed in Fig. 3 have the same width and depth sizes of l bits and L storage elements, respectively, except FIFO-a, FIFO-dd and FIFO-rd. FIFO-a has a fixed width-size of 2 bits and a depth-size of $L-1$. FIFO-dd and FIFO-rd have a fixed width-size of 1 bit and a depth-size of $L-1$.

Fig. 4 shows the structure of the semi-systolic array block for word-size $l = 3$. The PEs (the light orange PEs) of the semi-systolic array are similar except the leftmost one (the dark orange PE). Figs. 5 and 6 show the logic details of the leftmost PE and the remaining PEs of the semi-systolic array, respectively. As we notice, the leftmost PE is slightly different from the remaining PEs. It has two extra tri-state buffers controlled by the control signal u to pass the intermediate bits of a_{k-1}^{i-1} and a_{k-2}^{i-1} to the light orange PEs at time instances $n = (i-1)\lceil \frac{k}{l} \rceil + 1, 1 \leqslant i \leqslant g$. These bits besides the input bits of $a_{2i-1}, a_{2i-2}, b_{2i-1}$, and b_{2i-2} are used inside each PE to update the intermediate words of variables C, D, Q, R, and A.

Fig. 7 shows the structure of the post-processing array block for the word-size $l = 3$. Similar to the semi-systolic array block, the PEs (the light blue PEs) of the post-processing array are similar except the leftmost one (the dark blue PE). Figs. 8 and 9 show the logic details of the leftmost PE and the remaining PEs of the post-processing array, respectively. As we notice, the leftmost PE (the dark blue PE) is slightly different from the remaining PEs (the light blue PEs) of the post-processing array. It has two extra tri-state buffers controlled by the control signal u to pass the intermediate bits of r_{k-1}^{f} and d_{k-1}^{f} to the light blue PEs at time instances $n = (g)\lceil \frac{k}{l} \rceil + 1$. These bits are used inside all the PEs to compute the final product words of P and S. Tri-State buffers T_c, T_d, T_q and T_r shown in Figs. 8 and 9 pass the produced values of c_j^g, d_j^f, q_j^g and r_j^f from the semi-systolic array block to the post-processing array at the proper time. d_j^f, r_j^f are passed one time step earlier than c_j^g, q_j^g if $g \neq f$ (i.e., k has odd value).

For generic k and l values, we can summarize the operation of the scalable word-serial semi-systolic processor core as follows:

1. Through time periods, $1 \leqslant n \leqslant \lceil \frac{k}{l} \rceil$, the select signals of MUXes $M_A, M_{h'}$, and M_h are set to sequentially transfer the input words A, H', and H (starting from the most

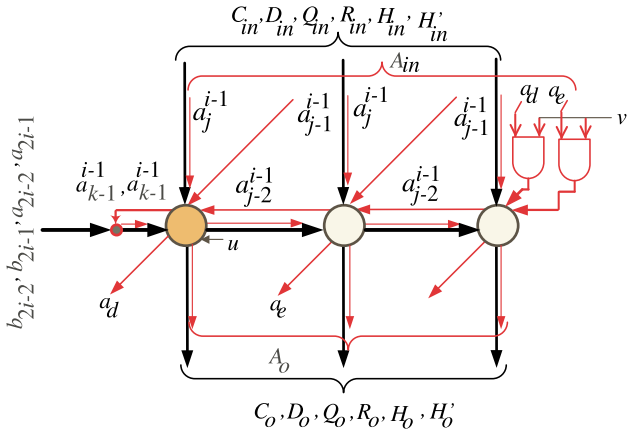

Fig. 4 Semi-systolic array structure for $l = 3$.

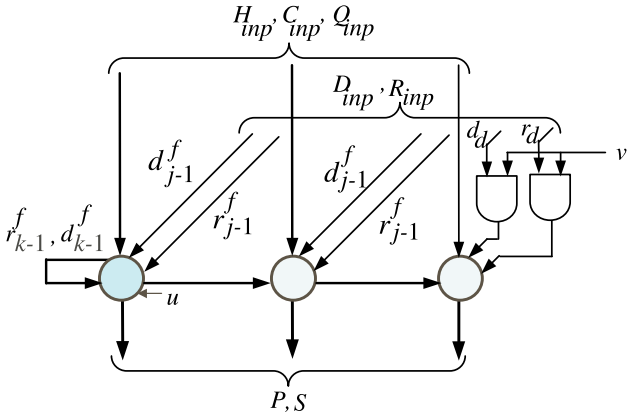

Fig. 5 Dark orange PE logic details of the semi-systolic array block.

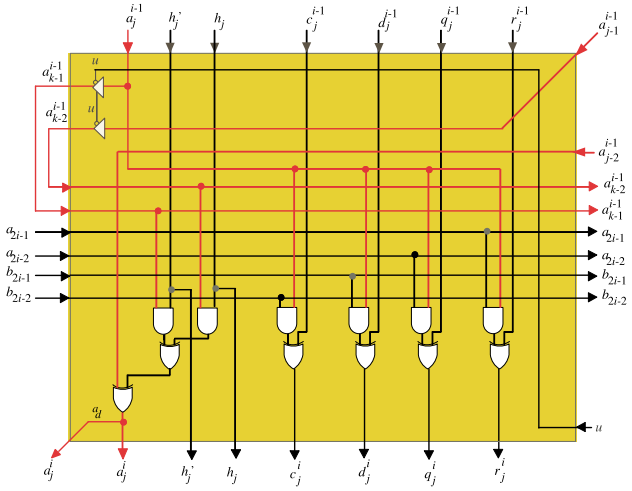

Fig. 6 Light Orange PE logic details the semi-systolic array.

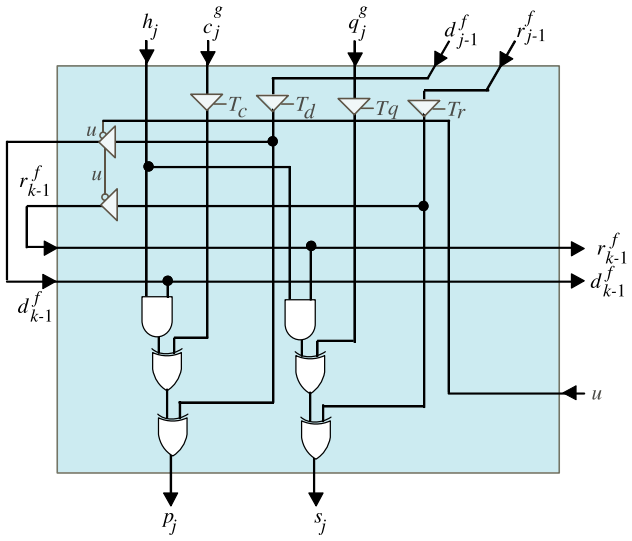

Fig. 7 Scalable post-processing array structure for $l = 3$.

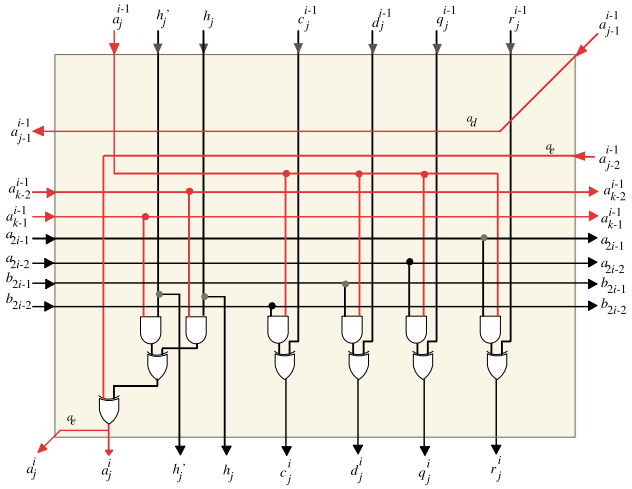

Fig. 8 Dark blue PE logic details of the post-processing array.

significant word) to inputs A_{in}, H'_{in}, and H_{in} of the semi-systolic array block, respectively. Also, through these execution times, FIFO buffers FIFO-C, FIFO-D, FIFO-Q, and FIFO-R are cleared to sequentially transfer zero words to inputs C_{in}, D_{in}, Q_{in}, and R_{in} of the semi-systolic array block. These zero words represent the initial values of variables C, D, Q, R as indicated in Algorithm 2. The depth of FIFO buffers assures storing the initial zero words through these time periods. Moreover, the bits of $a^0_{k-1}, a^0_{k-2}, a_0, a_1, b_0$, and b_1 are broadcasted horizontally to all PEs of the semi-systolic array block to be used alongside the previously mentioned inputs to sequentially compute the intermediate words of C, D, Q, R and A. The outputs of the semi-systolic array block are pipelined through FIFO buffers FIFO-C, FIFO-D, FIFO-Q, FIFO-R, FIFO-A, and FIFO-a, respectively, as shown in Fig. 3.

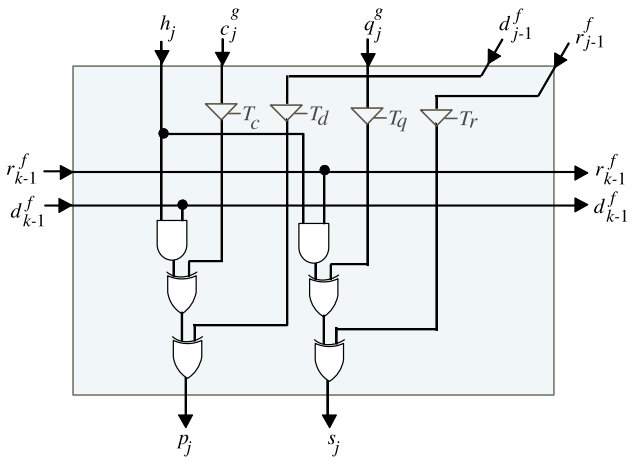

Fig. 9 Light blue PE logic details of the post-processing array.

2. Through time periods, $\lceil \frac{k}{l} \rceil < n < (g)\lceil \frac{k}{l} \rceil + 1$, the select signals of MUXes $M_A, M_{h'}$ and M_h are deactivated to sequentially transfer the updated A words stored in FIFOs (FIFO-A and FIFO-a) as well as the words of H' and H stored in FIFO-H′ and FIFO-H, respectively, to the inputs of the semi-systolic array block. These words alongside the updated words C, D, Q, R, stored in FIFO-C, FIFO-D, FIFO-Q and FIFO-R, as well as the broadcasted bits of $a_{k-1}^{i-1}, a_{k-2}^{i-1}, a_{2i-2}, a_{2i-1}, b_{2i-2}$ and $b_{2i-1}, 1 < i \leqslant l$, are used to sequentially compute the intermediate words of C, D, Q, R, and A.
3. At time periods $n = (i-1)\lceil \frac{k}{l} \rceil + 1, 1 \leqslant i \leqslant g$, input bits $a_{2i-2}, a_{2i-1}, b_{2i-2}$ and b_{2i-1} are sequentially transferred through the D-FFs shown in Fig. 3, to the corresponding inputs of the semi-systolic array bock. Also, through these time periods, the control signal u is enabled ($u = 0$) to pass the computed bits of a_{k-1}^{i-1} and $a_{k-2}^{i-1}, 1 \leqslant i \leqslant k$, through the tri-state buffers shown in Fig. 5. These computed bits alongside the input bits $a_{2i-2}, a_{2i-1}, b_{2i-2}, b_{2i-1}$ are horizontally broadcasted to all the PEs of the semi-systolic array through these execution periods.
4. Through time periods $n = i\lceil \frac{k}{l} \rceil, 1 \leqslant i \leqslant g$, the control signal v is activated ($v = 0$) to enforce the least significant two bits of operand A to have zero value as indicated at the rightmost edge of the activity graph shown in Fig. 2. This is done through the AND gates shown in Fig. 4. Through the remaining time periods, this control signal is deactivated ($v = 1$) to transfer a_d and a_e signals to the rightmost PE of the semi-systolic array indicated in Fig. 4.
5. At time period $n = (g)\lceil \frac{k}{l} \rceil + 1$, the common control signal of the tri-state buffers, shown in Fig. 8, is activated ($u = 0$) to broadcast the updated bits of r_{k-1}^f and d_{k-1}^f to all the PEs of the post-processing array shown in Fig. 7. These bits besides the remaining input bits of the post-processing array are used compute the final words of the products P and S.
6. At time periods, $n = (g+1)\lceil \frac{k}{l} \rceil$, the control signal v is activated ($v = 0$) to enforce the least significant bits of operands D and R to have zero value as indicated at the rightmost edge of the last row of the activity graph shown in Fig. 2. This is done through the AND gates shown in Fig. 7. Through the remaining time periods, this control signal is deactivated ($v = 1$) to transfer d_d and r_d signals to the rightmost PE of the post-processing array indicated in Fig. 7.
7. Through time periods $(g)\lceil \frac{k}{l} \rceil + 1 \leqslant n \geqslant (g+1)\lceil \frac{k}{l} \rceil$ the resulted output words of P and S will be loaded sequentially, word-by-word, in registers P and S, respectively, as shown in Fig. 3.

6. Complexities comparison

In this part, we compare the hardware and delay complexities as well as the consumed energy of the presented word-serial unified and scalable multiplier-squarer and the existing efficient word-serial multipliers reported in [11,17,32,33]. The hardware complexity (area) is estimated based on the number of the basic logic gates/components constituting the hardware structure of each design. The following gates/components are most common in all the compared designs: Tri-State buffers, 2-input AND gate, 2-input XOR gate, 2-input Multiplexers, and Flip-Flops. The delay complexity is estimated in terms of the total latency (number of clock cycles required to produce the product) and the circuit critical path delay (CPD).

Table 1 displays the estimated hardware and delay complexities of the compared designs. The following explains the notations used in this table:

- k: field size
- l: operands word-size
- D_A: propagation delay of the 2-input AND gate.
- D_X: propagation delay of the 2-input XOR gate.
- D_{MUX}: propagation delay of 1-bit 2-to-1 MUX.
- $F_1 = 7k + k(\lceil \log k \rceil) + l + 3$
- $F_2 = 2l^2 + 2l(\lceil k/l \rceil) + 4l + 1$
- $F_3 = 2l^2 + 3l(\lceil k/l \rceil) + 2l$
- $L_1 = 2l + 2\lceil k/l \rceil^2 + 2\lceil k/l \rceil$
- $\tau_1 = D_A + (\lceil \log_2 l \rceil + 1)D_X$
- $\tau_2 = D_A + 2D_X$
- $\tau_3 = D_A + D_X$

To obtain a fair comparison between the compared structures, the estimated number of Flip-Flops for each design should include the number of Flip-Flops of the I/O registers.

From Table 1, we notice the following: (1) The area complexity of all the gates/components of the multiplier of Xie [17] is of order $\mathcal{O}(kl)$ and the delay complexity is of order $\mathcal{O}(\lceil k/l \rceil)$. (2) The area complexities of AND gates, XOR gates, and Flip-Flops of the multiplier of Pan [11] are of orders $\mathcal{O}\left(k\sqrt{k}\right), \mathcal{O}\left(k\sqrt{kl}\right)$, and $\mathcal{O}(k)$, respectively, while the delay complexity is of order $\mathcal{O}\left(\sqrt{k/l}\right)$. (3) The area complexities of all the gates/components of the multipliers of Hua [32] and Chen [33] are of order $\mathcal{O}(l^2)$ and their delay complexities are of order $\mathcal{O}(\lceil k/l \rceil)^2$. 4) The proposed multiplier-squarer has area complexity of order $\mathcal{O}(l)$ for all the logic gates/components except the Flip-Flops have area complexity of order $\mathcal{O}(\lceil k/l \rceil)$. The delay complexity of the proposed multiplier-squarer is of order $\mathcal{O}(\lceil k/l \rceil)$.

Table 1 Area and delay complexities comparison between the different word-serial field multipliers.

Design	Tri-State	AND	XOR	MUXs	Flip-Flops	Latency	CPD
Xie [17]	0	kl	$kl+3k-3\frac{k}{l}+3$	0	$2kl+2k+l$	$2\lceil k/l\rceil+2\lceil\log_2 l\rceil$	$2D_X$
Pan [11]	0	$k\sqrt{k}$	$\sqrt{kl}(2+k)+l$	0	F_1	$2\lceil\sqrt{k/l}\rceil$	τ_1
Hua [32]	0	l^2	$l^2+4-5l+1^{(1)}$	0	F_2	$6l\lceil k/l\rceil^2$	τ_2
Chen [33]	0	l^2+l	l^2+2l	$2l^{(2)}$	F_3	L_1	τ_3
Proposed	$8l$	$6l+4$	$6l$	$3l+2$	$6l\lceil k/l\rceil+4(\lceil k/l\rceil)$	$(g+1)\lceil k/l\rceil$	τ_2

(1) The estimated hardware complexity of the 3-input XOR gate is 1.5 times that of the 2-input XOR gate.
(2) The switches used in Multiplier of [33] have the same number of transistor as the 1-bit 2-to-1 MUX.

Based on the order of area and delay complexities and for the recommended filed size $k = 409$ and the embedded word-sizes of $l = 8, l = 16, l = 32$, we can expect that the proposed multiplier-squarer achieves lower area complexity and a reasonable delay complexity, in terms of the counted logic gates/components, compared to the other multipliers. Table 1 does not include the area and delay complexities of the interconnecting wires that are difficult to estimate without CAD tools. Therefore, to have more accurate and fair results, we should perform practical implementations for the compared multipliers.

We modeled all the compared designs using the VHDL hardware description language and synthesized them for the NIST recommended field-size of $k = 409$ and distinct values of the embedded word sizes l $(8, 16, 32)$. We used Synopsys tools version 2005.09-SP2 with the NanGate (15 nm, 0.8 V) Open Cell Library to synthesis the modeled designs. We used the typical corner (V_{DD} = 0.8 V and $T_j = 25°C$) and unit drive strength for all the utilized primitives. Table 2 shows the produced synthesis results for all the compared designs. The obtained results include the design area (A) in terms of the equivalent numbers of the 2-input NAND gates (Kgates), the critical path delay (CPD) in *ps*, and the consumed power in *nW*. The remaining design metrics added to Table 2 are computed as follows: the total computation time (T) is calculated as the product of Latency (total number of clock cycles required to produce the output results) and the obtained CPD values. The consumed energy (E) is computed as the product of the obtained consumed power (P) and the computed computation time (T).

The proposed unified multiplier-squarer structure performs both multiplication and squaring operations concurrently, while the compared multiplier structures of [11,17,32,33] perform only the multiplication operation. For a fair comparison between the proposed unified multiplier-squarer structure and the multiplier structures of [11,17,32,33], they should be run twice to perform both operations. This will lead to duplicating their time and consumed energy, as indicated in Table 2. In spite of performing both multiplication and squaring operations, the proposed design has a reasonable area compared to most of the other designs for different word sizes l. For $l = 32$, it has a lower area compared to all other designs by at least 40.13%. This is attributed to the significant reduction of the number of Flip-Flops of the proposed design as it significantly decreases as the word size increases, as indicated in Table 1. Also, this accounts for the small variations in the total area of the proposed design at the different word sizes. To clarify this point more, as we notice in Table 1 all the logic gates/components of the proposed design are directly proportional to the word size l except the Flip-Flops are inversely proportional to l. This means that any increase in the number of the logic gates/components is offset with a decrease in the number of Flip-Flops.

Table 2 Implementation results of different word-serial field multipliers for $k = 409$ and different values of l.

Multiplier	l	Latency	Area (A) [Kgates]	CPD [ps]	Time (T) [ns]	AT	power (P) [nW]	Energy (E) [fJ]
Xie [17]	8	324	92.98	50.8	16.46	1530.5	225.56	3.71
	16	172	146.96	50.8	8.74	1284.4	375.5	3.28
	32	98	195.13	50.8	4.98	971.8	477.4	2.38
Pan [11]	8	48	97.46	206.3	9.90	964.9	252.91	2.50
	16	36	123.93	244.4	8.80	1090.6	320.07	2.82
	32	24	164.34	282.5	6.78	1114.2	425.09	2.88
Hua [32]	8	259584	7.99	73.4	19053.47	152237.2	4.35	82.88
	16	129792	10.40	73.4	9526.73	99077.9	5.85	55.73
	32	64896	19.91	73.4	4763.37	94838.7	11.15	53.11
Chen [33]	8	11946	10.16	55.2	659.42	6699.7	5.11	3.37
	16	4678	13.51	55.2	203.03	3488.7	8.38	1.70
	32	1572	26.58	55.2	86.77	2306.4	15.95	1.38
Proposed	8	10712	11.87	49.1	525.96	6248.4	5.25	2.76
	16	5356	11.67	49.1	262.98	3068.9	5.21	1.37
	32	2678	11.92	49.1	131.49	1567.4	5.28	0.69

Due to the reduction of the area of the proposed design at most of the word sizes l, it has lower power consumption except at $l = 8$ where the design of Pan [11] has a slightly lower area. This saving in area leads to reducing the extracted parasitic capacitances and hence reducing the switching activities, one of the main contributors of the dynamic power consumption. The significant reduction of power consumption of the proposed design leads to significant savings in the consumed energy, as indicated in Table 2. Despite designs of Pan [11] and Xie [17] have a significant reduction in total computation time compared to the proposed design, they have higher consumed energy at most values of l compared to the proposed design except at $l = 8$ where the design of Pan [11] has a slightly lower consumed energy. The increase in the consumed energy of these designs over the proposed design is attributed to the significant increase in their area and hence their consumed power. At $l = 16$ and $l = 32$, the proposed design has a higher saving of consumed energy over all the compared designs by at least 19.4% and 50%, respectively. Therefore, we can conclude that the proposed design outperforms the compared designs in terms of area and consumed energy at $l = 32$. This makes it more suitable for implementing resource-constrained cryptographic primitives in all embedded applications operating at word-size of 32 bits.

7. Summary and conclusion

This paper introduced a competent word-serial unified and scalable semi-systolic multiplier-squarer processor core. It has the advantage of simultaneously computing both multiplication and squaring operations over $GF(2^k)$. It also shares the data-path between the two operations, making it more effective in saving hardware and power resources. The design's scalability provides the designer with higher flexibility to manage the processor data-path size and its computational time. The achieved synthesis results of the proposed word-serial unified structure and the efficient existing word-serial ones indicate that the proposed structure outperforms the compared designs in terms of area and consumed energy at $l = 32$. Thus, the proposed architecture is more appropriate for implementing cryptographic primitives in all resource-constrained embedded applications operating at this word-size.

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.

Acknowledgements

The author would like to acknowledge the support of the Deanship of Scientific Research at Prince Sattam Bin Abdulaziz university under the research project # 2020/01/16466.

References

[1] R.L. Rivest, A. Shamir, L. Adleman, A method for obtaining digital signatures and public-key cryptosystems, Mag. Commun. ACM 21 (2) (1978) 120–126.
[2] R. Lidl, H. Niederreiter, Introduction to Finite Fields and their Applications, Cambridge University Press, Cambridge, UK, 1994.
[3] S. Choi, K. Lee, Efficient systolic modular multiplier/squarer for fast exponentiation over GF(2^m), IEICE Electron. Express 12 (11) (2015) 1–6.
[4] K.W. Kim, S.H. Kim, Efficient bit-parallel systolic architecture for multiplication and squaring over GF(2^m), IEICE Electron. Express 15 (2) (2018) 1–6.
[5] K.W. Kim, J.D. Lee, Efficient unified semi-systolic arrays for multiplication and squaring over GF(2^m), IEICE Electron. Express 14 (12) (2017) 1–10.
[6] C.-W. Chiou, C.-Y. Lee, A.-W. Deng, J.-M. Lin, Concurrent error detection in montgomery multiplication over GF(2^m), IEICE Trans. Fundam. Electron. Commun. Comput. Sci. E89-A (2) (2006) 566–574.
[7] K.W. Kim, J.C. Jeon, Polynomial basis multiplier using cellular systolic architecture, IETE J. Res. 60 (2) (2014) 194–199.
[8] C.H. Kim, C.P. Hong, S. Kwon, A digit-serial multiplier for finite field GF(2^m), IEEE Trans. Very Large Scale Integr. (VLSI) Sys. 13 (4) (2005) 476–483.
[9] S. Talapatra, H. Rahaman, J. Mathew, Low complexity digit serial systolic montgomery multipliers for special class of GF (2^m), IEEE Trans, Very Large Scale Integr. (VLSI) Sys. 18 (5) (2010) 847–852.
[10] J.H. Guo, C.L. Wang, Hardware-efficient systolic architecture for inversion and division in GF(2^m), IEE Proc. Comput. Digital Tech. 145 (4) (1998) 272–278.
[11] J.S. Pan, C.Y. Lee, P.K. Meher, Low-latency digit-serial and digit-parallel systolic multipliers for large binary extension fields, IEEE Trans. Circ. Sys.-I 60 (12) (2013) 3195–3204.
[12] C.Y. Lee, C.C. Fan, S.M. Yuan, New digit-serial three-operand multiplier over binary extension fields for high-performance applications, in: Proc. 2017 2nd IEEE International Conference on Computational Intelligence and Applications, 2017, pp. 498–502.
[13] A.H. Namin, H. Wu, M. Ahmadi, A word-level finite field multiplier using normal basis, IEEE Trans. Comput. 60 (6) (2011) 890–895.
[14] 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.
[15] S. Kumar, T. Wollinger, C. Paar, Optimum digit serial multipliers for curve-based cryptography, IEEE Trans. Comput. 55 (10) (2006) 1306–1311.
[16] C.Y. Lee, Super digit-serial systolic multiplier over GF(2^m), in: Proc. 6th Int. Conf. Genetic Evolutionary Computing, Kitakyushu, Japan, 2012, pp. 509–513.
[17] J. Xie, P.K. Meher, Z. Mao, Low-latency high-throughput systolic multipliers over GF(2^m) for NIST recommended pentanomials, IEEE Trans. Circ. Syst. 62 (3) (2015) 881–890.
[18] C.-Y. Lee, C.W. Chiou, J.M. Lin, C.C. Chang, Scalable and systolic montgomery multiplier over generated by trinomials, IET Circuits, Devices Syst. 1 (6) (2007) 477–484.
[19] L.H. Chen, P.L. Chang, C.-Y. Lee, Y.K. Yang, Scalable and systolic dual basis multiplier over GF(2^m), Int. J. Innov. Comput. Inform. Control 7 (3) (2011) 1193–1208.
[20] G. Orlando, C. Paar, A super-serial galois fields multiplier for FPGAs and its application to public-key algorithms, in: Proc. IEEE Symp. Field-Programm. Custom Comp., 1999, pp. 232–239.
[21] S. Bayat-Sarmadi, M.M. Kermani, R. Azarderakhsh, C.-Y. Lee, Dual basis super-serial mult. for secure applications and lightweight cryptographic arch, IEEE Trans. Circ. Sys.-II 61 (2) (2014) 125–129.
[22] F. Gebali, A. Ibrahim, Efficient scalable serial multiplier over GF(2^m) based on trinomial, IEEE Trans. Very Large Scale Integr. VLSI Syst. 23 (10) (2015) 2322–2326.
[23] A. Ibrahim, F. Gebali, H. El-Simary, A. Nassar, High-performance, low-power architecture for scalable radix 2

montgomery modular multiplication algorithm, IEEE Canadian J. Electr. Comput. Eng. 34 (4) (2009) 152–157.
[24] A. Ibrahim, F. Gebali, Scalable and unified digit-serial processor array architecture for multiplication and inversion over GF(2^m), IEEE Trans. Circuits Syst. I Regul. Pap. 22 (11) (2017) 2894–2906.
[25] F. Gebali, Algorithms and Parallel Computers, John Wiley, New York, USA, 2011.
[26] A. Ibrahim, H. Elsimary, F. Gebali, New systolic array architecture for finite field division, IEICE Electron. Express 15 (11) (2018) 1–11.
[27] A. Ibrahim, H. Elsimary, A. Aljumah, F. Gebali, Reconfigurable hardware accelerator for profile hidden markov models, Arab. J. Sci. Eng. 41 (8) (2016) 3267–3277.
[28] A. Ibrahim, Scalable digit-serial processor array architecture for finite field division, Microelectron. J. 85 (2019) 83–91.
[29] A. Ibrahim, T. Alsomani, F. Gebali, Unified systolic array architecture for field multiplication and inversion over GF(2^m), Comput. Electr. Eng. J. 61 (2017) 104–115.
[30] A. Ibrahim, T. Alsomani, F. Gebali, New systolic array architecture for finite field inversion, IEEE Can. J. Electr. Comput. Eng. 40 (1) (2017) 23–30.
[31] 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.
[32] Y.Y. Hua, J.M. Lin, C.W. Chiou, C.Y. Lee, Y.H. Liu, Low space-complexity digit-serial dual basis systolic multiplier over GF(2^m) using hankel matrix and karatsuba algorithm, IET Information Security 7 (2) (2013) 75–86.
[33] C.-C. Chen, C.-Y. Lee, E.-H. Lu, Scalable and systolic Montgomery multipliers over GF(2^m), IEICE Trans. Fundam. Electron. Commun. Comput. Sci. E91-A (7) (2008) 1763–1771.