Word-based processor structure for Montgomery modular multiplier suitable for compact IoT edge devices

Ibrahim, A. & Gebali, F.

2023

mathematics

MDPI

Article

Word-Based Processor Structure for Montgomery Modular Multiplier Suitable for Compact IoT Edge Devices

Atef Ibrahim [1,2,*] and Fayez Gebali [2]

[1] Computer Engineering Department, College of Computer Engineering and Sciences, Prince Sattam Bin Abdulaziz University, Al-Kharj 16278, Saudi Arabia
[2] Electrical and Computer Engineering Department, University of Victoria, Victoria, BC V8P 5C2, Canada
* Correspondence: aa.mohamed@psau.edu.sa

Abstract: The Internet of Things (IoT) is an emerging technology that forms a huge network of different objects and intelligent devices. IoT Security is becoming more important due to the exchange of sensitive sensor data and the potential for incorporating the virtual and real worlds. IoT edge devices create serious security threats to network systems. Due to their limited resources, it is challenging to implement cryptographic protocols on these devices to secure them. To address this problem, we should perform compact implementation of cryptographic algorithms on these devices. At the heart of most cryptographic algorithms is the modular multiplication operation. Therefore, efficient implementation of this operation will have a great impact on the implementation of the whole cryptographic protocol. In this paper, we will focus on the resource and energy efficient hardware implementation of the adopted Montgomery modular multiplication algorithm over GF(2^m). The main building block of the proposed word-based processor structure is a processor array that has a modular structure with local connectivity between its processing elements. The ability to manage the saving amounts of area, delay, and consumed energy is the main benefit of the suggested hardware structure. We used ASIC technology to implement the suggested word-based processor structure. The final results show an average reduction in the area of 86.3% when compared with the competitive word-based multiplier structures. Additionally, the recommended design achieves significant average savings in area-time product, power, and consumed energy of 53.7%, 83.2%, and 72.6%, receptively, over the competitive ones. The obtained results show that the provided processor structure is best suited for application in compact IoT edge devices with limited resources.

Keywords: montgomery modular multiplication; IoT security; crypto-processors; modular arithmetic; IoT networks; processor arrays; hardware security; cyber-physical systems

MSC: 11T71

Citation: Ibrahim, A.; Gebali, F. Word-Based Processor Structure for Montgomery Modular Multiplier Suitable for Compact IoT Edge Devices. *Mathematics* **2023**, *11*, 328. https://doi.org/10.3390/math11020328

Academic Editor: Raúl M. Falcón

Received: 9 December 2022
Revised: 3 January 2023
Accepted: 5 January 2023
Published: 8 January 2023

1. Introduction

The term "Internet of Things" (IoT) refers to an extensive network that allows different objects and intelligent devices to communicate with one another. Sensing, processing, and data transmission are the three key IoT components. The latest IoT platform is currently utilized across a wide range of industries, including those in the environment, home, energy, telecommunications, healthcare, business, manufacturing, water management, and construction. IoT technology, which uses embedded devices, is distinct from the computer, laptop, and mobile device technology.

Despite IoT having many benefits, it has three primary issues: data collection, data transmission, and data security. Many sensing tools have been developed and tailored for IoT devices to collect data. The development and adaptation of numerous protocols have made it possible for IoT devices to connect to existing networks and exchange data. The data security issue is not given the consideration it deserves. As a result, the IoT is strongly

linked to several traditional and modern security issues, including authentication, authorization, and data security. Denial of service attacks, replay attacks, Denning–Sacco attacks, password guessing attacks, and many other attacks can all result from authentication flaws.

Security is becoming increasingly important for IoT systems due to the exchange of sensitive data produced by sensors and the potential for merging the physical and digital worlds. It is very difficult to authenticate IoT devices across heterogeneous and connected protocols. These protocols should also consider issues with IoT device limitations in energy consumption, memory space, and processing power [1].

The majority of IoT applications use smart-edge devices to sense, process, and transfer recorded data [2,3]. Additionally, the majority of research has previously been focused on the creation of IoT systems without taking security into account. To address the security concerns in IoT networks, numerous solutions have been proposed in the past few years [4–8]. However, the majority of currently employed methods primarily focus on software solutions at the application layer of the IoT framework. Additionally, they do not offer resource and energy-efficient hardware security solutions suitable for resource-constrained edge devices. Hardware implementation of cryptographic algorithms offers more reliable and effective security than software implementations.

A compact IoT edge device is a subcategory of an embedded device and implies that it has fewer important computational resources, storage, energy, and networking capabilities than a typical computer device, such as modern mobile phones, laptops, and desktops, which need to be efficient enough for multitasking. Actually, a significant part of IoT devices share the characteristics of being embedded and resource-constrained. Here are only a few samples of these gadgets: smart toys, coffee makers, fridges, activity trackers, controllers, and sensors.

The memory and processing power of compact IoT edge devices are limited. They also work in isolated areas and are powered by the environment or by tiny batteries [1]. Therefore, there is a crucial and pressing need to lower their energy consumption. Compact IoT edge devices have limited resources, which makes it difficult to implement cryptographic algorithms on them. As a result, the IoT platform is vulnerable to numerous security breaches because these devices do not use cryptographic algorithms [9].

To safeguard the platform and resolve IoT potential threats, security services such as integrity, secrecy, authentication, non-repudiation, and availability must be implied. The kind of cryptographic algorithm used and how effectively it is implemented on the small IoT edge device determine how effectively these security features are implemented. When compared to popular cryptographic algorithms such as RSA, Elliptic Curve Cryptography (ECC) is the preferred cryptographic algorithm to use in IoT devices with limited resources [10–12]. The choice of ECC over RSA is attributed to its high level of security and shorter key lengths. Basic modular arithmetic operations are the foundation of the ECC algorithm. At the heart of the ECC algorithm is the basic modular arithmetic operations. Modular multiplication is the basic operation for the other modular operations used in this algorithm, such as modular exponentiation and modular inversion. Therefore, effective implementation of this operation will result in effective implementation of the entire cryptographic algorithm.

There has been a lot of research on modular multiplication in both binary extension fields $GF(2^m)$ and prime fields $GF(p)$. As a result of their large area and time complexities, many of the documented multipliers are not suitable for compact IoT edge devices [13–16]. To overcome these restrictions, a lot of publications have recommended word-serial modular multipliers. There are two types of approaches used to implement the word-based modular multipliers: non-systolic and systolic approaches. The non-systolic approaches were explained in [17–20], while the systolic approaches were discussed in [21–25]. In order to conserve energy and space, other publications merged modular multiplication and modular squaring operations [14,15,26–29]. However, the obtained structures were inappropriate for compact IoT devices due to their large space and high power costs.

Most of the publicly released modular multiplier constructions are one-of-a-kind constructions. The system performance characteristics of latency, throughput, power, and space are enhanced using ad hoc procedures without any consideration of how the architecture might be altered. In [30], the author created a methodical approach for putting any regularly iterative algorithm into practice based on the principally recommended arithmetic strategy. Linear mappings were used in the formal procedure to build the hardware architecture of the iterative algorithm. Linear mappings, on the other hand, have limited functionality with regard to the number of processing elements (PE) and the scheduling strategies that can be used.

In order to obtain more flexible scheduling techniques and map the recursive Montgomery modular multiplication algorithm onto parallel PEs, this paper suggests using the non-linear methodologies previously reported in [30]. Therefore, the main goal of this work is to develop a word-serial processor structure for the adopted Montgomery modular multiplication that is reported in [29]. The suggested structure can be modified to fit the needs of small IoT edge devices with constrained resources. The quantitative findings indicate that the proposed multiplier outperforms the efficient word-serial ones, initially discussed in the literature, in terms of area and consumed energy. The obtained implementation results show that the proposed design achieves significant average savings in terms of area and energy by 86.3% and 72.6%, respectively. As a result, the suggested multiplier structure is most appropriate for use in small IoT edge devices with constrained resources.

The work should be organized in the manner listed below. The presumed Montgomery modular multiplication algorithm's algebraic foundations and its representation in the bit-level form are briefly explained in Section 2. The algorithm dependency graph is also displayed in this section. The word-serial processor architecture of the Montgomery modular multiplier was investigated using the approach shown in Section 3, which also offers details on the logic structure of the processor construction. The findings of the implementation are displayed in Section 4. The conclusion of the suggested work appears in Section 5.

2. Mathematical Foundation of the Montgomery Modular Multiplication Algorithm over GF(2^m)

Assume that the irreducible polynomial $F = \sum_{j=0}^{m} f_j \cdot \phi^j, 1 \leq j \leq m-1$ and $f_m = f_0 = 1$, is the one that generates the binary extension field GF(2^m). Polynomials with degrees less than m are used to create each individual linear combination that makes up GF(2^m). Bitwise exclusive-OR (XOR) can be employed to add two polynomials in GF(2^m). However, modular multiplication of two polynomials is a little more challenging because the partial results necessitate an additional modular reduction by $\phi^m = \sum_{j=0}^{m-1} f_i \cdot \phi^i$

Assume that the two of the elements of GF(2^m) that will be multiplied are ω and χ. A further assumption is that ρ is a special element meeting the required $gcd(\rho, F) = 1$ and that C and D are the Montgomery residues of ω and χ, respectively. The computation $P = CD\rho^{-1} \bmod F = \chi\rho \bmod F = \sum_{j=0}^{m-1} p_j \cdot \phi^j$ represents the Montgomery Modular Multiplication (MMM) of the two polynomials $C = \omega\rho \bmod F = \sum_{j=0}^{m-1} c_j \cdot \phi^j$ and $D = \chi\ \rho \bmod F = \sum_{j=0}^{m-1} d_j \cdot \phi^j$. The final result Q can then be calculated by performing Montgomery multiplication through using inputs P and 1, which is expressed as $Q = P\rho^{-1} \bmod F = \omega\ \chi \bmod F$. In applications that involve recurring multiplications, such as elliptic curve point multiplication, inversion, and exponentiation, Montgomery multiplication is desirable due to the need for pre- and post-transformation.

By using $\rho = \phi^{(m-1)/2}$, the Montgomery multiplication, $P = CD\rho^{-1} \bmod F$, can always be formulated in this manner.

$$\begin{aligned} P =& C(d_0 + d_1\phi + \cdots + d_{(m-1)/2}\phi^{(m-1)/2} + \\ & \cdots + d_{m-1}\phi^{m-1})\phi^{-(m-1)/2} \bmod F \end{aligned} \tag{1}$$

Equation (1) could be formed to appear as follows:

$$\begin{aligned} P =& C(d_{(m-1)/2} + d_{(m+1)/2}\phi^1 + \cdots + d_{(m-1)}\phi^{(m-1)/2}) \\ &+ C(d_0\phi^{-(m-1)/2} + d_1\phi^{-(m-3)/2} + \\ &\cdots + d^{-1}_{(m-3)/2}) \bmod F \end{aligned} \tag{2}$$

It is important to note that the majority of useful applications use odd m. As a result, when designing the multiplier, we will concentrate on the use of odd m.

Equation (2) can be divided into two polynomials A and B as shown below. The summation of A and B will constitute Equation (2).

$$\begin{aligned} A =& Cd_{(m-1)}\phi^{(m-1)/2} + Cd_{(m-2)}\phi^{(m-3)/2} + \cdots \\ &+ Cd_{(m+1)/2}\phi^1 + Cd_{(m-1)/2} \bmod F \end{aligned} \tag{3}$$

$$\begin{aligned} B =& Cd_0\phi^{-(m-1)/2} + Cd_1\phi^{-(m-3)/2} + \cdots \\ &+ Cd_{(m-5)/2}\phi^{-2} + Cd_{(m-3)/2}\phi^{-1} \bmod F \end{aligned} \tag{4}$$

To determine the iterative forms of Equations (3) and (4), we can organize them as:

$$\begin{aligned} A =& (\cdots((Cd_{(m-1)})\phi \bmod F + Cd_{(m-2)})\phi \bmod F + \cdots + \\ &Cd_{(m+1)/2})\phi \bmod F + Cd_{(m-1)/2} \end{aligned} \tag{5}$$

$$\begin{aligned} B =& (\cdots(((Cd_0)\phi^{-1} \bmod F + Cd_1)\phi^{-1} \bmod F + \cdots + \\ &Cd_{(m-5)/2}\phi^{-1} \bmod F + Cd_{(m-3)/2})\phi^{-1} \bmod F \end{aligned} \tag{6}$$

Let us say that A_i and B_i are the results of the $(i)^{th}$ recursion of Equations (5) and (6), which can be determined recursively from the results of the $(i-1)^{th}$ pair of iterators. The following is a representation of the recursive equation of (5) at step i for $1 \leq i \leq (m+1)/2$:

$$A_i = A_{i-1}\phi \bmod F + Cd_{(m-i)} \tag{7}$$

with $A_0 = 0$.

Similar to Equation (5), Equation (6) can also be represented recursively as:

$$B_i = B_{i-1}\phi^{-1} \bmod F + Cd_{(i-1)} \tag{8}$$

with $B_0 = d_{(m-1)/2} = 0$.

Be aware that the value $d_{(m-1)/2} = 0$ is required to calculate the final product $B_{(m+1)/2}$. Since A_i and B_i do not share any data, they can be calculated simultaneously, as demonstrated by Equations (7) and (8). By changing the expansion of ϕ^m on Equation (7), one can acquire the reduced form of A_i for $1 \leq i \leq (m+1)/2$ using the bit-level representation. As a result, A_i can be expressed at the bit level as follows:

$$\begin{aligned} A_i =& a^{i-1}_{m-2}\phi^{m-1} + \cdots + a^{i-1}_1\phi^2 + a^{i-1}_0\phi + \\ &a^{i-1}_{m-1}(f_{m-1}\phi^{m-1} + \cdots + f_1\phi + f_0) + \\ &d_{m-i}(c_{m-1}\phi^{m-1} + \cdots + c_1\phi + c_0) \end{aligned} \tag{9}$$

It is possible to write the iterative representation of A at step i as follows:

$$a^i_{m-1-j} = a^{i-1}_{m-2-j} + a^{i-1}_{m-1}f_{m-1-j} + d_{m-i}c_{m-1-j} \tag{10}$$

with $a^0_j = a^{i-1}_{-1} = 0$ and $0 \leq j \leq m-1$.

We can obtain $\phi^{-1} = \sum_{j=1}^{m} f_j \phi^{j-1}$ by multiplying each side of F by ϕ^{-1}. This is possible because ϕ is a root of F and $f_0 = f_m = 1$ for any irreducible polynomial.

B_i is rewritten in a manner similar to Equation (9) by swapping out the expansion of ϕ^{-1} in Equation (8):

$$\begin{aligned} B_i = & b_{m-1}^{i-1}\phi^{m-2} + \cdots + b_1^{i-1} + b_0^{i-1}(f_m\phi^{m-1} + \cdots + \\ & f_2\phi + f_1) + d_{i-1}(c_{m-1}\phi^{m-1} + \cdots + c_1\phi + c_0) \end{aligned} \tag{11}$$

It is possible to write the iterative representation of B at step i as follows:

$$b_j^i = b_{j+1}^{i-1} + b_0^{i-1} f_{j+1} + d_{i-1} c_j \tag{12}$$

with $b_j^0 = b_m^{i-1} = d_{(m-1)/2} = 0$ for $0 \le j \le m-1$.

In order to combine $A^{(m+1)/2}$ and $B^{(m+1)/2}$ to produce the desired result, P, a number of m two-input XOR gates should indeed be employed.

Developing the Dependency Graph

The two recursive expressions (10) and (12) define the iterative part of the Montgomery multiplication algorithm. As can be seen, the two equations' computation structures are identical and separate, but they differ in their coordinate directions. Figures 1 and 2 display the derived dependency graphs (DGs) for field size $m = 5$. The DGs are displayed in a two-dimensional integer domain $\mathbb{D}$ with indices i and j. The iterative Formulas (10) and (12) are computed by the $m \times (m+1)/2$ nodes in the DGs.

All inputs start out in the following directions: The left-to-right direction is used to enter the input signals d_{m-i} and d_i, $1 \le i \le (m+1)/2$. The top of the DGs serves as the entry point for the input signals c_{m-1-j} and c_j. Additionally, from the top of the DGs, f_{m-1-j} and f_{j+1} are entered sequentially. The red slanted lines that are depicted at the right and left corners of the input nodes, respectively, are used to insert input signals with initial parameters of a_{m-2-j}^0 and b_{j+1}^0, which are both equal to 0. The intermediate partial products of the A and B coefficients are calculated in each node and sent to the nodes of the subsequent row. The sum of the the last coefficients of $A^{(m+1)/2}$ and $B^{(m+1)/2}$ yields the final outcome, P. XOR gates with two inputs are used to implement the summation, as will be demonstrated in the next section.

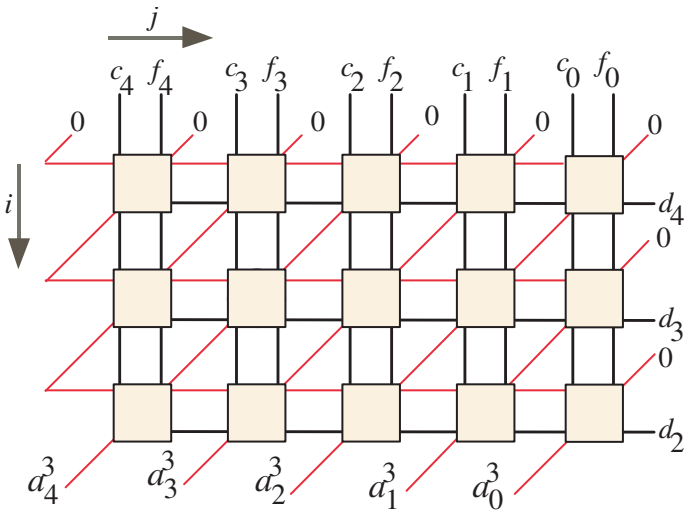

Figure 1. DG of iterative Equation (10) for $m = 5$.

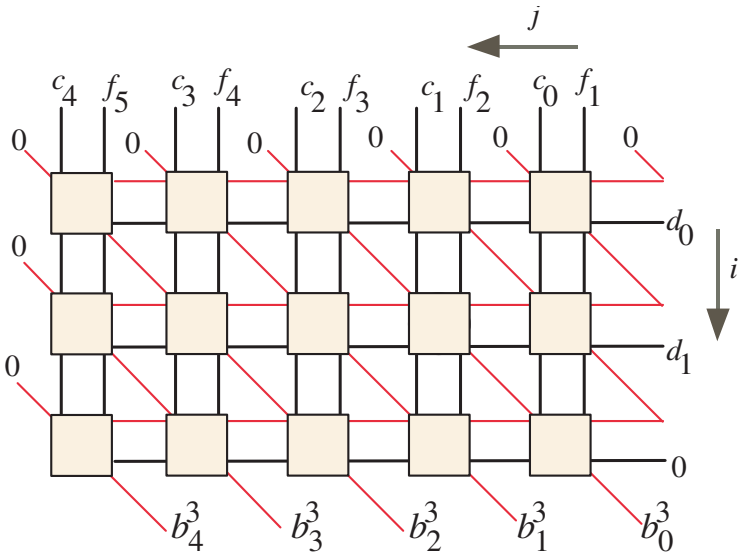

Figure 2. DG of iterative Equation (12) for $m = 5$.

3. Construction of the Montgomery Word-Serial Multiplier Processor

We will convert the recursive formulas, Equations (10) and (12), to processor arrays using a formally defined procedure previously described in [20,30–36].

3.1. Scheduling Functions

Assume that the bus-size of the processor arrays we are working with is k bits. By inspecting the DG displayed in Figure 1, we can develop the non-linear scheduling function shown below, Equation (13). Each node or point $\mathbf{P}$ in this figure can be given a suitable execution time value with the aid of this function.

$$\Psi_A(\mathbf{P}) \quad = \quad (i-1)\left\lceil \frac{m}{k} \right\rceil + \left\lfloor \frac{m-1+k-j}{k} \right\rfloor \tag{13}$$

The node $\mathbf{P}(i,j)$ in the DG can be given a scheduling time value by the function $\Psi_A(\mathbf{P})$.

For the assumed field size $m = 5$ and word size $k = 3$, the scheduling time index values after applying the non-linear scheduling function of Equation (13) are displayed in Figure 3. As we notice from this figure, each group of k nodes can be assigned the same scheduling time to be executed in parallel. Additionally, we can notice that an extra column with zero inputs is added to the left side of the DG. Adding this column is attributed to the number of columns not being a multiple of k. For the general case, we should add $\nu = k\lceil \frac{m}{k} \rceil - m$ extra columns with zero inputs at the left side when the number of columns is not a multiple of k. Furthermore, after $\left(\frac{m+1}{2}\right)\left\lceil \frac{m+1}{k} \right\rceil$ time steps, the output A will be reachable at the output bus.

To assign the same index time values to each node of the DG of Figure 2, we can use the following non-linear scheduling function.

$$\Psi_B(\mathbf{P}) \quad = \quad (i-1)\left\lceil \frac{m}{k} \right\rceil + \left\lfloor \frac{j+k}{k} \right\rfloor \tag{14}$$

The following application of this scheduling function to the DG of Figure 2, the scheduling time index values for the assumed field size $m = 5$ and word-size $k = 3$ are shown in Figure 4. As the case of the DG in Figure 3, all k nodes are assigned the same scheduling times to be executed simultaneously. Additionally, one extra column with zero inputs is added to the left side of the DG as the number of columns is not a multiple of k. For the general case, we should incorporate at the left side of the DG extra $\nu = k\lceil \frac{m}{k} \rceil - m$ when

the number of columns is not an integer multiple of k. Additionally, the output B will be reachable after $\left(\frac{m+1}{2}\right)\left\lceil\frac{m+1}{k}\right\rceil$ time steps.

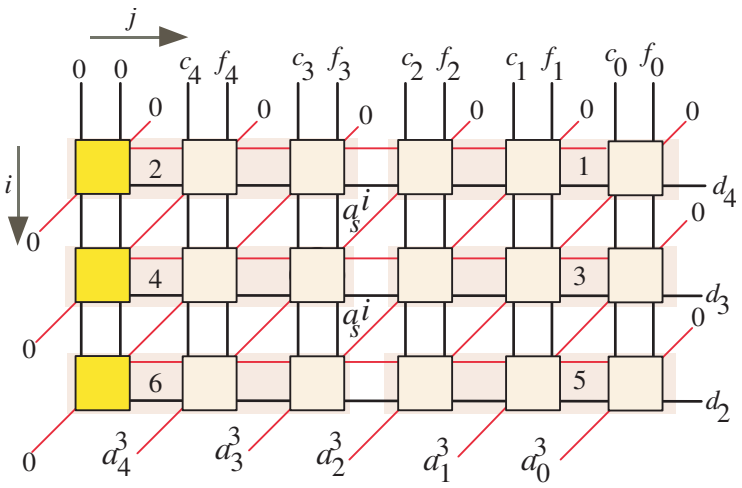

Figure 3. DG scheduling of iterative Equation (10) for $m = 5$ and $k = 3$.

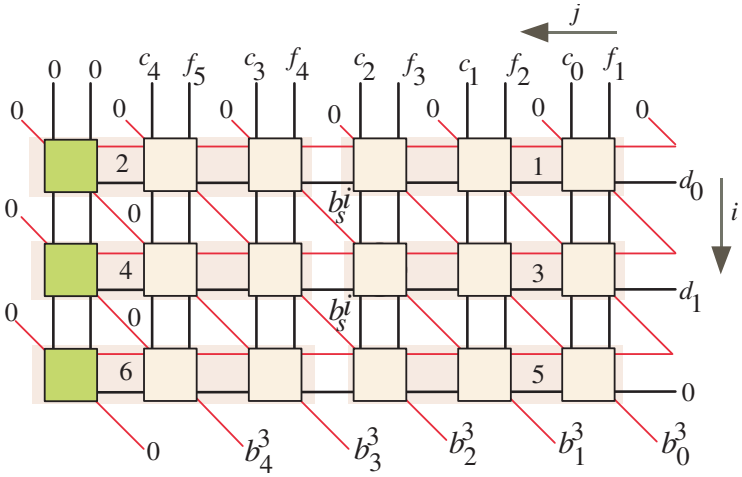

Figure 4. DG scheduling of iterative Equation (12) for $m = 5$ and $k = 3$.

The system developer's ability to control the volume of work of the whole processor array constitutes one of the most crucial aspects of the adapted scheduling functions. Based on the non-linear scheduling Formulas (13) and (14), there is only one group of k bits engaged at any particular time. As a result, the volume of work on the PE corresponds to the course load on the whole processor array.

3.2. Projection Function

The projection function approach described in [30] is used to map multiple nodes in the DGs displayed in Figures 3 and 4 to a specific node. The reason for this is that the selected batches of nodes in Figures 3 and 4 only perform an operation once. In order to utilize the processing components, we should map multiple nodes into a single PE. For the DG of Figure 3, we suggest the below non-linear projection function to map a DG node $\mathbf{\Sigma}(i,j)$ to a new node $\overline{\mathbf{\Sigma}}(z,w)$.

$$\overline{\mathbf{\Sigma}}(z,w) = \mathbf{\Sigma}_s\,\mathbf{\Sigma}(i,j) \quad (15)$$

$$z = i \quad (16)$$

$$w = m-1-j \bmod k \quad (17)$$

$$\mathbf{\Sigma}_s = [\ 1 \quad .\bmod k\ ] \quad (18)$$

where "·" represents a placeholder for the argument as discussed in [30].

Additionally, we advise using the non-linear projection function below to project a node $\mathbf{\Sigma}(i,j)$ to a new node $\overline{\mathbf{\Sigma}}(z,w)$ in the case of DG of Figure 4.

$$\overline{\mathbf{\Sigma}}(z,w) = \mathbf{\Sigma}_s \, \mathbf{\Sigma}(i,j) \tag{19}$$

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

$$w = j \bmod k \tag{21}$$

$$\mathbf{\Sigma}_s = [\ 1 \quad . \bmod k\ ] \tag{22}$$

Referring to Figures 3 and 4, the assumed projection functions are applied to produce the word-serial processor structure, which is displayed in Figure 5. It includes two parallel-running k bits processor arrays. The coefficients of polynomial A are calculated by the processor array to the right. The left processor array also calculates the polynomial B coefficients. Using k two-input XOR gates, the output coefficients of A and B are added to produce the result P. The processor architecture also includes three input registers, with word sizes of k bits, for inputs C and F. Additionally, it has one output register, P, with a word size of k bits. The input bits d_{m-1} and d_{i-1} are serialized to each processor array block using the flip-flops to the right of each processor array block. It is not necessary to add input registers to either A or B because their preliminary input values are both set to zero. We will discuss how to set them to zero values in more detail later.

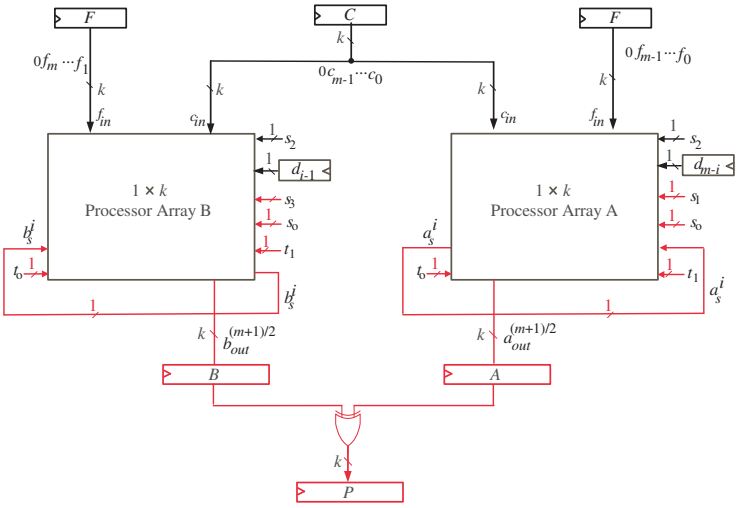

Figure 5. Word-serial accelerator structure.

Figure 6 illustrates the components of processor array block A for the generic scenario of k-bits. In the processor array, there are four PE types that are marginally distinct. The PEs' design characteristics are shown in Figures 7–10. The regular PEs are positioned in the middle of the processor array and contain fewer logic elements as depicted in Figure 8. Each PE has three shift registers: SR-C, SR-F, and SR-A. The number of flip-flops that make up the shift registers SR-C and SR-F are given by $h = \lceil m/k \rceil$. Using these registers, the input signals f_{in} and c_{in} are fed back to the PE after being postponed by h time steps. In processing elements PE_0 and PE_j, shift register SR-A contains h flip-flops. However, it has $h+1$ flip-flops in PE_k. As a result of the different times at which signals a_{m-1}^{i-1} and a_{int}^{i} are fed to the final processing element PE_k, SR-A is separated in processing element PE_{k-1} as indicated in Figure 9. After $h-1$ time steps, the final signal a_{m-1}^{i-1} should be accessible at the input of PE_k, and a_{int}^{i} should be accessible at PE_k after h time steps. In order to further delay a_{int}^{i} by one more time step, a flip-flop represented by a red box is added after the SR-A block. In order to determine which of the two signals should be passed to PE_k at the appropriate time, the control signal t_1 will control tri-state buffers T_2 and T_3. As we can see in Figure 10, the signal a_{m-1}^{i-1} is produced from PE_{k-2} and sent to all PEs through PE_{k-1}.

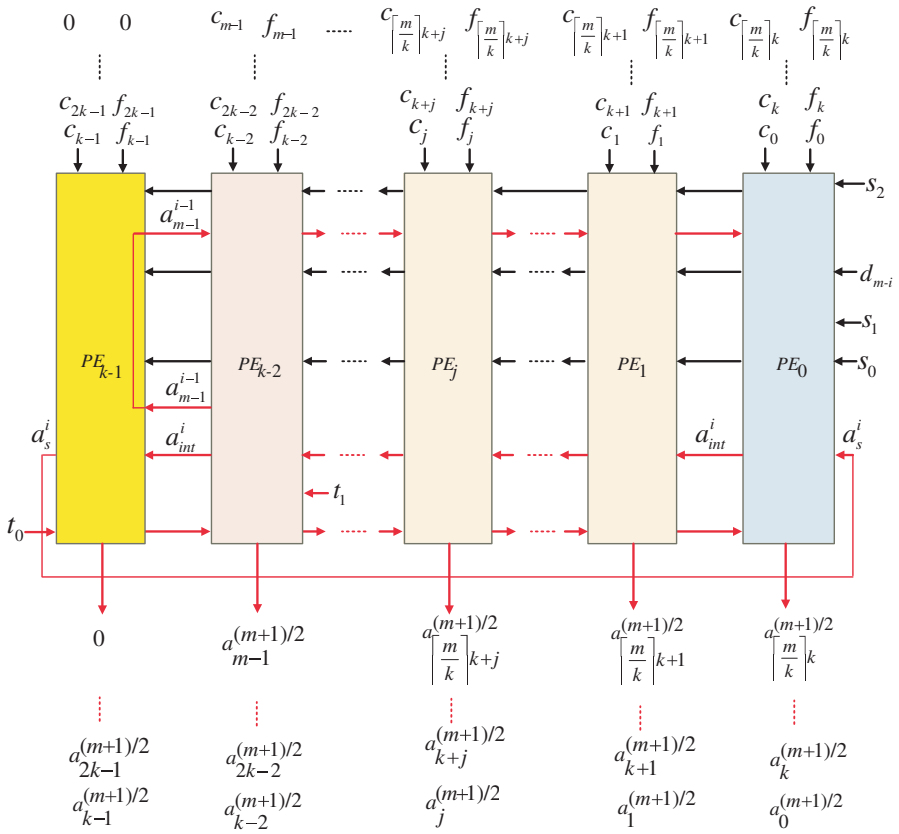

Figure 6. The details of processor array block A.

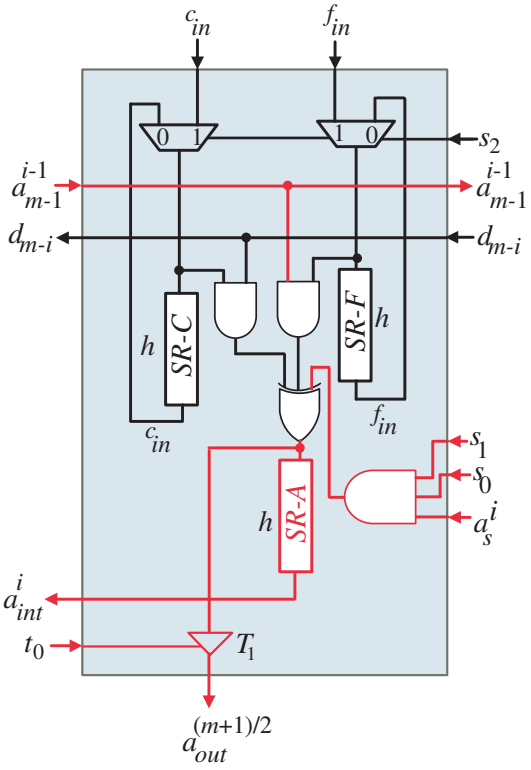

Figure 7. First PE details of processor array block A.

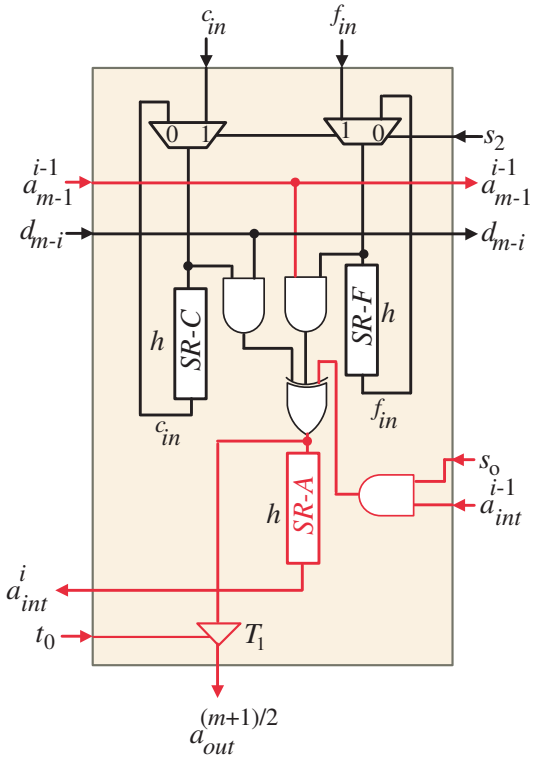

Figure 8. Intermediate PE details processor array block A.

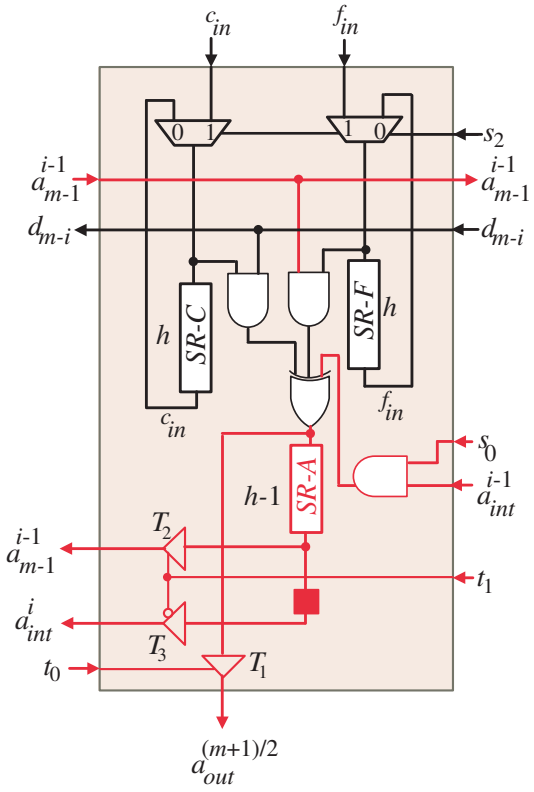

Figure 9. Before Last PE details of processor array block A.

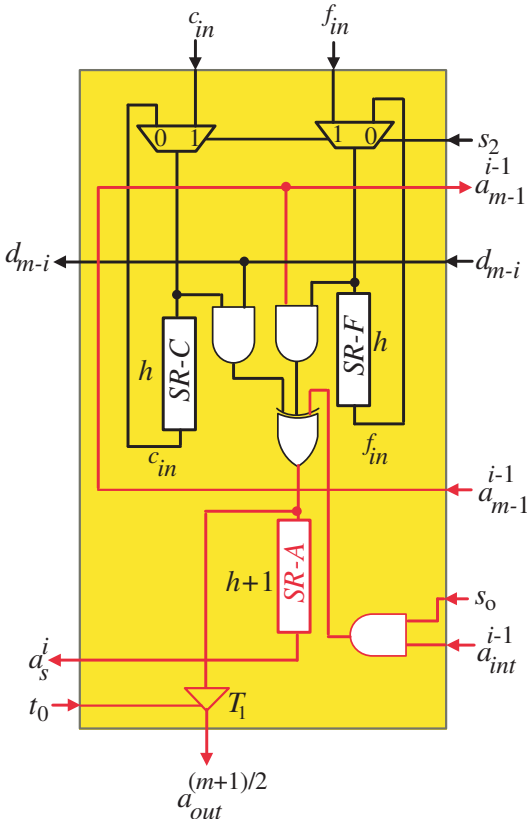

Figure 10. Last PE details of processor array block A.

Tri-state buffers T_1, in all PEs, are controlled by control signal t_0 to pass the final product values of A, $a_{out}^{(m+1)/2}$, to the output bus at the proper time. Control signal s_2 is used to select between the initial input values of f_{in} and c_{in} and their delayed values obtained from SR-F and SR-C shift registers. Control signal s_1 is connected to PE_0 to select between passing a_s^i signal or forcing a zero value at the output of the three-input AND gate, shown in Figure 7. The forced zero values represent the zero values of variable A shown at the right edge of the DG depicted in Figure 1.

The initial zero values of variable A will be forced by control signal s_0 at the first and second time steps. As shown in all PEs, s_0 is connected to one input of the two-input AND gate. When s_0 is deactivated, $s_0 = 0$, it will force zero value at the output of the AND gate, which represents the initial value of A variable. When s_0 is activated, $s_0 = 1$, the AND gate will pass the resulted intermediate value of A, a_{int}^{i-1}.

Figure 11 presents the components of the processor array block B for the generic scenario k-bits. It has three PEs that are marginally distinct. The PEs design characteristics are displayed in Figures 12–14. The regular PEs are positioned in the middle of the processor array and contain fewer logic elements as depicted in Figure 13. Each PE has three shift registers: SR-C, SR-F, and SR-B. The number of flip-flops that make up the shift registers SR-C and SR-F are given by $h = \lceil \frac{m}{k} \rceil$. Using these registers, the input signals f_{in} and c_{in} are fed back to the PE after being postponed by h time steps. In all PEs except PE_0, shift register SR-B contains h flip-flops and PE_0 contains only $h - 1$ flip-flops. In PE_0, one more flip-flop is added after the SR-B block, as shown in Figure 12, to delay b_0^{i-1} signal by one more time step. In order to ensure passing b_0^{i-1} at the appropriate time, control signal t_1 is also employed to control the Tri-state buffer T_4. As can be seen in Figure 12, the signal b_0^{i-1} is created from $PE0$ and transferred to all PEs.

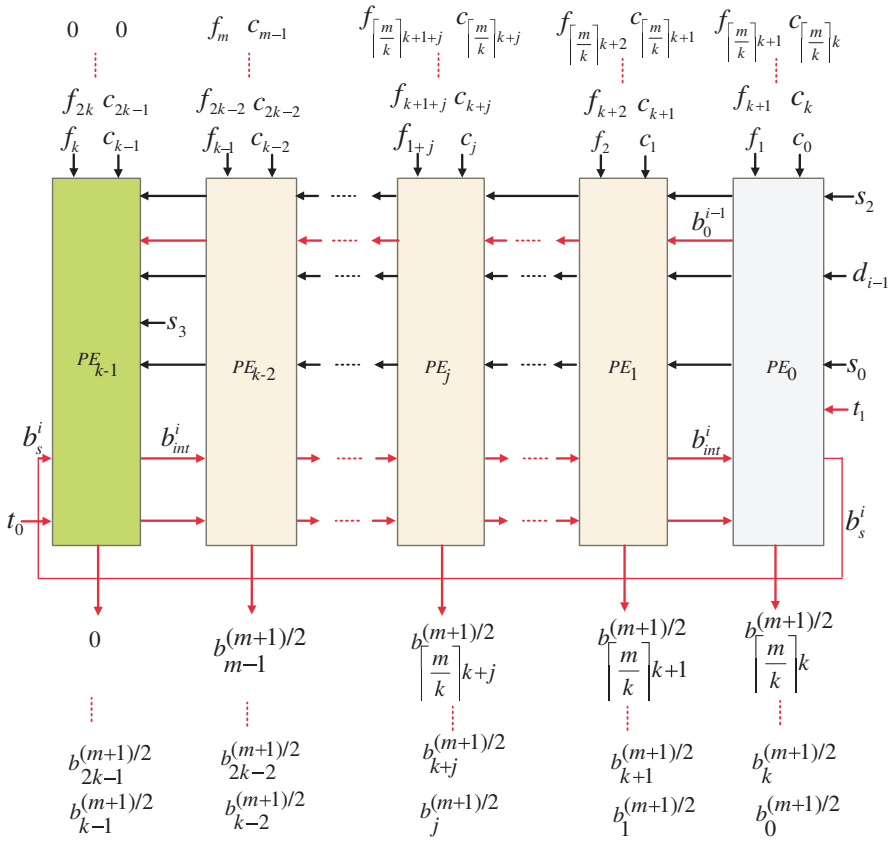

Figure 11. The details of processor array block B.

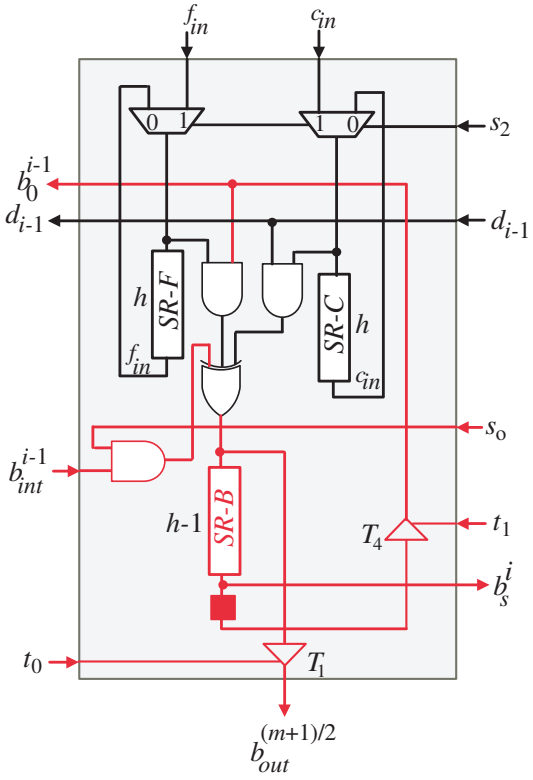

Figure 12. First PE details of processor array block B.

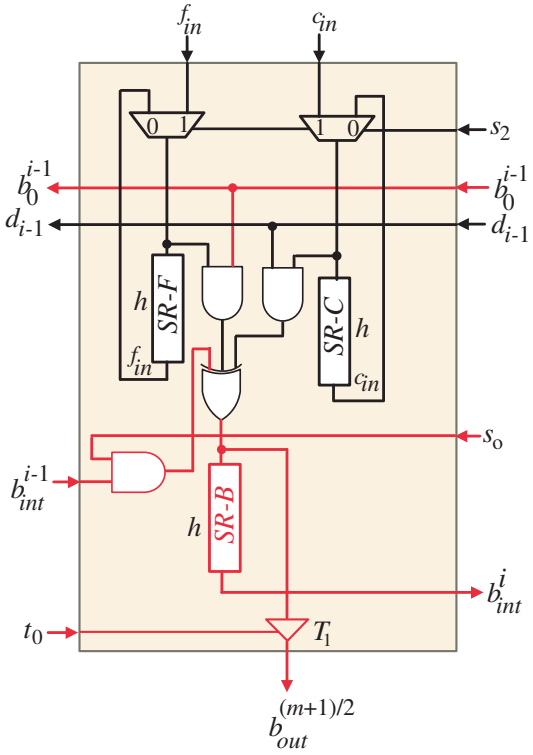

Figure 13. Intermediate PE details of processor array block B.

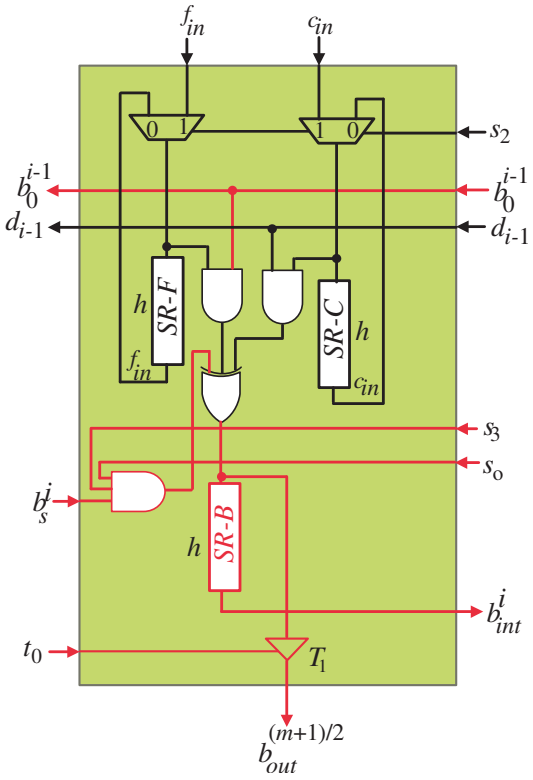

Figure 14. Last PE details of processor array block B.

Control signal t_0 directs tri-state buffers T_1 in all PEs to pass the final product values of B, $b_{out}^{(m+1)/2}$, to the output bus at the appropriate time. To choose between the initial input values of f_{in} and c_{in} and their delayed values obtained from SR-F and SR-C shift registers, control signal s_2 is used. In order to choose between forcing a zero value at the output of the three-input AND agate, shown in Figure 14, or passing the b_s^i signal, control signal s_3 is

connected to PE_k. The forced zero values correspond to the zero values of the variable B that are displayed at the left edge of the DG in Figure 2.

Control signal s_0 will be utilized at the first and second time steps to force variable B's initial zero values. As we can observe in each PE, the input of the two-input AND gate is linked to s_0. It will force a value of zero, which corresponds to the initial value of the B variable, at the output of the AND gate when s_0 is deactivated, $s_0 = 0$. The AND gate will transfer the intermediate value of B, b_{int}^{i-1}, when s_0 is activated, $s_0 = 1$.

The operation characteristics of the investigated multiplier can be described as follows for general field size m and word size k:

1. Control signal s_2 should be placed to one ($s_2 = 1$) during the initial $\lceil m/k \rceil$ clock cycles to enable MUXes to transmit the initial words of variables C and F to all PEs of the processor array blocks. In order to achieve the initial zero values of both variables A and B, control signal s_0 should also be set to 0 ($s_0 = 0$) within the same clock cycles;
2. The control signal s_2 deactivates ($s_2 = 0$) to re-feed the input values of C and F to all PEs of the processor array blocks during the remaining clock cycles. In order to pass the intermediate values of the variables A, a_{int}^{i-1}, and B, b_{int}^{i-1}, control signal s_0 should also be activated ($s_0 = 1$);
3. During the clock cycles $T = (i-1)\lceil \frac{m}{k} \rceil$+1, $1 \leq i \leq (m+1)/2$, input bits d_{m-i} and d_{i-1} are circulated to the PEs of the processor array blocks of A and B, respectively;
4. To force the three-input AND gate, depicted in Figure 7, to generate zero values in the processor array block A, control signal s_1 deactivates ($s_1 = 0$) at clock cycles $T = (i-1)\lceil \frac{m}{k} \rceil + 1$, $2 \leq i \leq (m+1)/2$. These values, as we previously stated, stand for the zero values of variable A that are displayed at the right edge of the DG displayed in Figure 1. In order to send the a_s^i signal through the three-input AND gate, control signal s_1 activates, $s_1 = 1$, at the remaining clock cycles;
5. The control signal s_3 deactivates ($s_3 = 0$) at clock cycles $T = i\lceil \frac{m}{k} \rceil$, $2 \leq i \leq (m+1)/2$, causing the three-input AND gate, depicted in Figure 14, to produce zero values in processor array block B. As stated previously, these values represent the zero values of variable B that are depicted at the left edge of the DG in Figure 2. Control signal s_3 activates, $s_3 = 1$, to transmit the b_s^i signal through the three-input AND gate at the remaining clock cycles;
6. Control signal t_1, shown in Figures 9 and 12, activates ($t_1 = 1$) in both processor array blocks to enable the Tri-State buffers T_2 and T_4, respectively, to pass the bits of a_{m-1}^{i-1} and b_0^{i-1}, $1 \leq i \leq (m+1)/2$, to all PEs. During the remaining clock cycles, the control signal t_1 deactivates ($t_1 = 0$), allowing the Tri-Sate buffer T_2, shown in Figure 9, to pass the intermediate bits of a_{int}^i to the next PE;
7. The output variables A and B come from the processor array blocks A and B, respectively, in a word serial format during the clock cycles $T \geq \frac{m-1}{2}\lceil \frac{m}{k} \rceil + 1$. The corresponding bits of output variables A and B are added together using the k two-input XOR gates, as shown in Figure 5, to produce the result P in a word- serial manner after being postponed by one further clock cycle.

4. Complexities Analysis

This section compares the suggested word-serial Montgomery multiplier to current effective word-serial multipliers proposed in [23,37–39] by describing their area, delay, and consumed energy complexities. The suggested Montgomery multiplier's area and delay complexities are compared to those of the prior, effective word-serial multipliers in Table 1. Logic gates and other logic elements of the multiplier processor are counted to determine the area complexity. The total number of clock cycles needed to generate the multiplier result is known as the multiplier's latency (L). The total of all gate delays in the longest path of the logic circuit represents the critical path delay (CPD) of the multiplier architecture. CPD and latency are used to quantify the total delay complexity of the multiplier. The symbols φ_T, φ_A, φ_X, and φ_{MUX}, respectively, stand for the delays of the Tri-sate buffer, two-input AND, two-input XOR, and 2-to-1 MUX.

The following is a description of the additional mathematical notation in Table 1:

1. $D_1 = 7m + m(\lceil \log m \rceil) + k + 3$;
2. $D_2 = 2k^2 + 2k(\lceil m/k \rceil) + 4k + 1$;
3. $D_3 = 2k^2 + 3k(\lceil m/k \rceil) + 2k$;
4. $L_1 = k + \lceil m/k \rceil^2 + \lceil m/k \rceil$;
5. $\sigma_1 = \varphi_A + (\lceil \log_2 k \rceil + 1)\varphi_X$;
6. $\sigma_2 = \varphi_A + 2\varphi_X$;
7. $\sigma_3 = \varphi_A + \varphi_X$;
8. $\sigma_4 = \varphi_T + \varphi_A + 1.5\varphi_X + \varphi_{MUX}$.

Table 1. Estimated space and time for the selected word-serial multipliers.

Multiplier	Tri-State	AND	XOR	MUXes	Flip-Flops	Latency	CPD
Xie [37]	0	$2mk$	$2mk + 6m - 6\frac{m}{k} + 6$	0	$4mk + 4m + 2k$	$2\lceil \frac{m}{k} \rceil + 2\lceil \log_2 k \rceil$	$2\varphi_X$
Pan [23]	0	$m\sqrt{m}$	$\sqrt{mk}(2 + k) + k$	0	D_1	$2\lceil \sqrt{\frac{m}{k}} \rceil$	σ_1
Hua [38]	0	k^2	$k^2 + 4 - 5k + 1$ (2)	0	D_2	$6k\lceil \frac{m}{k} \rceil^2$	σ_2
Chen [39]	0	$k^2 + k$	$k^2 + 2k$	$2k$ (3)	D_3	L_1	σ_3
Proposed	$2k + 3$	$6k + 2$	$4k$	$4k$	$6k(h$ (1) $+ 1) + 1$	$(\frac{m+1}{2})\lceil \frac{m+1}{k} \rceil + 1$	σ_4

(1) $h = \lceil m/k \rceil$. (2) The average area of the three-input logic XOR is 1.5× that of the two-input logic XOR. (3) The switches in multiplier of Chen [39], occupy the same space as the 2-to-1 MUX because they share the same number of transistors.

For an accurate comparison, the overall number of approximated flop-flops is appended to the flip-flops of the multiplier processor's input and output registers. Table 1 shows that the recommended multiplier layout has a much smaller footprint than the prior multiplier constructions due to its space complexity of order $\mathcal{O}(k)$.

For the purpose of validating the qualitative conclusions in Table 1, we modeled the existing and recommended multiplier constructions using the VHDL hardware description language. For the embedded word sizes of $k = 8$, $k = 16$, $k = 32$, as well as the field size $m = 409$, we synthesized the compared multiplier constructions. Version 2005.09-SP2 of Synopsys tools and the NanGate Open Cell Library (15 nm, 0.8 V) were used for the synthesis.

According to Table 2, the outlined synthesis design metrics are as follows:

- The outlined area (A) results are computed using a two-input NAND gate and are given in kilo-gates (*kgates*);
- The outlined total computation time (T) values are expressed in nanoseconds (*ns*);
- At a frequency of 1 KHz, the obtained values of consumed power (P) are expressed in milliwatt (*mW*);
- P and T are multiplied to determine the consumed energy (E), and the results are expressed in femtojoule (*fJ*);
- By multiplying A and T, the Area-Time (AT) design metric is computed, and the outcomes are expressed in (*kgates.ns*).

The last columns of Table 2 list cost savings of the recommended multiplier architecture in terms of area (%A), area-time (%AT), power (%P), and energy (%E) compared to the existing multiplier constructions. The charts in Figures 15–18 compare the actual results obtained for A, T, P, and E of the proposed multiplier construction with those of the selected multiplier constructions.

The proposed and compared word-serial multipliers are displayed in Figure 15, which presents the area results for the various embedded word sizes on a line chart. The dark blue line serves as a visual representation of the suggested layout. The area is represented on a logarithmic scale by the vertical axis, and the embedded word sizes are shown on the horizontal axis. The average area savings for the employed embedded word sizes of the

offered multiplier over the compared ones are represented by the green line. By looking at the chart, we can deduce that the recommended multiplier architecture saves a significant amount of space when compared to the existing word-serial multiplier structures by an average percentage of 86.3%. As previously mentioned, the offered multiplier saves space because it has a lower area complexity of $\mathcal{O}(k)$ than the other multiplier designs.

Table 2. Performance measures for the chosen word-serial multipliers at $m = 409$ and various k values.

Multiplier	k	A [*Kgates*]	T [*ns*]	P [*mW*]	E [*fJ*]	AT	%A	%AT	%P	%E
Xie [37]	8	92.9	18.3	225.6	4.1	1698.7	97.3	24.5	99.4	69.4
	16	147	9.7	375.5	3.6	1425.9	97.5	33.7	99.3	67.5
	32	195.1	5.5	477.4	2.6	1078.9	97.7	46.8	99.2	69.0
Pan [23]	8	130.5	9.9	252.9	2.5	1291.6	98.1	0.7	99.4	49.6
	16	153.9	8.8	320.1	2.8	1354.6	97.6	30.2	99.1	58.0
	32	194.3	6.8	425.1	2.9	1317.6	97.7	56.4	99.1	71.6
Hua [38]	8	7.9	19,053.5	4.4	82.9	152,237.2	69.1	99.2	66.5	98.5
	16	10.4	9526.7	5.9	55.7	99,077.9	65.0	99.0	53.3	97.9
	32	19.9	4763.3	11.2	53.1	94,838.7	77.8	99.4	66.2	98.5
Chen [39]	8	10.2	659.4	5.1	3.4	6699.7	75.7	80.8	71.5	62.6
	16	13.5	203.0	8.4	1.7	2742.9	73.1	65.5	67.4	30.4
	32	26.6	86.8	15.9	1.4	2306.3	83.4	75.1	76.4	40.7
Proposed	8	2.5	519.4	1.5	1.3	1282.8	-	-	-	-
	16	3.6	259.7	2.7	1.2	945.2	-	-	-	-
	32	4.4	129.8	3.8	0.8	573.9	-	-	-	-

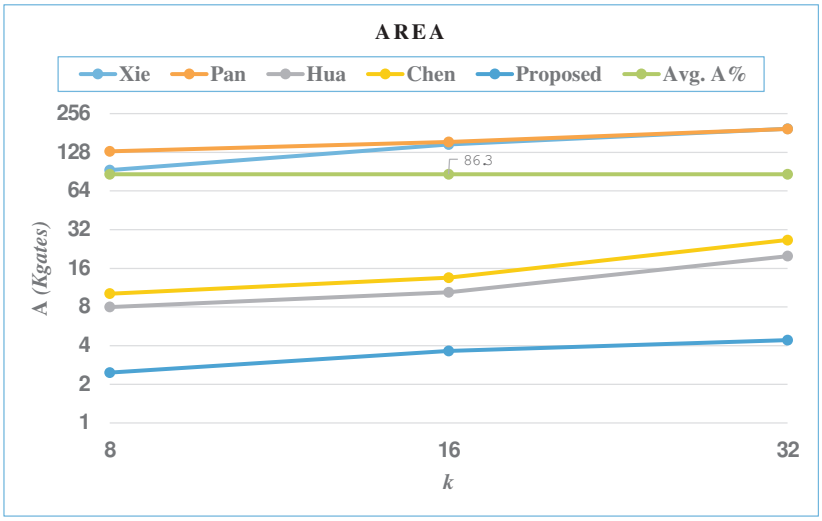

Figure 15. Area results.

Figure 16 illustrates the offered multiplier architecture's Area-Time (AT) complexity and the AT complexity of the compared word-serial ones for the recommended embedded word sizes. It shows a graph with two coordinates. The vertical coordinate denotes the achieved AT values on a log scale, whereas the horizontal coordinate indicates the recommended embedded word sizes. AT average savings of the suggested multiplier structure over the existing ones is demonstrated by the green horizontal line. The graph

demonstrates that the suggested multiplier is the one that has the lowest AT at the different word sizes. As illustrated in Figure 16, the recommended multiplier layout provides an average savings in AT of 53.7% over the existing ones.

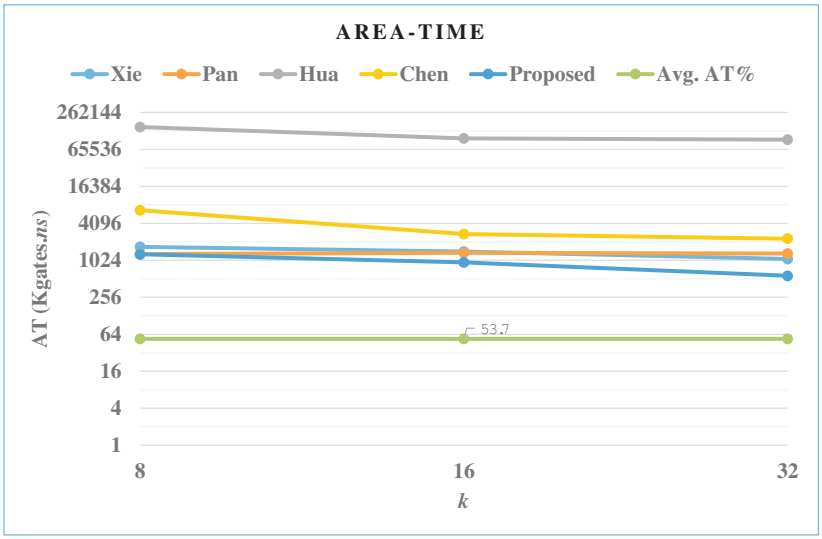

Figure 16. Area-Time results.

According to Figure 17, the recommended multiplier architecture accomplishes average power reductions of 83.2%. The horizontal green line serves as a signal for this. The embedded word sizes are represented on the horizontal line of the graph, and the power outcomes are displayed on the vertical line, which is scaled logarithmically. The reduction in power is attributed to the proposed multiplier design having less space complexity than the other designs. Minimizing the layout area lowers parasitic capacitance, which in turn lowers overall switching activity, being one of the main sources of power consumption. Additionally, the systolic characteristic of the suggested design eliminates glitches that significantly affect the amount of power consumed.

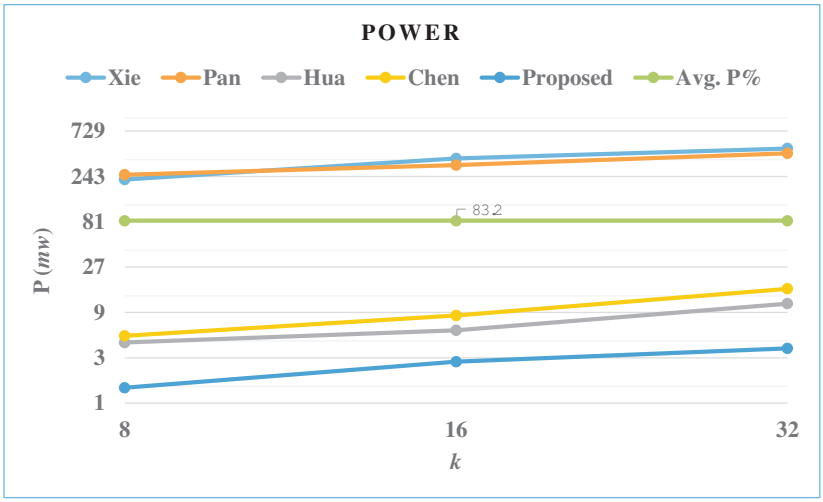

Figure 17. Results of the consumed power.

The energy outcomes for the various embedded word sizes are plotted on a logarithmic scale chart as displayed in Figure 18. The described multiplier layout conserves energy on average by 72.6% compared to the state-of-the-art word-serial multipliers, as indicated by the horizontal green line in Figure 18. The reductions in energy are mainly attributed to the offered multiplier layout consuming significantly less power than the compared ones.

According to the previous study, the recommended word-serial multiplier layout performs better in terms of space and consumed energy compared to the other competing multiplier designs. Consequently, the recommended multiplier layout is suitable for IoT edge devices used in IoT applications with limited resource availability.

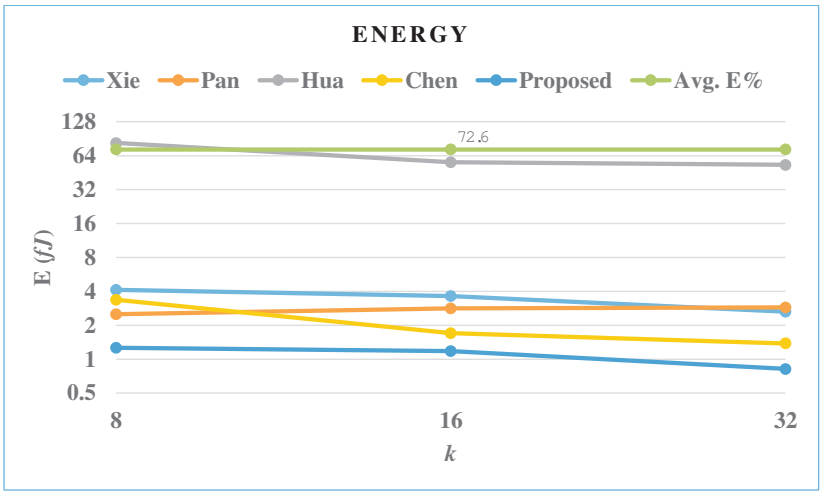

Figure 18. Results of the consumed energy.

5. Summary and Conclusions

In this article, we presented a word-serial processor architecture for Montgomery modular multiplier over GF(2^m). The primary advantage of the offered multiplier is its capacity to control both the workflow on the processor and the total number of computation steps needed to generate the required output. The test results reveal that for different embedded word sizes, the suggested multiplier processor performs better in terms of area and consumed energy than the effective word-serial multipliers historically reported in the literature, making it much more appropriate for use in IoT edge devices used within resource-constrained IoT applications. We intend to implement the whole ECC cryptographic processor based on the suggested multiplier processor structure in subsequent work. This will make it easier to calculate the system's overall energy and space savings.

Author Contributions: Conceptualization, A.I.; methodology, A.I. and F.G.; software, A.I.; validation, A.I.; formal analysis, A.I.; investigation, A.I.; resources, A.I.; data curation, A.I.; writing—original draft preparation, A.I.; writing—review and editing, A.I. and F.G.; visualization, A.I.; supervision, A.I.; project administration, A.I.; funding acquisition, A.I. and F.G. All authors have read and agreed to the published version of the manuscript.

Funding: Deputyship for Research & Innovation, Ministry of Education in Saudi Arabia, project number (IF2-PSAU-2022/01/21637).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Acknowledgments: The authors extend their appreciation to the Deputyship for Research & Innovation, Ministry of Education in Saudi Arabia for funding this research work through the project number (IF2-PSAU-2022/01/21637).

Conflicts of Interest: The authors declare no conflict of interest.

Sample Availability: Not applicable.

Abbreviations

The following abbreviations are used in this manuscript:

IoT	Internet of Things
ASIC	Application Specific Integrated Circuit
ECC	Elliptic Curve Cryptography
DG	Dependency Graph
AT	Area-Time
GF	Galois Field
RSA	Rivest, Shamir, and Adleman
CPD	Critical Path Delay

References

1. Mittal, M.; Vijayal, S. Detection of attacks in iot based on ontology using sparql. In Proceedings of the 2017 7th International Conference on Communication Systems and Network Technologies (CSNT), Nagpur, India, 11–13 November 2017 ; pp. 206–211.
2. Pourghebleh, B.; Hayyolalam, V.; Anvigh, A.A. Service discovery in the internet of things: review of current trends and research challenges. *Wirel. Netw.* **2020**, *26*, 5371–5391. [CrossRef]
3. Anajemba, J.H.; Iwendi, C.; Mittal, M.; Yue, T. Improved advance encryption standard with a privacy database structure for IoT nodes. In Proceedings of the 2020 IEEE 9th International Conference on Communication Systems and Network Technologies (CSNT), Gwalior, India, 10–12 April 2020; pp. 201–206.
4. Qiu, J.; Tian, Z.; Du, C.; Zuo, Q.; Su, S.; Fang, B. A survey on access control in the age of internet of things. *IEEE Internet Things J.* **2020**, *7*, 4682–4696. [CrossRef]
5. Shafiq, M.; Tian, Z.; Bashir, A.K.; Du, X.; Guizani, M. IoT malicious traffic identification using wrapper-based feature selection mechanisms. *Comput. Secur.* **2020**, *94*, 101863. [CrossRef]
6. Su, S.; Tian, Z.; Li, S.; Deng, J.; Yin, L.; Du, X.; Guizani, M. IoT root union: a decentralized name resolving system for IoT based on blockchain. *Inf. Process. Manag.* **2021**, *58*, 102553. [CrossRef]
7. Gu, Z.; Li, H.; Khan, S.; Deng, L.; Du, X.; Guizani, M.; Tian, Z. Iepsbp: A cost-efficient image encryption algorithm based on parallel chaotic system for green IoT. *IEEE Trans. Green Commun. Netw.* **2022**, *6*, 89–106. [CrossRef]
8. Wang, H.; Zhang, W.; He, H.; Liu, P.; Luo, D.X.; Liu, Y.; Jiang, J.; Li, Y.; Zhang, X.; Liu, W.; et al. An evolutionary study of IoT malware. *IEEE Internet Things J.* **2021**, *8*, 15422–15440. [CrossRef]
9. Anajemba, J.H.; Yue, T.; Iwendi, C.; Alenezi, M.; Mittal, M. Optimal cooperative offloading scheme for energy efficient multi-access edge computation. *IEEE Access* **2020**, *8*, 53931–53941. [CrossRef]
10. Majumder, S.; Ray, S.; Sadhukhan, D.; Khan, M.K.; Dasgupta, M. Ecc-coap: Elliptic curve cryptography based constraint application protocol for internet of things. *Wirel. Pers. Commun.* **2021**, *116*, 1867–1896. [CrossRef]
11. Ali, U.; Idris, M.Y.I.; Frnda, J.; Ayub, M.N.B.; Alroobaea, R.; Almansour, F.; Shagari, N.M.; Ullah, I.; Ali, I. Hyper elliptic curve based certificateless signcryption scheme for secure IIoT communications. *CMC-Comput. Mater. Contin.* **2022**, *71*, 2515–2532. [CrossRef]
12. Dong, J.; Zheng, F.; Lin, J.; Liu, Z.; Xiao, F.; Fan, G. Ec-ecc: Accelerating elliptic curve cryptography for edge computing on embedded gpu tx2. *ACM Trans. Embed. Comput. Syst.* **2022**, *21*, 1–25. [CrossRef]
13. Kim, K.W.; Jeon, J.C. Polynomial basis multiplier using cellular systolic architecture. *Iete J. Res.* **2014**, *60*, 194–199. [CrossRef]
14. Choi, S.; Lee, K. Efficient systolic modular multiplier/squarer for fast exponentiation over GF(2^m). *IEICE Electron. Express* **2015**, *12*, 20150222. [CrossRef]
15. Kim, K.W.; Kim, S.H. Efficient bit-parallel systolic architecture for multiplication and squaring over GF(2^m). *IEICE Electron. Express* **2018**, *15*, 20171195. [CrossRef]
16. Matteo, S.D.; Baldanzi, L.; Crocetti, L.; Nannipieri, P.; Fanucci, L.; Saponara, S. Secure elliptic curve crypto-processor for real-time IoT applications. *Energies* **2021**, *14*, 4676. [CrossRef]
17. Chen, L.H.; Chang, P.L.; Lee, C.-Y.; Yang, Y.K. Scalable and systolic dual basis multiplier over GF(2^m). *Int. J. Innov. Inf. Control.* **2011**, *7*, 1193–1208.
18. Bayat-Sarmadi, S.; Kermani, M.M.; Azarderakhsh, R.; Lee, C.-Y. Dual-basis superserial multipliers for secure applications and lightweight cryptographic architectures. *IEEE Trans. Circ. Sys. II* **2014**, *61*, 125–129. [CrossRef]
19. Gebali, F.; Ibrahim, A. Efficient scalable serial multiplier over GF(2^m) based on trinomial. *IEEE Trans. Very Large Scale Integr. Systems* **2015**, *23*, 2322–2326. [CrossRef]
20. Ibrahim, A.; Gebali, F. Scalable and unified digit-serial processor array architecture for multiplication and inversion over GF(2^m). *IEEE Transactions Circuits Syst. Regul. Pap.* **2017**, *22*, 2894–2906. [CrossRef]
21. Talapatra, S.; Rahaman, H.; Mathew, J. Low complexity digit serial systolic montgomery multipliers for special class of GF(2^m). *IEEE Trans. Very Large Scale Integr. Sys.* **2010**, *18*, 847–852. [CrossRef]
22. Guo, J.H.; Wang, C.L. Hardware-efficient systolic architecture for inversion and division in GF(2^m). *IEE Proc. Comput. Digit.* **1998**, *145*, 272–278. [CrossRef]

23. Pan, J.S.; Lee, C.Y.; Meher, P.K. Low-latency digit-serial and digit-parallel systolic multipliers for large binary extension fields. *IEEE Trans. Circ. Sys. I* **2013**, *60*, 3195–3204. [CrossRef]
24. Lee, C.-Y.; Fan, C.-C.; Yuan, S.-M. New digit-serial three-operand multiplier over binary extension fields for high-performance applications. In Proceedings of the 2017 2nd IEEE International Conference on Computational Intelligence and Applications, Beijing, China, 8–11 September 2017; pp. 498–502.
25. Ramakrishna, P.S.; Lakshmi, B. Low-latency area-efficient systolic bit-parallel GF(2^m) multiplier for a narrow class of trinomials. *Microelectron. J.* **2021**, *117*, 105275.
26. Kim, K.W.; Lee, J.D. Efficient unified semi-systolic arrays for multiplication and squaring over GF(2^m). *IEICE Electron. Express* **2017**, *14*, 20170458. [CrossRef]
27. Meher, P.K.; Lou, X. Low-latency, low-area, and scalable systolic-like modular multipliers for GF(2^m) based on irreducible all-one polynomials. *IEEE Trans. Circuits Syst. Regul. Pap.* **2016**, *64*, 399–408. [CrossRef]
28. Lee, K. Resource and delay efficient polynomial multiplier over finite fields GF(2^m). *J. Korea Soc. Digit. Ind. Inf. Manag.* **2020**, *16*, 1–9.
29. Lee, K. Low complexity systolic montgomery multiplication over finite fields GF(2^m). *J. Korea Soc. Digit. Ind. Inf. Manag.* **2022**, *18*, 1–9.
30. Gebali, F. *Algorithms and Parallel Computers*; John Wiley: New York, NY, USA, 2011.
31. Ibrahim, A.; Alsomani, T.; Gebali, F. New systolic array architecture for finite field inversion. *Can. J. Electr. Comput. Eng.* **2017**, *40*, 23–30. [CrossRef]
32. Ibrahim, A.; Gebali, F.; El-Simary, H.; Nassar, A. High-performance, low-power architecture for scalable radix 2 montgomery modular multiplication algorithm. *Can. J. Electr. Comput. Eng.* **2009**, *34*, 152–157. [CrossRef]
33. Ibrahim, A.; Alsomani, T.; Gebali, F. Unified systolic array architecture for field multiplication and inversion over GF(2^m). *Comput. Electr. J.* **2017**, *61*, 104–115. [CrossRef]
34. Gebali, F.; Ibrahim, A. Low space-complexity and low power semi-systolic multiplier architectures over GF(2^m) based on irreducible trinomial. *Microprocess. Microsyst.* **2016**, *40*, 45–52. [CrossRef]
35. Ibrahim, A.; Elsimary, H.; Gebali, F. New systolic array architecture for finite field division. *IEICE Electron. Express* **2018**, *15*, 20180255. [CrossRef]
36. Ibrahim, A. Efficient parallel and serial systolic structures for multiplication and squaring over GF(2^m). *Can. J. Electr. Comput. Eng.* **2019**, *42*, 114–120. [CrossRef]
37. Xie, J.; Meher, P.K.; Mao, Z. Low-latency high-throughput systolic multipliers over GF(2^m) for nist recommended pentanomials. *IEEE Trans. Circuits Syst. I* **2015**, *62*, 881–890. [CrossRef]
38. Hua, Y.Y.; Lin, J.-M.; Chiou, C.W.; Lee, C.-Y.; Liu, Y.H. Low space-complexity digit-serial dual basis systolic multiplier over GF(2^m) using hankel matrix and karatsuba algorithm. *IET Inf. Secur.* **2013**, *7*, 75–86.
39. Chen, C.-C.; Lee, C.-Y.; Lu, E.-H. Scalable and systolic Montgomery multipliers over GF(2^m). *IEICE Trans. Fundam. Electron. Comput. Sci.* **2008**, *91*, 1763–1771. [CrossRef]