Compact modular multiplier design for strong security capabilities in resource-limited Telehealth IoT devices

Atef Ibrahim, Fayez Gebali

Abstract

Telehealth is an emerging model of delivering quality health to remote communities and stay-at-home users. This is motivated by the rising health care costs and by the benefits of many patients staying at home as opposed to extended hospital stays. Telehealth relies on IoT technology, but IoT devices present the weakest security link to the system. The challenge is to implement strong security capabilities in resource-limited IoT devices. This justifies the use of elliptic curve cryptography (ECC) over the other traditional and resource-consumed approaches such as RSA. Efficient modular multiplication is a basic operation needed for ECC systems. Therefore, the compact and efficient implementation of this operation will significantly affect the performance of ECC in resource-limited applications. This work presents a compact serial-in/serial-out word-based systolic implementation of modular multiplication. The proposed structure is derived using a formal and systematic technique for mapping regular iterative algorithms (RIA) onto processor arrays. The proposed methodology enables control of the processor array workload as well as the workload of each processing element. Controlling the processor word size allows control of system speed, latency, and area. The proposed processor structure saves area and energy consumption by a factor up to 96.3% and 98.5%, respectively.

Keywords: IoT security, Telehealth IoT devices, Cyber physical security, Crypto processors, Modular multipliers, Systolic arrays.

1. Introduction and Related Work

The internet of things (IoT) finds a handy application in providing quality healthcare delivery to remote communities or stay-at-home patients. Moving extended stay patients to their homes proves beneficial to patients, and their families and reduces the escalating costs of healthcare (Belcher et al., 2021; Dykgraaf et al., 2021; Granjal et al., 2015; Atzori et al., 2010). However, IoT devices are considered the weakest link in any system that uses these devices. This is due to the limited computing and energy resources of IoT devices coupled with the fact that most of these devices are heterogeneous and seldom undergo password changes or operating system updates. A practical approach to secure these IoT devices, especially for telehealth, is to use physically unclonable functions (PUFs) to facilitate authentication and secure key establishment/exchange (Fakroon et al., 2021).

Securing IoT operations and communications is provided using elliptic curve cryptography (ECC) in preference to using the more traditional, and very expensive, approaches such as RSA (Rivest et al., 1978; Lidl and Niederreiter, 1994). ECC offers similar security as RSA with shorter key sizes (Di Matteo et al., 2021). An essential step in implementing ECC is providing efficient modular multiplication in the binary extension field GF(2m) since this is the crucial step for field arithmetic, including modular exponentiation, modular squaring, and finding multiplicative inverse (Chiou et al., 2006; Kim and Jeon, 2014; Choi and Lee, 2015; Kim and Kim, 2018).

Modular multipliers can be implemented in serial or parallel based on the intended application. In the case of parallel implementation, the multiplier produces all output bits in a single clock cycle resulting in high throughput at the expense of consuming many hardware resources. On the other hand, serial implementation targets low-apse applications at the expense of increasing computation latency to m clock cycles. As we target resource-limited IoT applications, we will concentrate on the serial implementation of the adopted modular multiplier algorithm. We can implement the multiplier in a bit-serial fashion or a word-serial fashion. Word-serial implementation realizes better area and time.
complexity than the bit-serial implementation making it more efficient for resource-constrained IoT devices.

The structures of modular multipliers are classified into: serial-in/serial-out (SISO), serial-in/parallel-out (SIPO), parallel-in/serial-out (PISO), and scalable structures. Polynomial SISO multipliers were reported in (Kim et al., 2005; Talaptra et al., 2010; Guo and Wang, 1998; Pan et al., 2013; Lee et al., 2017) and polynomial SIPO multipliers were reported in (Hariri et al., 2008; Kumar et al., 2006; Lee et al., 2012; Xie et al., 2015). PISO multiplier using Type-T Gaussian normal basis was reported in (Namin et al., 2011). Lastly, scalable multiplier structures were reported in (Lee et al., 2007; Chen et al., 2011; Orlando et al., 1999; Bayat-Sarmadi et al., 2014; Gebali and Ibrahim, 2015; Ibrahim et al., 2009; Ibrahim and Gebali, 2017). Compared to other types of word-serial multiplier structures, SISO multiplier structure achieves better area and time complexity (Kim et al., 2005; Pan et al., 2013). Therefore, we will focus on extracting the SISO structure of the utilized algorithm.

We propose in this work a two-dimensional (2-D) word-based SISO modular multiplier processor. The explored construction has the exceptional features of regularity, modularity, concurrency, and local interconnections of the systolic structure, making it more efficient for VLSI implementation. Using the formal mapping technique described in (Gebali, 2011; Ibrahim et al., 2018; Ibrahim et al., 2016; Ibrahim, 2019; Ibrahim et al., 2017; Ibrahim et al., 2017; Gebali and Ibrahim, 2016). The system designer can control the area and power consumption of the explored structure to fit IoT devices. Applying a non-linear scheduling function allows the system designer to control the resulting processor array’s workload and the workload for each processing element. In addition, the latency of the algorithm is also controlled through non-linear task scheduling. The empirical results prove that the offered multiplier structure archives significant savings in area and energy consumption, making it more suitable for resource-limited IoT devices.

The paper can be laid out as follows: Section 2 provides an overview of the adopted modular interleaved multiplication algorithm over the binary extension field GF($2^n$). Section 3 explains the systematic approach used to explore the 2-D word-based SISO processor. Section 4 displays the experimental results and analysis of the developed word-based multiplier structure and its competitor word-based ones previously reported in the literature. Lastly, the conclusion of this work is provided in Section 5.

2. Modular Interleaved Multiplication Algorithm

Assume an order $m$ irreducible polynomial $H(\theta)$ defining the binary extension field GF($2^n$). Also, assume the two polynomials $X(\theta)$ and $Y(\theta)$ in this field. We can write:

$$H(\theta) = \sum_{j=0}^{m-1} h_j \theta^j$$
$$X(\theta) = \sum_{j=0}^{m-1} x_j \theta^j$$
$$Y(\theta) = \sum_{j=0}^{m-1} y_j \theta^j$$

where $h_j, x_j, y_j \in GF(2)$.

We can perform modular multiplication as the product of $X(\theta)$ and $Y(\theta)$ and reducing the result using $H(\theta)$. The following mathematical expression shows the interleaved method used to perform this operation:

$$Z(\theta) = X(\theta)Y(\theta) \mod H(\theta)$$
$$= X(\theta) \left( y_0 + y_1 \theta + \cdots + y_{m-1} \theta^{m-1} \right) \mod H(\theta)$$
$$= y_0 X(\theta) + y_1 X(\theta) \theta \mod H(\theta) + \cdots + y_{m-1} X(\theta) \theta^{m-1} \mod H(\theta)$$

Assume $X^{(i+1)}(\theta) = 0X(\theta) \mod H(\theta)$, $(0 \leq i \leq m - 1)$. The following recursive expression can be used to obtain $X^{(i+1)}(\theta)$ from $X^{(i)}(\theta)$:

$$X^{(i+1)}(\theta) = \left( \sum_{j=0}^{m-1} x_j \theta^j \right) \mod H(\theta)$$
$$= \sum_{j=0}^{m-1} x_j \theta^j \mod H(\theta)$$
$$= x_0 + \sum_{j=0}^{m-1} h_j X^{(i)}(\theta)^j$$

With initial value $X^0(\theta) = X(\theta)$, Eq. (5) can be represented in the bit-level form at iteration $i$ as follows:

$$x_{i,j+1} = x_{i,m-1} + h_j, 0 \leq j \leq m - 1$$
with $x_{i,0} = 0$ for $0 \leq i \leq m - 1$.

Algorithm 1: Interleaved Modular Multiplication Algorithm

Input: $X(\theta), Y(\theta),$ and $H(\theta)$
Output: $Z(\theta) = X(\theta) \times Y(\theta) \mod H(\theta)$
Initialization:
$X^0(\theta) = X(\theta), Z^0(\theta) \leftarrow 0$
Algorithm:
1: for $0 \leq i \leq m - 1$
2: $X^{i+1}(\theta) = 0, X^{i}(\theta) \mod H(\theta)$
3: $Z^{i+1}(\theta) = Z^{i}(\theta) + y_i X^{i}(\theta)$
4: end for

Algorithm 2: Interleaved Modular Multiplication Algorithm in the bit-level form.

Input: $X(\theta), Y(\theta),$ and $H(\theta)$
Output: $Z(\theta) = X(\theta) \times Y(\theta) \mod H(\theta)$
Initialization:
$X^0 = (x_{0,m-1}, \ldots, x_00) \leftarrow (x_{m-1}, \ldots, x_0)$
$Z^0 = (z_{0,m-1}, \ldots, z_00) \leftarrow (0, \ldots, 0)$
Algorithm:
1: for $0 \leq i \leq m - 1$
2: for $0 \leq j \leq m - 1$
3: $x_{i,j+1} = x_{i,m-1} + h_j$
4: $z_{i,j+1} = z_{i,j} + y_i x_{i,j}$
5: end for
6: end for
3. Exploration of the word-based 2D SISO Multiplier

In this section, we will follow a systematic methodology, previously reported by the second author in (Gebali, 2011), to extract the word-based 2D SISO multiplier structure. The approach starts by extracting the data dependency graph (DG) of the adopted multiplier algorithm. Then, non-linear scheduling and projection functions are applied to nodes of the DG to extract the systolic multiplier processor. The details of this approach are given in the following subsections.

3.1. Algorithm Dependence Graph

The data dependence graph (DG) associated with the iterations defining the modular multiplication in Eqs. (6) and (8) is obtained from the iterations defining the modular multiplication in Eqs. (6) and (8) following the guidelines set forth in (Gebali, 2011). The two equations use two iteration indices i and j to define a 2-D integer domain \( D \in \mathbb{Z}^2 \). The details of the DG for the case when \( m = 5 \) are shown in Fig. 1. The operations in Eqs. (6) and (8) are executed at the nodes. The signals of \( x_i \) and \( y_i \) are represented by the horizontal lines. The updated signals \( x_{i-1} \) and \( x_j \) are required to compute \( x_{i-1} \) and \( x_{i+1} \), respectively. Signal \( x_{i-1} \) are obtained at column \( m - 1 \) and broadcasted horizontally as shown in Fig. 1. Input signals \( x_0^{m} \) and \( y_0 \) are supplied at the top row and the output signals \( z_0^{m} \) are obtained from the bottom row.

3.2. Node Scheduling and Projection Functions

The DG of Fig. 1 can be used for design space exploration by choosing node scheduling and projection functions based on the approach discussed in (Gebali, 2011).

![Diagram of the adopted multiplication algorithm for \( m = 5 \).](image)

We will not use the linear scheduling and projection functions since they provide few options in choosing the resulting processor array area, latency, processing elements workload, or overall system workload. In this work, we use the non-linear node scheduling and projection techniques. This choice affords a rich set of design options to optimize the resulting systolic array area, latency, processing elements workload, and overall system workload.

Our goal is to design a SISO multiplier that requires supplying the input polynomials \( X, Y, \) and \( H \) in word-serial fashion. The resulting reduced polynomial \( Z \) is also obtained in a word-serial manner. Assume the goal of the system designer is to simultaneously process \( k \)-bits of each polynomial at the same time and obtain \( k \) bits of the output polynomial. The following parts explain the steps to be used by the system designer.

3.2.1. Non-Linear Task Scheduling

The nonlinear scheduling technique in Gebali (2011), is used to partition domain \( D \) into \( k \times k \) equitemporal zones or clusters. The choice of \( k \) allows the system designer to control the number of bits of the input or output polynomials to be processed simultaneously. This indirectly affects the system area, speed, and latency.

We choose the following non-linear scheduling function to assign timing to each node \( p \) of the DG:

\[
l(p) = \left\lceil \frac{m}{k} \right\rceil \left\lfloor \frac{i}{k} \right\rfloor + \left\lfloor \frac{m - 1 - j}{k} \right\rfloor + 1
\]

where \( l(p) \) is the time assigned to node \( p \) of the DG, \( 0 < i < m + 1, \lambda < j < m - 1 \) with \( \lambda \) given by:

\[
\lambda = k \left\lceil \frac{m}{k} \right\rceil - m
\]

\( \lambda \) represents the number of columns and rows that must be added to the DG in order to make their number an integer multiple of \( k \). For the case shown in Fig. 2, where \( m = 5 \) and \( k = 2 \), we have \( \lambda = 1 \) which implies adding one column on the left and one row at the bottom. The blue boxes highlight the equitemporal zones (the cluster of nodes having the same time values).

Fig. 3 shows the scheduling time for the nodes of the DG when \( m = 5 \) and \( k = 4 \). In this case, we have \( \lambda = 3 \), which implies adding three columns on the left and three rows at the bottom. Examination of Fig. 2 or Fig. 3 reveals that any block I receives two inputs from the north and east directions and produces two outputs from the south and west directions. The times associated with these inputs and outputs are summarized in Table 1. We note that the inputs at the top row produce the outputs of the right column. Similarly, the inputs at the left column produce the outputs at the bottom row. Therefore, the number of iterations for the modular multiplication should be given by:

\[
\#\text{Iterations} = \left\lceil \frac{m}{k} \right\rceil + 1
\]

This means that the first product of the proposed multiplier will be available on the output bus after \( \left\lceil \frac{m}{k} \right\rceil + 1 \) clock cycles. Through each subsequent clock cycle, we will have another product. Therefore, the throughput should be \( \frac{1}{\left\lceil \frac{m}{k} \right\rceil + 1} \).

3.2.2. Non-Linear Task Projection

Figs. 2 and 3 indicate that the \( k \times k \) equitemporal zones execute at the same time. This observation and the projection technique explained in Gebali (2011) produce the following nonlinear task projection function:

\[
\overline{h}(r, s) = \lfloor r \mod k \rfloor \mod k
\]

The extracted projection function maps the \( k \times k \) node clusters to a single processor array. The processor array consists of a two-dimensional \( k \times k \) processing elements (PEs). The whole system
is shown in Fig. 4. Notice that we used two input registers X and XL to store input X. Register X stores k bit values starting from bit $x_{m-1}^0$, while register XL stores k bit values starting from bit $x_{m-2}^0$.

The intermediate words of Z, XL, and X are pipelined through the shift-right registers SHR-Z, SHR-XL, and SHR-X, respectively.

The content of H is pipelined through the rotate-right register ROR-H. The registers are based on k-bits words and the size is $d = \lfloor \frac{M}{k} \rfloor$. Fig. 4 illustrates the word update at the bottom outputs of the processor array.

The details of each PE are shown in Fig. 5 for the case when $m = 5$ and $k = 4$. Two tri-state buffers are used to select between signals $x_{m-1}^i$ and $x_d$. Control signal $t$ is activated ($t = 1$) at time instances $l = q \cdot \lfloor m/k \rfloor + 1, 0 \leq q < \lfloor m/k \rfloor$, enabling the tri-state buffers $Tr$ to pass $x_{m-1}^i$ signals. Signals $x_{m-1}^i$, and the input bits of $y_{e}$, are broadcast to the processors to compute the intermediate results $Z, X$, and $XL$. Signal $t$ is deactivated at $t = 0$, enabling the tri-state buffers $Tr$ to pass the intermediate $x_d$ signals as shown in Fig. 5. At time instances $l = q \cdot \lfloor m/k \rfloor, 1 \leq q < \lfloor m/k \rfloor$, control signal $n$ will be deactivated ($n = 0$) to force zero values of $XL$ signals, as shown at the left of Fig. 3. The logic circuit of the PE is shown in Fig. 6.

We can describe the operation details of the 2-D SISO multiplier for general values of $m$ and $k$ as follows:

1. At the first time instance $l = 1$, the controller activates the select signal (S) of all MUXes, depicted in Fig. 4, to allow the k MSB bits of $X, XL, H$ to be input to the processor array shown in Fig. 4. This ensures $Z$ has zero initial value as described in Algorithm 1, the controller resets the right-shift register SHR-Z at the first time instance. Also, at this time instance, the controller activates the control signal $t$ ($t = 1$) to enable the tri-state buffers $Tr$ in Fig. 5 to move bits of $x_{m-1}^i, 1 \leq i \leq k$ to the

is shown in Fig. 4. Notice that we used two input registers X and XL to store input X. Register X stores k bit values starting from bit $x_{m-1}^0$, while register XL stores k bit values starting from bit $x_{m-2}^0$.

The intermediate words of Z, XL, and X are pipelined through the shift-right registers SHR-Z, SHR-XL, and SHR-X, respectively.
A. Ibrahim and F. Gebali
Journal of King Saud University – Computer and Information Sciences xxx (xxxx) xxx

2. At time 1, notice that the least significant remaining PEs in the same row of the processor array. We also casted horizontally, at the first time instance, to the PEs nodes to supply processor array inputs. These words together with $X_{m-1}^i$, $Y_i$, $1 \leq i \leq k$, are used to calculate in sequence the partial words of $Z$, $X$, and $XL$. These words are pipelined through the shift-right registers $SHR-Z$, $SHR-X$, and $SHR-XL$, respectively, as displayed in Fig. 4. Also, the fixed words of $H$ are pipelined through the rotate-right register $ROR-H$. It is worth noticing that the depth of the shift-right register $SHR-Z$ keeps the initial values of $Z$ having zero values during these time instances. Also, the updated bits $X_{m-1}^i$, are pipelined through the shift-right register $SHR-Xm$ of the depth size $d$ during these time instances.

3. At time $l > \left[ \frac{n}{d} \right]$, the controller deactivates all MUXes ($S = 0$) to pass in sequence the resulted partial words $Z,X, XL$ kept in shift-right registers $SHR-Z$, $SHR-X$, $SHR-XL$, respectively, and the fixed $H$ words kept in rotate-right resister $ROR-H$ to the processor array. Also, at the same time instances, the updated bits $X_{m-1}^i$, kept in SHRXm register are passed to the processor array block. These bits are utilized alongside the bits of $Y_{m-1}^i$, $qk < i \leq q(k + 1), 1 \leq q \leq \left[ \frac{n}{d} \right] - 1$, to compute in sequence the partial words of $Z$, $X$, and $XL$.

4. During times $l = \left[ \frac{n}{d} \right] + 1, 1 \leq q \leq \left[ \frac{n}{d} \right]$, the controller resets the control signal $n$ in Fig. 5 to force zero values of $XL$ in Fig. 3. Signal $n$ is set to 1 for the remaining times.

5. During times $l > \left[ \frac{n}{d} \right] + 1$, the controller activates the load signal of the register $Z$, Fig. 4, to pass in sequence the resulted output words of $Z$.

We added delay elements (D Flip-Flop blocks) to the processor array as shown in Fig. 5 to ensure that there is always one time step difference between the words of $Z$, $X$, and $H$ above and below the delay elements. These elements synchronize the operation of the processor array by lagging the words of $Z$, $X$, and $H$ by one time instance to arrive at the same time of the resulted bits of $X_0$. We notice from Fig. 3 that the $x_0$ bits are generated starting from the second time instance and this results in the extra 1 term in Eq. (11).

4. Experimental Results and Discussion

We estimated the area and delay complexities of the proposed 2-D word-based multiplier structure and the efficient word-based ones in the literature (Pan et al., 2013; Xie et al., 2015; Hua et al., 2013; Chen et al., 2008). The area estimation is based on counting the number of basic logic gates and components (AND gates, Tri-state buffers, XOR gates, Flip-Flops (FFs), and MUXs) of the compared multiplier structures. In this work, we define latency as the number of clock cycles required to complete the multiplication operation. We also define critical path delay (CPD) as the delay of the basic gates/components of the longest path of the multiplier logic circuit. Table 2 shows the estimated area and time results of the examined multiplier structures. We can interpret the symbols utilized in Table 2 as follows:

1. $k$ designates the word size of the multiplier structures.
2. $\delta_A$ designates the delay of the basic 2-input AND gate.
3. $\delta_X$ designates the delay of the basic 2-input XOR gate.
4. $\delta_{MAX}$ designates the delay of the 2-to-1 MUX.
5. $\eta_1 = 7m + m(\lfloor \log m \rfloor) + 3$ represents the total number of FFs used in the multiplier structure of Pan et al. (2013).
6. $\eta_2 = 2k^2 + 2k(\lfloor m/k \rfloor) + 4k + 1$ represents the total number of FFs used in the multiplier structure of Hua et al. (2013).
7. $\eta_3 = 2k^2 + 3k\lfloor m/k \rfloor + 2k$ represents the total number of FFs used in the multiplier structure of Chen et al. (2008).
8. $\beta_1 = k + \lfloor m/k \rfloor^2 + \lfloor m/k \rfloor$ depicts the latency of the multiplier structure of Chen et al. (2008).
9. $\gamma_1 = c + (\lfloor \log k \rfloor + 1)c_0$ is the estimated CPD of the multiplier structure of Pan et al. (2013).
Table 2
Area and time complexities of the word-based multipliers.

| Design          | Tri-State | AND | XOR | MUXs  | FFs   | Latency | CPD  |
|-----------------|-----------|-----|-----|-------|-------|---------|------|
| Xie et al. (2015)| 0         | 2mk|     | 2mk + 6m + 6k + 6 | 0       | 4mk + 4m + 2k | 2(\lceil |\frac{mk + 2}{2}\rceil) + 6 | \frac{mk}{2}\rceil) | 2(\lceil |\frac{mk}{2}\rceil) + 2(\lceil |\frac{mk}{2}\rceil) | 2\lceil |\frac{mk}{2}\rceil) |
| Pan et al. (2013)| 0         |     |     |       | \eta_1 |         |       | \gamma_1 |
| Hua et al. (2013)| 0         |     |     |       | \eta_2 |         |       | \gamma_2 |
| Chen et al. (2008)| 0         |     | \gamma^2 | \gamma^2 | \eta_3 | \gamma_3 |       | \gamma_4 |
| Proposed        | 2k        | \gamma^2 + k | \gamma^2 + 2k | \gamma^2 | 3k + 1 | 4k(\lceil |\frac{mk}{2}\rceil) + 1 | \gamma_4 |

(1) Area of 3-input XOR gate equals 1.5 \times a 2-input XOR gate.
(2) Multiplier in Chen et al. (2008) uses switches that have complexity equivalent to MUX.

10. \gamma_2 = \gamma_A + 2\gamma_X is the estimated CPD of the multiplier structure of Hua et al. (2013).
11. \gamma_3 = \gamma_A + \gamma_X is the estimated CPD of the multiplier structure of Chen et al. (2008).
12. \gamma_4 = k\gamma_A + k\gamma_X + \gamma_{\text{MUX}} is the estimated CPD of the multiplier structure of proposed multiplier structure.

It is worth reporting that the estimated number of FFs includes the input/output registers. This ensures a fair comparison between the multiplier structures.

By investigating the area expressions in Table 2, we can conclude the following:

1. The multipliers in (Pan et al., 2013; Xie et al., 2015) have area complexity approximately of order \( e(\gamma k) \).
2. The area complexity of all other designs are of order \( e(\gamma k^2) \) excluding the MUXes and FFs of the suggested multiplier structure are of order \( e(\gamma k) \) and \( e(\gamma mk/k) \), respectively.
3. The area complexity of the multiplier structures of Pan et al. (2013) and Xie et al. (2015) is higher than the area of the other multiplier structures. This is attributed to the value of field size \( m \) is extremely larger than the values of the embedded word size \( k \).
4. Our proposed multiplier has the lowest number of FFs relative to the other multipliers. This is attributed to the area complexity of the proposed multiplier is of order \( e(\gamma k/mk) \) compared to \( e(\gamma k^2) \) and \( e(\gamma mk/mk) \) for the other multiplier structures.
5. Increasing the values of the word size \( k \) will not significantly increase the number of FFs of the proposed multiplier structure. This is due to the area complexity of the FFs of the proposed multiplier structure is of order \( e(\gamma mk/k) \).

The FFs consume a larger chip area than the other logic components, as indicated in Rabaey (2002). Therefore, reducing the number of FFs will significantly reduce the overall area of the multiplier structures. As we mentioned before, increasing the word size will not significantly increase the total number of FFs in the proposed multiplier structures. This will result in the overall area of the proposed multiplier structure will not significantly increase as \( k \) increases.

By analyzing the latency expressions in Table 2, we can observe the following:

1. The multiplier of Hua et al. (2013) has the least latency compared to the other multiplier structures.
2. The latency results obtained in Table 3 for the standard value \( m = 409 \) and word sizes \( k = 8, 16, 32 \) confirm that the latency expression of the proposed multiplier structure will lead to a more significant latency than that of the multiplier structures in (Xie et al., 2015; Pan et al., 2013) and more economical latency than that of the Hua et al. (2013) and Chen et al. (2008) multiplier structures.
3. The latency decreases when word size \( k \) increases. This is attributed to the latency expressions are inversely proportional to \( k \).

By investigating CPD expressions, we can observe the following:

1. CPD expressions of Xie et al. (2015), Hua et al. (2013), and Chen et al. (2008) multiplier designs do not depend on the word sizes \( k \). Therefore, they will always have fixed CPD values for all values of \( k \).
2. CPD expressions of Pan et al. (2013) and the proposed multiplier structures directly depend on \( k \). Therefore, the CPD values of these multipliers will increase as \( k \) increases.

Since it is difficult to qualitatively estimate the latency decrease and CPD increase as \( k \) increases, we can not precisely expect which multiplier structure has the best computation time. However, the quantitative results obtained in Table 3 will verify which multiplier structure outperforms the others in computation time.

All the multiplier structures were modelled using VHDL language. The modelled multipliers are synthesized for the standard field size \( m = 409 \) and \( k = 8, 16, 32 \). For synthesizing the modelled multipliers, we used Synopsys tools version 2005.09-SP2 with the NanGate (15 nm, 0.8 V) Open Cell Library. The typical corner of (VDD = 0.8 V and T.j = 25 μC) and unit drive strength are used for all the utilized primitives. We recorded the switching activities of each design during the simulation process (using Mentor Graphics ModelSim SE 6.0a) in the Switching Activity Interchange Format (SAIF) file. This file is read by Synopsys power compiler to have the power report.

Table 3 displays the design metrics - Latency, Area (A), CPD, maximum frequency (F), Total Computation Time (T), Consumed Power (P), and Consumed Energy (E) - used to compare the adopted word-based multiplier structures. Area and CPD are obtained from the synthesis tools. The maximum operating frequency is obtained by calculating the multiplicative inverse of the CPD values. The area is normalized by the area of a 2-input NAND gate. The total computation time can be defined as the required time to complete one product operation. It is obtained by multiplying latency and CPD. The consumed power is measured at a 1 kHz frequency. The consumed energy results are obtained as the product of P and T.

We can read the performance results obtained in Table 3 as follows:

1. Our proposed multiplier outperforms the other multipliers in terms of A. It significantly reduces area at all embedded word sizes \( k \) by rates varying from 33.5% to 94.6% at \( k = 8, 30.7% \) to 95.1% at \( k = 16, \) and 31.2% to 96.3% at \( k = 32. \)
2. The multiplier structure of Pan et al. (2013) outperforms the other multiplier structures, including the offered one, in terms of the computation time at \( k = 8 \) and \( k = 16. \) It saves at least 45.8% of time at \( k = 8 \) and 9.3% at \( k = 16. \)
The area and consumed energy.
other resource-limited applications that impose limitations on wearable and implantable medical devices. Also, it can be used in crypto-processors in resource-limited medical IoT devices such as design can be efficiently utilized in the implementation of computation speed compared to some of its competitors, the per-
embedded word sizes. Despite the proposed design having a lower area, consumed power, and consumed energy for all the common the proposed multiplier outperforms its competitors in terms of consumption, and vice versa. In this paper, we mainly target the other multiplier structures. The reduced area reduces the parasitic capacitance and consequently the dynamic and overlap energy loss of the circuit. Dynamic and overlap power loss is a major contributor to power loss in electronic circuits. The proposed multiplier structure decreases power consumption at all $k$ values by 31.5% to 98.7% at $k = 8$, 27.4% to 98.9% at $k = 16$, and 19.7% to 98.7% at $k = 32$.

5. Our proposed multiplier outperforms the other multiplier structures in terms of consumed energy. It saves energy by rates varying from 37.6% to 98.1% at $k = 8$, 43.5% to 98.3% at $k = 16$, and 42.8% to 98.5% at $k = 32$. The energy reduction results from the significant saving of the consumed power and the reasonable computation time of the offered multiplier structure over the other multiplier structures.

In our comparison, we focused on all design metrics (Area, Time, Power, and Energy) to have a fair comparison between the proposed design and its competitor ones. It is known that there is always a trade-off between design metrics. This means that having more resources leads to having more area, speed, power consumption, and vice versa. In this paper, we mainly target resource-constrained IoT applications that have more restrictions on the area and consumed energy. The obtained results show that the proposed multiplier outperforms its competitors in terms of area, consumed power, and consumed energy for all the common embedded word sizes. Despite the proposed design having a lower computation speed compared to some of its competitors, the performance is still in the acceptable range. Therefore, the proposed design can be efficiently utilized in the implementation of crypto-processors in resource-limited medical IoT devices such as wearable and implantable medical devices. Also, it can be used in other resource-limited applications that impose limitations on the area and consumed energy.

5. Summary and Conclusion

In this manuscript, we presented a compact and practical 2-D word-based SISO processor for the modular multiplier over $GF(2^m)$. The proposed processor architecture is derived using a formal and systematic technique for mapping regular iterative algorithms (RIA) onto processor arrays. The methodology allows the system designer to control the workload of the entire processor array system and the workload of each processing element. Managing the processor word size provides control of system speed, latency, and area. The proposed processor size can be manipulated to fit the expected chip area, making the implementation of the offered multiplier processor more efficient in resource-limited IoT devices. The regularity and modularity of the proposed processor array make it more suitable for implementation in ASIC technology. The obtained results show that the proposed multiplier processor has the advantage of reducing area, power consumption, and consumed energy over the other competitor word-based multiplier designs. Thus, it can be perfectly used to implement crypto-processors in resource-limited medical IoT devices, such as wearable and implantable medical devices, and other resource-limited devices such as smart cards, RFID devices, and wireless sensor nodes. In future work, we intend to implement the entire ECC cryptographic processor based on the proposed multiplier accelerator structure to estimate the overall savings in area and energy consumed with the entire system.

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 National Research Council of Canada (NRC) under Grant No. 51291–52200.

References

Atzori, L., Iera, A., Morabito, G., 2010. The internet of things: A survey. Computer networks 54 (15), 2787–2805.
Bayat-Sarmadi, S., Kermani, M.M., Azaderakhsh, R., Lee, C.-Y., 2014. Dual-basis superserial multipliers for secure applications and lightweight cryptographic architectures. IEEE Trans. Circ. and Sys.-II 61 (2), 125–129.
Belcher, R.H., Phillips, J., Virgin, F., Werkhoven, J., Whigham, A., Wilcox, L., Wootten, C.T., 2021. Pediatric otolaryngology telehealth in response to covid-19 pandemic: Lessons learned and impact on the future management of pediatric patients. Ann. Otology, Rhinology Laryngology 130 (7), 788–795.
Chen, C.-C., Lee, C.-Y., Lu, E.-H., 2008. Scalable and systolic Montgomery multipliers over $GF(2^m)$. IEE Transactions on Fundamentals Electronics. Commun. Computer Sci. E91-A 7, 1763–1771.
Chen, L.H., Chang, P.L., Lee, C.-Y., Yang, Y.K., 2011. Scalable and systolic dual basis multiplier over $GF(2^m)$. Int. J. Innovative Computing, Information Control 7 (3), 1193–1208.
