Abstract:

IoT is extensively used in many infrastructure applications including telehealth. However, IoT presents the weakest link in the system security since they often have low processing and power resources. It becomes important to implement the necessary security primitives in these devices using extremely efficient finite field hardware structures. Modular multiplication is the core of many cryptographic operators and we present in this work a modular multiplier accelerator structure that provides the system designer with the ability to manage area, delay and energy consumption through selecting the appropriate processor word size $l$. We achieve area reduction that varied from 63.70%-97.02% for $l = 8$, 57.12%-96.97% for $l = 16$, and 71.37%-97.08% for $l = 32$. The energy reduction ranged from 61.20%-98.83% for $l = 8$, 67.73%-98.26% for $l = 16$, and 76.14%-98.81% for $l = 32$. These results indicate that the proposed modular multiplier is suitable for IoT devices in telehealth applications.
Compact Word-Serial Modular Multiplier Accelerator Structure for Cryptographic Processors in Telehealth IoT Network

Atef Ibrahimab, Fayez Gebali

aComputer Engineering Department, College of Computer Engineering and Sciences, Prince Sattam Bin Abdulaziz University
Alkhair, Saudi Arabia
email: attif.ali2002@yahoo.com, aa.mohamed@psau.edu.sa

bECE Department, University of Victoria, Victoria, BC, Canada
atief@ece.uvic.ca, fayez@ece.uvic.ca

Abstract

IoT is extensively used in many infrastructure applications including telehealth. However, IoT presents the weakest link in the system security since they often have low processing and power resources. It becomes important to implement the necessary security primitives in these devices using extremely efficient finite field hardware structures. Modular multiplication is the core of many cryptographic operators and we present in this work a modular multiplier accelerator structure that provides the system designer with the ability to manage area, delay and energy consumption through selecting the appropriate processor word size l.

We achieve area reduction that varied from 63.70%-97.02% for l = 8, 57.12%-96.97% for l = 16, and 71.37%-97.08% for l = 32. The energy reduction ranged from 61.20%-98.83% for l = 8, 67.73%-98.26% for l = 16, and 76.14%-98.81% for l = 32. These results indicate that the proposed modular multiplier is suitable for IoT devices in telehealth applications.

Key Words: modular multipliers, embedded security, Telehealth IoT network, hardware security, parallel computing, cryptography.

1. Introduction and Related Work

Telehealth, is one of the many emerging infrastructure applications that relay on the Internet of Things (IoT) to provide services to remote users such as stay-at-home patients and providing quality healthcare to remote communities [1, 2]. Figure 1 shows a telehealth system that relies on IoT edge devices to deliver healthcare to remote locations. The main entities of a telehealth system are: (a) Server which could be a hospital or medical centre which is naturally considered a hardware root-of-trust (HRoT) due to the layered security measures implemented. (b) Internet cloud which is in general an insecure communication medium. (c) Gateway that provides interface between the IoT edge devices and the Internet. (d) Edge IoT devices that comprise sensors and actuators to measure and deliver medications to remote patients. (e) Mobile devices that allow healthcare practitioners (doctors/nurses) to remotely connect to the telehealth system. It is clear from the figure that there are many opportunities for attack due to the use of diverse hardware platforms, diverse operating systems, insecure wireless communication media, limited processing power for many of the system entities, and the sheer number of people involved in the operation of the system [3, 4].

Securing any system (telehealth or otherwise) implies many features that include integrity, confidentiality, authentication, non-repudiation, and availability. These security features are implemented using fundamental finite field arithmetic operations where modular multiplication is found in almost all of them. Given the power and delay restrictions for most of the devices used in most systems, e.g. telehealth, elliptic curve cryptography (ECC) became the encryption technique of choice due to its high level of security with shorter key lengths compared to common approaches such as RSA [5, 6, 7].

An essential operation in ECC arithmetic is modular multiplication. There is an extensive body of literature covering modular multiplications in both prime fields $GF(p)$ and binary extension fields $GF(2^n)$. Most of the proposed multipliers possessed high area and delay complexities which makes them unsuitable for the resource-constrained IoT edge devices [8], [9], [10]. To overcome these limitations, several authors developed word-serial modular multipliers. Systolic approaches were re-
ported in [11, 12, 13, 14] and non-systolic designs were report
in [15, 16, 17, 18]. Other authors attempted to save power and
area by merging the modular multiplication and modular squar
ing operations [9, 10, 19]. However the resulting structures
were not suitable for resource-constrained IoT devices due to
their high area and power requirements.

Most of the reported modular multiplier structures are classi
ced as one-of-a-kind structures. Ad hoc approaches are adopted
with no consideration on how the structure can be modified
to optimize system performance parameters such as latency,
throughput, power and area requirements. The authors of this
article presented a systematic methodology for implementing
the modular multiplication algorithm based on the algebraic ap
proach first proposed by the first author [20]. The systematic
methodology applied linear mappings to obtain modular mul
tiplier structures. However, linear mappings have limited abil
ities both in terms of number of parallel processing elements
(PPE) and also the timing strategies that could be developed.

This paper proposes to use nonlinear techniques for mapping
the algorithm onto parallel PEs and also to obtain more flexible
timing strategies. The goal of the paper is to obtain word-serial
processor accelerator for modular multiplication operation. The
resulting structure allow the designer ability to control the PE
workload and the algorithm latency. The experimental results
confirm that the proposed multiplier outperforms the efficient
word-serial ones previously reported in the literature in terms
of area and consumed energy for various embedded word-sizes.
These design features makes the proposed design more suit-
able for embedded telehealth applications and other resource-
constrained IoT applications.

The outlining of the paper is as follows. Section 2 briefly
describes the adopted modular multiplication algorithm and ex-
hibits the details of its dependency graph. Section 3 presents
the followed approach to extract the modular multiplier word-
serial accelerator structure with its related logic details. Section
4 shows the realized implementation results. Section 5 con
cludes the recommended work.

2. Algorithm of Interleaved Modular Multiplication
We can perform modular multiplication over \(GF(2^m)\) by
multiplying two polynomials \(P(\gamma)\) and \(Q(\gamma)\) and reducing the
result using the reduction polynomial \(T(\gamma)\) as:

\[
S(\gamma) = P(\gamma)Q(\gamma) \mod T(\gamma)
\]

(1)

The general polynomial format of \(P(\gamma)\), \(Q(\gamma)\), and \(T(\gamma)\) can be
given as:

\[
P(\gamma) = \sum_{j=0}^{m-1} p_j \gamma^j = (p_0 + p_1 \gamma^1 + \cdots + p_{m-1} \gamma^{m-1})
\]

(2)

\[
Q(\gamma) = \sum_{j=0}^{m-1} q_j \gamma^j = (q_0 + q_1 \gamma^1 + \cdots + q_{m-1} \gamma^{m-1})
\]

(3)

\[
T(\gamma) = \sum_{j=0}^{m} t_j \gamma^j = (t_0 + t_1 \gamma^1 + \cdots + t_m \gamma^m)
\]

(4)

with \(p_j, q_j, t_j \in GF(2)\). The expansion of Eq. (1) can be represented as:

\[
S(\gamma) = P(\gamma)(q_0 + q_1 \gamma^1 + \cdots + q_{m-1} \gamma^{m-1}) \mod T(\gamma)
\]

(5)

we can arrange Eq. (1) in the interleaved form as:

\[
S(\gamma) = q_0[\gamma T(\gamma)] + q_1[\gamma^2 T(\gamma)] + \cdots + q_{m-1}[\gamma^{m-1} T(\gamma)]
\]

(6)

We choose to drop \(\gamma\) from polynomials \(P(\gamma), Q(\gamma), S(\gamma),\) and
\(T(\gamma)\) to simplify the upcoming expressions. Investigating Eq.
(6), we notice that the multiplication product can be produced by
accumulating the terms \(q_i[\gamma^j T(\gamma)]\), with \(0 \leq i \leq m - 1\).

Suppose \(P' = P[\gamma T] \mod T\), we can represent \(P'^{i+1}\) in terms of
\(P'\) as \(P'^{i+1} = P[\gamma^i T] \mod T = [P[\gamma^j] T] \mod T = P' \gamma^i T\).
Thus, the recursive form of \(P'^{i+1}\) can be represented as:

\[
P'^{i+1} = P' \gamma \mod T
\]

\[
\sum_{j=0}^{m-1} \gamma^j P'^{j+1} = \sum_{j=0}^{m-1} (P'[\gamma^j] \gamma^j) \mod T
\]

\[
= \sum_{j=0}^{m-1} (P'[\gamma^j]) T
\]

\[
= P'[\gamma^m T] + \sum_{j=0}^{m-2} \gamma^j P'^{j+1}
\]

\[
= P'[\gamma^m j] + \sum_{j=0}^{m-2} \gamma^j P'^{j+1}
\]

(7)

with the initialization condition \(P'^{0} = P\). The term \(\gamma^m T\) in
Eq. (7) is equivalent to \(\sum_{j=0}^{m-1} \gamma^j T\) as proved by [9]. Also, the
term \(\sum_{j=0}^{m-2} \gamma^j P'^{j+1}\) represents a polynomial of order less than \(m\).

The recursive Eq. (7) can be expressed in the bit-level form as:

\[
p'^{j+1} = p'^{j-1} + p'^{m-1} \gamma^j, 0 \leq j \leq m - 1
\]

(8)

with \(p'^{i} = 0\) for \(0 \leq i \leq m - 1\).

The recursive form of partial product \(S'^{i+1}\), \(0 \leq i \leq m - 1\) can be
given from accumulating \(P'\) terms as:

\[
S'^{i+1} = S'^{i} + q_i P'^{i}
\]

\[
\sum_{j=0}^{m-1} \gamma^j T^{j+1} = \sum_{j=0}^{m-1} \gamma^j T^{j+1} + \sum_{j=0}^{m-1} \gamma^j P'^{j+1}
\]

(9)

with \(S'^{\gamma} = 0\) and \(S'^{\gamma} = 0\) represents the final result \(S\). Recursive Eq.
(9) can be expressed in the bit-level form as:

\[
\gamma S'^{i+1} = S'^{i} + q_i P'^{i}
\]

(10)

with \(0 \leq i \leq m - 1, 0 \leq j \leq m - 1, \) and \(\gamma S'^{i} = 0\) for \(0 \leq j \leq m - 1\).

2.1. Dependency Graph
Using reference [20], the dependence graph DG describing
the modular multiplication can be obtained from Eqs. (8), (10).
The indices \(i, j\) in the two equations designate that the DG
can be defined in a two-dimensional integer domain. Index \(i\)
denotes the rows, and index \( j \) denotes the columns. Fig. 2 presents the DG for the field size \( m = 5 \). The circled nodes compute the operations depicted by Eq. (8) and (10). The vertical lines represent the partial product signal \( p_j' \), the multiplier signal \( p_j' \), and the irreducible polynomial coefficient \( t_j \). The horizontal lines represent the broadcast multiplicand signal \( q_i \). The diagonal lines represent the signal \( p_j'_{j-1} \). Signals \( p_j'_{j+1} \) and \( s_j^{j+1} \) requires signals \( p_j'_{j-1} \) and \( p_j' \) to be computed. The last column nodes produce signal \( p_{m-1} \) which is used inside that column nodes as well as broadcasted to the remaining row nodes as indicated in Fig. 2.

As we observe from Fig. 2, the input signals \( s_0^0, p_0^0 \) are fed at the upper row of the DG, and the output signals \( s_m^0 \) are produced from the lower row.

We assume the processor array we would like to develop has \( l \)-bit digit or word size.

A valid nonlinear scheduling function assigns an execution time value to each node or point \( P \) in Fig. 2 according to the nonlinear expression:

\[
\Gamma(P) = i \left\lceil \frac{m}{l} \right\rceil + \frac{m-1-j}{l} + 1
\]

(11)

\( \Gamma(P) \) is the function that assigns a time instance to node \( P(i, j) \) in the DG.

Figure 3 shows the time index values after applying the nonlinear scheduling function in Eq. (11) for the case when \( m = 5 \) and \( l = 3 \). The figure shows the DG points are being grouped horizontally in \( l \)-bit groups having the same execution time value. This ensures that all the bits in a single processor word are executed at the same time. An extra column of nodes is added at the left side of the DG to ensure that the number of columns in the DG is an integer multiple of \( l \). For the general case, we need to add extra \( \theta = \left\lceil \frac{m}{l} \right\rceil - m \) columns, with zero inputs, at the left side of the DG. As we notice from Fig. 3, the output of the multiplier will be available after \( m \left\lceil \frac{m}{l} \right\rceil \) computation steps.

3. Word-Serial Accelerator Structure Exploration

We will follow a formal and systematic methodology we have previously developed in [18, 20, 21, 22, 23, 24, 25, 26] to map the recursive-iterative multiplier algorithm to a processor array and assign an execution schedule to each processing element (PE) in the resulting array.

3.1. Scheduling Function

Consider the two-dimensional dependence graph for the polynomial modular multiplication algorithm shown in Fig. 2.

Figure 2: DG of the polynomial modular multiplication algorithm for the case \( m = 5 \).

Figure 3: DG scheduling for the case \( m = 5 \) and \( l = 3 \).

An important feature of the proposed scheduling function is to provide the system designer with the ability to control the workload of the entire processor array system. For the nonlinear scheduling formula in Eq. (11), we note that only one group of \( l \) bits is active at any given time instance. Therefore, the PE workload is equal in that case to the system workload. Of course the system designer could choose another scheduling function to choose a different system workload.
3.2. Projection Function

The projection function approach discussed in [20], projects several nodes in the DG of Fig. 3 to a single node. This operation is necessary since each $l$ group of nodes in Fig. 3 operates only once. Therefore, to reuse the processing elements, we map several groups into one PE. The system workload in Fig. 3 implies that we need to map all the nodes of the DG into one PE only. We propose the following nonlinear projection function to map a node $P(i,j)$ to a new node $P(x,y)$:

$$P(x,y) = P_{\text{serial}}(i,j)$$  \hspace{1cm} (12)

$$x = i$$  \hspace{1cm} (13)

$$y = m - 1 - j \mod l$$  \hspace{1cm} (14)

$$P_{\text{serial}} = \lceil 1 \mod l \rceil$$  \hspace{1cm} (15)

where ”$l$” is a place holder for the argument [20].

Figure 4 shows the resulting word-serial accelerator structure after applying the assumed projection function to Fig. 3. The system consists of the following components:

1. A processor array block whose word size is $l$
2. Three input registers $T$, $P$, and $PL$
3. One output register $S$
4. Three shift-right registers $SHR-S$, $SHR-pd$ and $SHR-P$
5. Rotate-right register $RO-T$
6. Four 3-input MUXes (two of them inside the processor array block) to select between the inputs and partial results of variables $P$ and $T$.

Register $P$ passes bit values starting from bit $p_{m-1}^0$, while register $PL$ passes the bit values of the word variable starting from bit $p_{m-2}^0$.

The partial results stored in $S$ and $P$ are cycled through the shift registers $SHR-S$ and $SHR-P$, respectively. The fixed words of $T$ are rotated through the rotate-right register $RO-T$.

Observation of Figs. 3 and 4 indicates that the rightmost bit $p_d$ of register $P$ is transferred diagonally to the following node after delayed by $r - 1$ time steps, where $r = \lceil m/l \rceil$, while the remaining vertical and slanted word bits of $P$ are transferred to the following bottom nodes after being blocked by $r$ time steps. Therefore, bit $P_d$ should be passed through the shift-right register, $SHR-pd$, that has depth size $r - 1$ as shown in Fig. 4. Shift-right registers $SHR-P$ and $SHR-S$, and rotate-register $RO-T$ all have the same width and depth sizes of $l$ and $r$, respectively.

Figure 5 displays the details of the processor array block for the case $l = 3$ bits. The processor array contains two types of PEs. Figures 6 and 7 depict the design details of the PEs. The yellow PE in Fig. 3 has two more tri-state buffers managed by the control signal $e$. We will discuss below the role of these extra buffers.

The operation details of the explored multiplier accelerator can be summarized for generic sized $m$ and word size $l$ as follows:

1. Control signal $C$, controlling the selection of all MUXes, activates ($C = 1$) during the first $\lceil m/l \rceil$ clock cycles to feed the input words of operands $T$, $P$, and $PL$ to all PEs of the processor array block. The words are fed starting with the most significant words. Also, the most significant bit $p_{m-1}^0$ is passed to the last PE, $PE_i$, and broadcasted to the remaining PEs. At the first clock cycle, SHR-S is cleared to initialize the $S$ variable with zero values.

2. Control signal $C$ of all MUXes deactivates ($C = 0$) during the remaining clock cycles to feed the resulted intermediate words of $P$ and fixed words of $T$ to all PEs of the processor array block. These words are passed through shift-registers $SHR-P$, $SHR-pd$, and $RO-T$, respectively. Also, the resulted intermediate words of $S$ are fed to all PEs of the processor array block through the shift-register $SHR-S$.

3. Control signal $e$ activates ($e = 1$) at the clock cycles $T = (i\lceil m/l \rceil) + 1, 0 \leq i \leq m$, to enable the Tri-Sate buffer $TR_1$ shown in Fig. 6 to horizontally feed the bits of $p_{m-1}^i$ to the remaining PEs. Also, $q_i$ input bits are broadcasted during the same clock cycles to all PEs in the processor array block. The control signal $e$ deactivates ($e = 0$) during the remaining clock cycles to enable the Tri-Sate buffer $TR_2$ displayed in Fig. 6 to feed the bits of $p_d$ through the shift-register $SHR-pd$ to the input of the processor array block as shown in Fig. 4.

4. Control signal $v$, shown in Fig. 5, deactivates ($v = 0$) at clock cycles $T = (i+1)\lceil m/l \rceil, 0 \leq i \leq m$, to force zero bit values to the $P$ words shown at the leftmost side of the DG, Fig. 3. Control signal $v$ activates ($v = 1$) at the remaining clock cycles to feed the $P_d$ signal through the
4. Complexities Analysis

The area, delay, and consumed energy complexities of the proposed multiplier are reported and compared to other efficient word-serial multipliers reported in [13, 27, 28, 29].

Table 1 summarizes the area and delay complexities of recommended multiplier and the previously reported efficient word-serial ones. The total count of logic gates/components in the accelerator structure estimates the area complexity. The entire number of clock cycles needed to produce the product represents the latency (L) of the multiplier. The whole gate delays in the longest path of the logic circuit represent the critical path delay (CPD) of the multiplier structure. The product of latency and critical path delay (CPD) estimates the delay complexity.

We can represent the delays of the 2-input AND, 2-input XOR, and 2-to-1 MUX by \( \tau_A \), \( \tau_X \), and \( \tau_{MUX} \) symbols, respectively.

The following describes the remaining notations in Table 1:

\[
R_1 = 7m + m([\log_2 m]) + l + 3; \quad R_2 = 2l + 2l([m/l]) + 4l + 1; \quad R_3 = 2l + 3l([m/l]) + 2l; \quad D_1 = l + [m/l]^2 + [m/l]; \quad \eta_1 = \tau_A + ((\log_2 l) + 1)\tau_X \; ; \; \eta_2 = \tau_X + 2\tau_X \; ; \; \eta_3 = \tau_A + \tau_X \; ; \; \eta_4 = 2\tau_A + \tau_X + \tau_{MUX}.
\]

Input and output flip-flops of each multiplier structure are added to the total estimated number of its flop-flops. AS we notice from Table 1, the proposed multiplier structure has a significant reduction in the area compared to other multiplier structures due to having area complexity of order \( O(l) \).

We modeled the proposed multiplier structure and the adopted ones using VHDL hardware language and synthesized them for the recommended field size \( n = 409 \) and embedded word sizes of \( l = 8 \), \( l = 16 \), and \( l = 32 \). The synthesis was performed using NanGate Open Cell Library (15nm, 0.8V) and Synopsys tools version 2005.09-SP2. The following describes the synthesis design parameters obtained in Table 2:

1. Area (A) results are obtained in terms of the 2-input NAND gate and represented in units of kilo-gates kgates.
2. Critical Path Delay (CPD) is represented in pico-second ps time unit.
3. Total computation time (T) is represented in nano-second ns time unit.
4. Consumed power (P) is obtained at a frequency of 1 KHz in units of mili-watt mW.
5. Consumed energy (E) is obtained as the product of P and T in units of femto-joul fJ.

Figures 8, 9, 10, and 11 compares the obtained results of A, T, P, and E, respectively, of the proposed multiplier structure with the adopted ones.

Figure 8 depicts that the proposed multiplier structure saves a significant amount of area ranging from 63.70% to 97.02% at \( l = 8 \), 57.12% to 96.97% at \( l = 16 \), and 71.37% to 97.08% at...
Table 1: Estimation of area and delay for the adopted word-serial multipliers.

<table>
<thead>
<tr>
<th>Multiplier</th>
<th>Tri-State</th>
<th>AND</th>
<th>XOR</th>
<th>MUXes</th>
<th>Flip-Flops</th>
<th>Latency</th>
<th>CPD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Xie [27]</td>
<td>0</td>
<td>2ml</td>
<td>2ml + 6m − 62 + 6</td>
<td>0</td>
<td>4ml + 4m + 2l</td>
<td>2l/m[l] + 2log₂ l</td>
<td>2D₇</td>
</tr>
<tr>
<td>Pan [13]</td>
<td>0</td>
<td>m√m</td>
<td>√m(2l + l) + l</td>
<td>0</td>
<td>R₁</td>
<td>l/2√m[1]</td>
<td>η₁</td>
</tr>
<tr>
<td>Hua [28]</td>
<td>0</td>
<td>l²</td>
<td>l² + 4 − 3l + l(1)</td>
<td>0</td>
<td>R₂</td>
<td>6l[m/l²]</td>
<td>η₂</td>
</tr>
<tr>
<td>Chen [29]</td>
<td>0</td>
<td>l² + l</td>
<td>l² + 2l</td>
<td>2(2)</td>
<td>R₃</td>
<td>D₁</td>
<td>η₃</td>
</tr>
<tr>
<td>Proposed</td>
<td>2</td>
<td>3l + 2</td>
<td>3l</td>
<td>2l</td>
<td>4l[m/l + 1] + 1</td>
<td>m[l/m]</td>
<td>η₄</td>
</tr>
</tbody>
</table>

(1) The 3-input logic XOR area is estimated as 1.5× the area of the 2-input logic XOR.
(2) The switches in Multiplier of [29] have the same area as the 2-to-1 MUX as it has the same number of transistors.

Table 2: Performance parameters of the adopted word-serial multipliers for n = 409 and different values of l.

<table>
<thead>
<tr>
<th>Multiplier</th>
<th>l</th>
<th>A (Kgates)</th>
<th>CPD (ps)</th>
<th>T (ns)</th>
<th>P (nW)</th>
<th>E (V)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Xie [27]</td>
<td>8</td>
<td>324.0</td>
<td>92.98</td>
<td>51.0</td>
<td>16.50</td>
<td>223.60</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>172.0</td>
<td>97.1</td>
<td>170.0</td>
<td>50.90</td>
<td>477.4</td>
</tr>
<tr>
<td></td>
<td>32</td>
<td>98.0</td>
<td>195.13</td>
<td>170.0</td>
<td>50.90</td>
<td>477.4</td>
</tr>
<tr>
<td>Pan [13]</td>
<td>8</td>
<td>48.0</td>
<td>97.5</td>
<td>206.3</td>
<td>10.0</td>
<td>255.0</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>26.0</td>
<td>124.6</td>
<td>244.4</td>
<td>8.8</td>
<td>320.1</td>
</tr>
<tr>
<td></td>
<td>32</td>
<td>14.0</td>
<td>164.3</td>
<td>282.5</td>
<td>6.9</td>
<td>425.1</td>
</tr>
<tr>
<td>Hua [28]</td>
<td>8</td>
<td>293.84</td>
<td>8.0</td>
<td>73.4</td>
<td>10053.4</td>
<td>4.4</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>129792.0</td>
<td>4.0</td>
<td>73.4</td>
<td>9536.7</td>
<td>5.9</td>
</tr>
<tr>
<td></td>
<td>32</td>
<td>64869.0</td>
<td>19.9</td>
<td>73.4</td>
<td>4763.4</td>
<td>11.2</td>
</tr>
<tr>
<td>Chen [29]</td>
<td>8</td>
<td>11546.0</td>
<td>10.2</td>
<td>55.2</td>
<td>659.4</td>
<td>5.1</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>3678.0</td>
<td>13.5</td>
<td>55.2</td>
<td>203.0</td>
<td>8.4</td>
</tr>
<tr>
<td></td>
<td>32</td>
<td>1572.0</td>
<td>26.6</td>
<td>55.2</td>
<td>86.8</td>
<td>16.0</td>
</tr>
<tr>
<td>Proposed</td>
<td>8</td>
<td>21228.0</td>
<td>4.4</td>
<td>47.7</td>
<td>1014.5</td>
<td>1.9</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>10634.0</td>
<td>4.7</td>
<td>47.7</td>
<td>507.2</td>
<td>3.1</td>
</tr>
<tr>
<td></td>
<td>32</td>
<td>537.0</td>
<td>8.5</td>
<td>47.7</td>
<td>253.6</td>
<td>4.1</td>
</tr>
</tbody>
</table>

Figure 8: Area experimental results.

l = 32 compared with the adopted word-serial multipliers. As we mentioned before, the saving in the area is due to the lower area complexity O(l) of the proposed design compared to the other designs. It is worth noticing that the area of the proposed design has a slight difference at the different word sizes. This is due to the reverse relationship between the number of Flip-Flops and the word size l as indicated in Table 1. Therefore, as l increase, the number of Flip-Flops decrease and the number of basic logic components significantly increase as it is directly proportional to the word-size l. Thus, the net result is a slight increase in the proposed design area as its word size increases.

Figure 9 displays the obtained computation time results of the proposed design and the adopted word-serial ones. We can read the results based on the different word-sizes l as follows:

i) At word-size l = 8, the multiplier that achieves the lowest computation time is the multiplier of Pan [13].

ii) At word-sizes l = 16 and l = 32, the multiplier that achieves the lowest computation time is the multiplier of Xie [27].

Figure 10 shows that proposed multiplier structure achieves a significant reduction in power consumption ranging from 74.25% to 99.56% at l = 8, 64.10% to 99.44% at l = 16, and 73.99% to 99.39% at l = 32 compared to the other multipliers.

5. Summary and Conclusion

This paper proposed a word-serial accelerator multiplier structure that performs multiplication in GF(2ᵐ). The main feature of the proposed multiplier is its flexibility to manage the accelerator workload and the required total computation time steps to produce the output results. The experimental results designs. The reduction of power is attributed to the lower area complexity of the proposed design over the other designs.

Figure 11 shows that the proposed multiplier structure achieves a magnified reduction in energy ranging from 61.20% to 98.83% at l = 8, 67.73% to 98.26% at l = 16, and 76.13% to 98.81% at l = 32 compared to the adopted word-serial multipliers. The energy reduction is attributed to the magnified reduction of the consumed power of the proposed design over the adopted ones.

From the previous analysis, we can conclude that the recommended word-serial multiplier structure outperforms the other competitor multiplier structures in terms of area and consumed energy for the different embedded word sizes. This indicates that the proposed multiplier is suitable for IoT devices in tele-health applications and other resource-constrained IoT applications.
confirm that the proposed multiplier outperforms the efficient word-serial ones previously reported in the literature in terms of area and consumed energy for various embedded word sizes making it more suitable for embedded telehealth applications and other resource-constrained IoT applications.

Acknowledgements

The authors would like to acknowledge the financial support of the National Research Council of Canada (NRC) grant under the Collaborative R&D Initiative.

References


Compact Word-Serial Modular Multiplier Accelerator Structure for Cryptographic Processors in Telehealth IoT Network
Atef Ibrahim\textsuperscript{1,2}, Fayez Gebali\textsuperscript{2}

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

**Corresponding Author:**
Atef Ibrahim, PhD
Computer Engineering Department
College of Computer Engineering and Sciences
Prince Sattam Bin Abdulaziz University
11942 Alkharj
Saudi Arabia
Tel: 00966546825905
Declaration of interests

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.