Digit-Level Serial-In Parallel-Out Multiplier Using Redundant Representation for a Class of Finite Fields

Parham Hosseinzadeh Namin  
*University of Windsor*

Roberto Muscedere  
*University of Windsor*

Majid Ahmadi  
*University of Windsor*

Follow this and additional works at: [https://scholar.uwindsor.ca/electricalengpub](https://scholar.uwindsor.ca/electricalengpub)  
Part of the [Electrical and Computer Engineering Commons](https://scholar.uwindsor.ca/electricalengpub)

Recommended Citation  
[https://scholar.uwindsor.ca/electricalengpub/10](https://scholar.uwindsor.ca/electricalengpub/10)
Digit-Level Serial-In Parallel-Out Multiplier Using Redundant Representation for a Class of Finite Fields

Parham Hosseinzadeh Namin, Student Member, IEEE, Roberto Muscedere, Member, IEEE, and Majid Ahmadi, Life Fellow, IEEE

Abstract—Two digit-level finite field multipliers in $\mathbb{F}_{2^n}$ using redundant representation are presented. Embedding $\mathbb{F}_{2^n}$ in cyclotomic field $\mathbb{F}_{2^{m(n)}}$ causes a certain amount of redundancy and consequently performing field multiplication using redundant representation would require more hardware resources. Based on a specific feature of redundant representation in a class of finite fields, two new multiplication algorithms along with their pertaining architectures are proposed to alleviate this problem. Considering area-delay product as a measure of evaluation, it has been shown that both the proposed architectures considerably outperform existing digit-level multipliers using the same basis. It is also shown that for a subset of the fields, the proposed multipliers are of higher performance in terms of area-delay complexities among several recently proposed optimal normal basis multipliers. The main characteristics of the postplace&route application specific integrated circuit implementation of the proposed multipliers for three practical digit sizes are also reported.

Index Terms—Digit-level architecture, finite field arithmetic, multiplication algorithm, redundant representation.

I. INTRODUCTION

FINITE field computation has recently gained growing attention due to its wide range of applications in coding theory, error control coding, and especially in cryptography, where ElGamal [1] and elliptic curve cryptography (ECC) [2], two out of the three well-known cryptosystems, are based on finite field arithmetic [3]. Finite field computation is performed using arithmetic operations in the underlying finite field. Among the basic field operations, multiplication plays a fundamental role as more complicated operations, namely, field exponentiation and field inversion can be carried out with consequent use of field multiplication [2, 4, 5].

Similar to linear algebra, the concept of representation bases is also used in finite field arithmetic to represent field elements. The choice of representation system—mainly affected by the hardware in use and the requirements of the cryptosystem, has a great impact on computational performance. A few number of representation systems for extension binary fields have been proposed in the literature, such as polynomial basis [6], normal basis (NB) [7], redundant basis (RB) [8], and dual basis [9]. In both NB and redundant representation, squaring operation can be performed by applying a simple permutation operation on the coordinates. This makes them more efficient for the hardware implementations of cryptographic algorithms that utilize frequent squaring or exponentiation, such as point addition/doubling in ECC. Moreover, redundant representation is of a special interest due to its unique feature in accommodating ring type operations. This not only offers almost cost-free squaring operation but also eliminates the need for modular reduction in multiplication.

The idea of embedding a field in a larger ring was first put forward by Gao et al. [10], [11] for performing fast multiplication using NB. Later on, Wu et al. [8] introduced redundant representation, also known as RB, and finite field multiplication using this representation system. In efforts to increase the multiplication speed or to reduce the hardware complexities, several architectures have been proposed afterward, such as comb-style architecture [12] and linear feedback shift register (LFSR)-based architectures [13], [14]. More recently, Xie et al. [15] proposed a recursive decomposition scheme for digit-level serial/parallel structures to achieve less area–time–power complexities.

Despite the structure of the architecture in use, the main drawback of redundant representation is that it contains a certain amount of redundancy as embedding field $\mathbb{F}_{2^n}$ of size $m$ in cyclotomic field $\mathbb{F}_{2^{m(n)}}$ of size $n$, ($n > m$), is not a one-to-one mapping operation. As a result, redundant representation requires more bits to represent a field element, where the number of representation bits depends on the size of the cyclotomic field in which the underlying field is embedded.

In this paper, our focus is on digit-level architectures for RB multipliers. We show that a specific feature of redundant representation can be used for a class of finite fields to significantly reduce the architectural complexity of RB multipliers to compensate for the inherent redundancy in this representation system. Two variants of multiplication algorithms along with their corresponding architecture are presented. It is shown that the proposed architectures have highly regular structures and thus suitable for hardware implementation. Comparisons with existing digit-level RB architectures reveal that both the proposed architectures outperform other RB architectures when considering area-delay product as a measure of performance. A comparison between the performances of the proposed multipliers with those of several optimal

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.
NB (ONB) multipliers is also given. Finally, hardware realizations of the proposed multipliers for three practical digit sizes are presented.

The organization of this paper is as follows. Section II contains a brief review of RB and finite field multiplication using this representation system. In Section III, two new digit-level algorithm and architectures for RB multiplication are presented. The architectural complexity and the performance comparison are discussed in Section IV followed by the details of VLSI implementations of three practical field size multipliers in Section V. The conclusion remarks are given in Section VI.

II. BRIEF REVIEW OF REDUNDANT BASIS REPRESENTATION IN \( \mathbb{F}_{2^m} \)

A. Redundant Representation for \( \mathbb{F}_{2^m} \)

Let \( \mathbb{F}_2 \) denote a field of characteristic 2 and \( x^n - 1 \) be a polynomial of degree positive integer \( n \) over \( \mathbb{F}_2 \). Then, the splitting field of \( x^n - 1 \), denoted by \( \mathbb{F}_{2^n} \), is called the \( n \)th cyclotomic field over \( \mathbb{F}_2 \). Let \( \beta \) be a primitive \( n \)th root of unity in an extension field of \( \mathbb{F}_2 \). Then, \( \mathbb{F}_{2^n} \) is generated by \( \beta \) over \( \mathbb{F}_2 \) and elements of \( \mathbb{F}_{2^n} \) can be represented in the form of

\[
A = a_0 + a_1 \beta + a_2 \beta^2 + \cdots + a_{n-1} \beta^{n-1}, \quad a_i \in \mathbb{F}_2. \tag{1}
\]

This representation of \( A \) is not unique, i.e., for a given element of \( \mathbb{F}_{2^n} \) represented by \( n \)-tuple \( (a_0, a_1, \cdots, a_{n-1}) \), \( a_i \in \mathbb{F}_2 \), there exist different tuples representing the same element. For instance, each element in \( \mathbb{F}_{2^n} \) and its ones’ complement represent the same field element as explained in Lemma 1.

\begin{lemma}
Assume that field element \( E \in \mathbb{F}_{2^n} \) is represented by \( (e_0, e_1, \ldots, e_{n-1}) \), \( e_i \in \mathbb{F}_2 \) with respect to \( I = \{1, \beta, \beta^2, \ldots, \beta^{n-1}\} \). Then

\[
E = e_0 + e_1 \beta + \cdots + e_{n-1} \beta^{n-1} = (1 + e_0) + (1 + e_1) \beta + \cdots + (1 + e_{n-1}) \beta^{n-1}. \tag{2}
\]

\end{lemma}

\begin{proof}
Since the set of powers of primitive \( n \)th root of unity, i.e., \( \{\beta^i, 0 \leq i \leq n - 1\} \), form a cyclic group of order \( n \), then, \( \beta^n = 1 \) and \( 1 + \beta + \beta^2 + \cdots + \beta^{n-1} = 0 \) accordingly.

An interesting example would be the identity element of the field with respect to operation “+,” namely, “0,” which can be represented by both \( n \)-tuples \( (0, 0, \ldots, 0) \) and \( (1, 1, \cdots, 1) \). Due to the redundancy in this representation system, \( (1) \) is called a redundant representation of \( A \) and \( I = \{1, \beta, \beta^2, \ldots, \beta^{n-1}\} \) is referred to as a RB for any subfield of \( \mathbb{F}_{2^n} \) [8].

In order for a field of characteristic two, \( \mathbb{F}_{2^m} \), to be embedded in \( \mathbb{F}_{2^n} \), the following relationship between \( n \) and \( m \) should be satisfied.

\begin{theorem}
Let \( n \) be an odd positive integer greater than \( m \). Then, \( \mathbb{F}_{2^m} \) is contained in \( \mathbb{F}_{2^n} \) iff \( m \) divides the multiplicative order of 2 modulo \( n \) [16, Th. 2.47].

For more information about conversion from/to normal representation system to/from redundant representation, the reader is referred to [8] and [17].

B. Multiplication Using Redundant Representation in \( \mathbb{F}_{2^m} \)

One of the unique advantages of using RB in finite field arithmetic is that it eliminates the need for modular reduction in multiplication operation. This useful feature stems from the fact that the basis elements \( 1, \beta, \beta^2, \ldots, \beta^{n-1} \) form a cyclic group of order \( n \). As a direct result

\[
\beta \cdot \beta^i = \begin{cases} 
\beta^{i+1}, & i \neq n - 1 \\
1, & i = n - 1. 
\end{cases} \tag{3}
\]

Let field elements \( A \) and \( B \in \mathbb{F}_{2^m} \) be expressed with respect to the RB \( I = \{1, \beta, \beta^2, \ldots, \beta^{n-1}\} \) as

\[
A = \sum_{i=0}^{n-1} a_i \beta^i, \quad B = \sum_{i=0}^{n-1} b_i \beta^i
\]

respectively, where \( a_i, b_i \in \mathbb{F}_2 \). Note that \( n \geq m + 1 \) and \( \beta^n \neq 0 \). Then \( C \), the product of \( A \) and \( B \), can be given by

\[
C = A \cdot B = \sum_{i=0}^{n-1} (a_i \beta^i) \cdot B = \sum_{i=0}^{n-1} a_i \left( \sum_{j=0}^{n-1} b_j \beta^{i+j} \right)
\]

\[
= \sum_{i=0}^{n-1} a_i \left( \sum_{j=0}^{n-1} b_{(j-i)n} \beta^{i} \right)
\]

\[
= \sum_{j=0}^{n-1} \left( \sum_{i=0}^{n-1} a_i b_{(j-i)n} \right) \beta^{j}. \tag{4}
\]

where \( (j-i)_n \) denotes that \( (j-i) \) is to be reduced modulo \( n \).

Define, \( A \cdot B = C \equiv \sum_{j=0}^{n-1} c_j \beta^{j} \), then, \( c_j \) can be given by

\[
c_j = \sum_{i=0}^{n-1} a_i b_{(j-i)n}, \quad j = 0, 1, \ldots, n - 1. \tag{5}
\]

III. PROPOSED DIGIT-LEVEL SIPO MULTIPLIERS USING REDUNDANT REPRESENTATION

In this section, we first present a new algorithm for RB multiplication. Based on this algorithm, we propose two new optimized digit-level serial-in-parallel-out (SIPO) architectures. These architectures are adopted for a class of finite fields in which \( n \) can be expressed as \( n = T m + 1 \), where \( T \geq 2 \) and \( m \) is an even number. As will be seen in the remainder of this section, this condition enables us to devise an architecture that significantly reduces the complexity of the multiplier. Corollary 1, which corresponds to [8, Lemma 2], describes a specific feature that results from the above-mentioned condition.

\begin{corollary}
Let \( A \in \mathbb{F}_{2^m} \) and assume that its redundant representation is given by \( (a_0, a_1, \ldots, a_{n-1}) \) with respect to RB \( I \) over \( \mathbb{F}_{2^m} \). If \( n \) can be expressed as \( T m + 1 \), assuming that \( T \geq 2 \) and is even, then

\[
a_k = a_{n-k}, \quad k = 1, 2, \ldots, n - 1. \tag{6}
\]

The degree of the smallest cyclotomic field that contains \( \mathbb{F}_{2^m} \) can be calculated using [18, Algorithm 1]. For ECC applications, field sizes are recommended to be
selected within the range of 150 to 600 according to the security standards [19]. It can be shown, via [18], that Corollary 1 covers over 60% of all finite fields within the practical range. As an example, for the first 100 fields in the aforementioned range, the orders of the smallest cyclotomic fields that contain $\mathbb{F}_{2^m}$ for a subset of fields that the relationship between $n$ and $m$ satisfies the requirement of Corollary 1 that are listed in Table I.

### A. Proposed Digit-Level RB Multiplication Algorithm, Type-a

Assume $w$, $1 \leq w \leq (n - 1/2)$, denotes the digit size of the multiplier. Excluding $a_0$ from the coordinate set let the rest of the coordinates be equally divided into $2w$ parts, $d$-bit long each, where $d = \lceil (n - 1/2w) \rceil$

$$A = a_0 \overbrace{a_1 \ldots a_{1d+1} \ldots a_{2d} \ldots}^{A_0} \ldots \overbrace{a_{(2w-1)d+1} \ldots a_{n-1} \ldots 0 \ldots 0}^{A_{2n-1}}.$$  

Note that the outside of the coordinate set is padded with zero. Replace subscript $i$ of $a_i$ in (5) with $kd + \ell$ for $0 \leq k \leq 2w - 1$ and $0 \leq \ell \leq d$. The product coefficient $c_j$, $j = 0, 1, \ldots, n - 1$, can be rewritten as

$$c_j = a_0 b_j + \sum_{k=0}^{2w-1} \sum_{\ell=1}^{d} a_{(kd+\ell)} b_{(j-kd-\ell)}. \quad (7)$$

Based on the definition of $d$, we have: $(n - 1/2w) < d \leq (n - 1/2w) + 1$. As a result, the upper bound of the subscript $kd + \ell$ in the above-mentioned double summation falls within the range of $n - 1$ to $n - 1 + 2w$, and thus, all the nonzero terms of the product coefficient $c_j$ is included in (7).

Under the required conditions of Corollary 1, the last $(Tm/2)$ coordinates of a field element are mirror reflections of the first $(Tm/2)$ coordinates. A new function $\varphi(i)$ can be utilized to map the set of integers used in the subscript of the coordinates to the set $\{0, 1, \ldots, (n - 1/2)\}$, as follows:

$$\varphi(i) = \begin{cases} i \mod n & 0 \leq i \mod n \leq \frac{n-1}{2} \\ n -(i \mod n) & \text{Otherwise}. \end{cases} \quad (8)$$

Now assume that $\hat{a}_i$, $0 \leq i \leq (n - 1/2)$ denotes the first $(n + 1/2)$ coordinates of $A$, as

$$\hat{a}_i \begin{cases} a_i & 0 \leq i \leq \frac{n-1}{2} \\ 0 & \text{Otherwise}. \end{cases} \quad (9)$$

Taking into account (8) and (9), (7) can be rewritten as

$$c_j = a_0 b_j + \sum_{k=0}^{w-1-d} \sum_{\ell=1}^{d} a_{(kd+\ell)} b_{\varphi(j-kd-\ell)}$$

$$+ \sum_{k=0}^{w-1} \sum_{\ell=\varphi^{-1}\left(\frac{n-1}{2}+1\right)}^{d} a_{(kd+\ell)} b_{\varphi(j-kd-\ell)} \quad (10)$$

$$= a_0 b_j + \sum_{k=0}^{w-1} \sum_{\ell=1}^{d} \hat{a}_{(kd+\ell)} b_{\varphi(j-kd-\ell)}$$

$$+ \sum_{k=w-1}^{0} \sum_{\ell=d}^{\varphi^{-1}\left(\frac{n-1}{2}\right)} \hat{a}_{(kd+\ell)} b_{\varphi(j-kd-\ell)}$$

$$= a_0 b_j + \sum_{k=0}^{w-1} \sum_{\ell=1}^{d} \hat{a}_{(kd+\ell)} [b_{\varphi(j-kd-\ell)} + b_{\varphi(j+k\ell+\ell)}] \quad (11)$$

for $j = 0, 1, \ldots, n - 1$. In the last term of (11), $a$ is replaced with $\hat{a}$ due to the fact that coordinates $a_i$ for $i = (n + 1/2)$ to $n - 1$ are equal to their corresponding $\hat{a}_i$ for $i = (n - 1/2)$ to 1. Also, from (8), it is clear that $b_{\varphi(j-(n+1/2))} = b_{\varphi(j+(n-1/2))}$ and $b_{\varphi(j-(n-1/2))} = b_{j+1}$.

The complexity of multiplication operation carried out using (12) can be further reduced by utilizing Lemma 1. Taking into consideration (2), term $a_0 b_j$ can be removed from (12). If coordinate $a_0$ is equal to zero then the original representation of $A$ is used without being changed. Otherwise, in the case $a_0 = 1$, $A$ complement can be used instead of $A$ without having any effect on the multiplication operation. As a result, the product coefficient $c_j$ can be obtained as

$$c_j = \sum_{k=0}^{w-1-d} \sum_{\ell=1}^{d} a_{(kd+\ell)} [b_{\varphi(j-kd-\ell)} + b_{\varphi(j+k\ell+\ell)}] \quad (13)$$

Lemma 2 shows that if $a_0 = 0$, and then, $c_0$ will be equal to zero too.

### Lemma 2

Let field elements $A$, $B$, and $C \in \mathbb{F}_{2^m}$ be expressed with respect to RB $I = \{1, \beta, \beta^2, \ldots, \beta^{n-1}\}$ as $A = \sum_{i=0}^{n-1} a_i \beta^i$, $B = \sum_{i=0}^{n-1} b_i \beta^i$, and $C = \sum_{i=0}^{n-1} c_i \beta^i$, respectively. Assume $n$ is expressed as $Tm + 1$, where $T \geq 2$ and is an even integer. Then, if $a_0 = 0$ and $c_0 = 0$. 

### Table I

<table>
<thead>
<tr>
<th>$m$</th>
<th>$n$</th>
<th>$T$</th>
<th>$m$</th>
<th>$n$</th>
<th>$T$</th>
<th>$m$</th>
<th>$n$</th>
<th>$T$</th>
</tr>
</thead>
<tbody>
<tr>
<td>151</td>
<td>907</td>
<td>6</td>
<td>186</td>
<td>373</td>
<td>2</td>
<td>219</td>
<td>877</td>
<td>4</td>
</tr>
<tr>
<td>152</td>
<td>1217</td>
<td>8</td>
<td>187</td>
<td>1123</td>
<td>6</td>
<td>221</td>
<td>443</td>
<td>2</td>
</tr>
<tr>
<td>153</td>
<td>613</td>
<td>4</td>
<td>189</td>
<td>379</td>
<td>2</td>
<td>223</td>
<td>2677</td>
<td>12</td>
</tr>
<tr>
<td>154</td>
<td>617</td>
<td>4</td>
<td>191</td>
<td>383</td>
<td>2</td>
<td>224</td>
<td>449</td>
<td>2</td>
</tr>
<tr>
<td>155</td>
<td>311</td>
<td>2</td>
<td>192</td>
<td>769</td>
<td>4</td>
<td>227</td>
<td>5449</td>
<td>24</td>
</tr>
<tr>
<td>157</td>
<td>1571</td>
<td>10</td>
<td>193</td>
<td>773</td>
<td>4</td>
<td>229</td>
<td>2749</td>
<td>12</td>
</tr>
<tr>
<td>158</td>
<td>317</td>
<td>2</td>
<td>194</td>
<td>389</td>
<td>2</td>
<td>230</td>
<td>461</td>
<td>2</td>
</tr>
<tr>
<td>161</td>
<td>967</td>
<td>6</td>
<td>197</td>
<td>3547</td>
<td>18</td>
<td>231</td>
<td>463</td>
<td>2</td>
</tr>
<tr>
<td>163</td>
<td>635</td>
<td>4</td>
<td>199</td>
<td>797</td>
<td>4</td>
<td>232</td>
<td>929</td>
<td>4</td>
</tr>
<tr>
<td>165</td>
<td>661</td>
<td>4</td>
<td>200</td>
<td>401</td>
<td>2</td>
<td>233</td>
<td>467</td>
<td>2</td>
</tr>
<tr>
<td>167</td>
<td>2339</td>
<td>14</td>
<td>201</td>
<td>1609</td>
<td>8</td>
<td>235</td>
<td>941</td>
<td>4</td>
</tr>
<tr>
<td>169</td>
<td>677</td>
<td>4</td>
<td>202</td>
<td>809</td>
<td>4</td>
<td>237</td>
<td>1423</td>
<td>6</td>
</tr>
<tr>
<td>170</td>
<td>1021</td>
<td>6</td>
<td>204</td>
<td>409</td>
<td>2</td>
<td>239</td>
<td>479</td>
<td>2</td>
</tr>
<tr>
<td>173</td>
<td>347</td>
<td>2</td>
<td>205</td>
<td>821</td>
<td>4</td>
<td>241</td>
<td>1447</td>
<td>6</td>
</tr>
<tr>
<td>174</td>
<td>349</td>
<td>2</td>
<td>207</td>
<td>829</td>
<td>4</td>
<td>243</td>
<td>487</td>
<td>2</td>
</tr>
<tr>
<td>175</td>
<td>701</td>
<td>4</td>
<td>208</td>
<td>2081</td>
<td>10</td>
<td>245</td>
<td>491</td>
<td>2</td>
</tr>
<tr>
<td>176</td>
<td>1409</td>
<td>8</td>
<td>209</td>
<td>419</td>
<td>2</td>
<td>247</td>
<td>1483</td>
<td>6</td>
</tr>
<tr>
<td>177</td>
<td>709</td>
<td>4</td>
<td>211</td>
<td>2111</td>
<td>10</td>
<td>248</td>
<td>1489</td>
<td>6</td>
</tr>
<tr>
<td>179</td>
<td>559</td>
<td>2</td>
<td>213</td>
<td>853</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>181</td>
<td>1087</td>
<td>6</td>
<td>215</td>
<td>1291</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>183</td>
<td>367</td>
<td>2</td>
<td>216</td>
<td>1297</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>185</td>
<td>1481</td>
<td>8</td>
<td>217</td>
<td>1303</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

* The information presented in the table above is extracted from a more inclusive table in [8] and can be calculated using the algorithm presented in [18].
Proof: Substituting zero for \( j \) in (13) and according to the definition of function \( \varphi(i) \) in (8) we have
\[
c_0 = \sum_{k=0}^{w-1} \sum_{\ell=1}^{d} \hat{a}_{(kd+\ell)} [b_{\varphi(-kd-\ell)} + b_{\varphi(kd+\ell)}]
\]
\[
= \sum_{k=0}^{w-1} \sum_{\ell=1}^{d} \hat{a}_{(kd+\ell)} [b_{\varphi(kd+\ell)} + b_{\varphi(kd+\ell)}] = 0.
\]

Using (13), it can be easily proven that \( c_{n-j} = c_j \) for \( j = 1, 2, \ldots, n-1 \)
\[
c_{n-j} = \sum_{k=0}^{w-1} \sum_{\ell=1}^{d} \hat{a}_{(kd+\ell)} b_{\varphi(-j-kd-\ell)} + b_{\varphi(-j-kd+\ell)}
\]
\[
= \sum_{k=0}^{w-1} \sum_{\ell=1}^{d} \hat{a}_{(kd+\ell)} b_{\varphi(j-kd-\ell)} + b_{\varphi(j-kd+\ell)}
\]
The definition of function \( \varphi(i) \) in (8) implies that \( \varphi(n-i) = \varphi(i) \), hence
\[
c_{n-j} = \sum_{k=0}^{w-1} \sum_{\ell=1}^{d} \hat{a}_{(kd+\ell)} b_{\varphi(j-kd+\ell)} + b_{\varphi(j-kd-\ell)}
\]
\[
= c_j.
\]

In order to obtain a digit-level multiplication algorithm, first decompose (13) into two double summations as
\[
c_j = \sum_{k=0}^{w-1} \sum_{\ell=1}^{d} \hat{a}_{(kd+\ell)} b_{\varphi(\ell)} + \sum_{k=0}^{w-1} \sum_{\ell=1}^{d} \hat{a}_{(kd+\ell)} b_{\varphi(j-kd+\ell)}.
\]
Then, define two signals \( p_{j,k}^{(\ell)} \) and \( q_{j,k}^{(\ell)} \) as follows:
\[
\begin{align*}
p_{j,k}^{(\ell)} &= 0 \quad \text{and} \quad p_{j,k}^{(\ell-1)} = \hat{a}_{(kd+\ell)} b_{\varphi(j-kd-\ell)}
p_{j,k}^{(0)} &= 0 \quad \text{and} \quad p_{j,k}^{(0)} = \hat{a}_{(kd+\ell)} b_{\varphi(j-kd+\ell)}
\end{align*}
\]
where \( \ell = 1, 2, \ldots, d \) indicates the current clock cycle. \( p_{j,k}^{(\ell)} \) and \( q_{j,k}^{(\ell)} \) hold the sum of inner products of certain coordinates of A and B [denoted by \( P \) and \( Q \) in (14)] up to the \( \ell \)th clock cycle. After \( d \) clock cycles, the values of signals \( p_{j,k}^{(d)} \) and \( q_{j,k}^{(d)} \) would be equal to
\[
\begin{align*}
p_{j,k}^{(d)} &= \sum_{\ell=1}^{d} \hat{a}_{(kd+\ell)} b_{\varphi(\ell)}
q_{j,k}^{(d)} &= \sum_{\ell=1}^{d} \hat{a}_{(kd+\ell)} b_{\varphi(j-kd+\ell)}
\end{align*}
\]
comparing (16) with (13), it follows:
\[
c_j = \sum_{k=0}^{w-1} \left[ p_{j,k}^{(d)} + q_{j,k}^{(d)} \right]
\]

If the values of \( p_{j,k}^{(\ell)} \) and \( q_{j,k}^{(\ell)} \) can be calculated and accumulated for all the values of \( j \) and \( k \) at each clock cycle, then it takes \( d = \lceil (n-1)/2w \rceil \) clock cycles to obtain all the product coefficients. Algorithm 1 describes the multiplication process in more detail. To perform arithmetic operations in binary field \( \mathbb{F}_2 \), one XOR gate and one AND gate are required to realize a bitwise addition and a bitwise multiplication, respectively. In Step 5 of the algorithm, for given \( j \) and \( k \), an AND gate is used to multiply one bit of each input operands together, and then, an XOR gate is required to perform addition operation. Step 6 could also be implemented using one XOR and one AND gate in a similar way to Step 5.

A pair of flip-flops is also required for given \( j \) and \( k \) to store the values of two signals \( p_{j,k}^{(\ell)} \) and \( q_{j,k}^{(\ell)} \) after each clock cycle. Note that Steps 5 and 6 of the multiplication algorithm are computed in parallel at each clock cycle, while the resulting values are accumulated throughout \( \ell \) clock cycles in serial. Finally, the accumulation in Step 12 is computed right after the \( d \)th clock cycle. For a given \( j \), \( w \) pairs of intermediate signals \( p_{j,k}^{(d)} \) and \( q_{j,k}^{(d)} \) (for \( k \) from 0 to \( w-1 \)) are to be added together to form the final value of the corresponding product coordinate.

### Algorithm 1 Digit-Level RB Multiplication Algorithm

**Where** \( n \) can Be Expressed as \( n = Tm + 1 \), \( T \geq 2 \) and Even

**INPUT:** \( A = (a_1, \ldots, a_{n-1}), B = (b_0, \ldots, b_{n-1}) \), w.r.t. RB

**OUTPUT:** \( C = A \times B = (c_0, \ldots, c_{n-1}) \) w.r.t. RB

1. **Initialization:** \( p_{j,k}^{(0)} = 0 \), \( q_{j,k}^{(0)} = 0 \), for \( j = 1, 2, \ldots, n-1 \) and \( k = 0, 1, \ldots, w-1 \)
2. **for all** values of \( j = 1, 2, \ldots, n-1 \), compute in parallel
3. **for all** values of \( k = 0, 1, \ldots, w-1 \), compute in parallel
4. **for** \( \ell = 1 \) to \( d \) compute in serial
5. \( p_{j,k}^{(\ell)} = p_{j,k}^{(\ell-1)} + \hat{a}_{(kd+\ell)} b_{\varphi(j-kd-\ell)} \)
6. \( q_{j,k}^{(\ell)} = q_{j,k}^{(\ell-1)} + \hat{a}_{(kd+\ell)} b_{\varphi(j-kd+\ell)} \)

7. **end for**
8. **end for**
9. **end for**
10. **for all** values of \( j = 1, 2, \ldots, n-1 \), compute in parallel
11. **for all** values of \( k = 0, 1, \ldots, w-1 \), compute in serial
12. \( c_j = \sum_{k=0}^{w-1} [p_{j,k}^{(d)} + q_{j,k}^{(d)}] \)

13. **end for**
14. **end for**

### B. New Multiplier Architecture, DL-SRB-a

An architecture for the proposed multiplier can be constructed based on the steps described in Algorithm 1 at \( \ell = 1 \). Fig. 1 shows the proposed architecture, hereafter referred to as digit-level symmetrical RB type-a multiplier (DL-SRB-a). From top to bottom, the architecture consists of an \( n \)-bit circular shift register which should be initialized with the coordinates of operand \( B \). This shift register provides inputs to a wire expansion module with \( n \) inputs and \( w(n-1) \) outputs followed by \( \lceil (n-1)/2w \rceil \) identical modules \( (M_1, M_2, \ldots, M_{(n-1)/2w}) \) shown inside the dashed boxes. At the bottom, there is a network of XOR gates adding \( 2w \) outputs of each module \( M_j \) together to form output coordinates.

Each module \( M_j \) is made of a layer of \( 2w \) AND gates and one XOR gate on the outputs of the wire expansion module as their
first input set. The second input set is received from certain bits of operand $A$ in a digit-serial fashion. Each AND gate is followed by an XOR gate connected immediately to a flip-flop. The output of the flip-flop is fed back to the XOR gate forming an accumulation unit together. Two AND gates along with their respective accumulation units form a structure responsible to realize the operations performed in Steps 5 and 6 of Algorithm 1. One of these structures is shown in the Fig. 1 inside a dotted block for $j = 0$ and $k = 0$. In total, the proposed architecture contains $w(n-1)/2$ such structures, each of which consists of two AND gates, two XOR gates, and two flip-flops to generate and store $p_{j,k}^{(ℓ)}$ and $q_{j,k}^{(ℓ)}$ in each clock cycle.

As mentioned earlier, input $A$ should be fed into the multiplier in a digit-serial fashion (comb style). According to (13), the multiplication operation is performed using $\hat{a}_i$ coefficients which are necessarily equal to the $(n-1)/2$ coordinates of $A$ starting from coordinate number 1 to $(n-1)/2$. We will refer to this set of coordinates of $A$ as $\hat{A}$. Let $\hat{A}$ be divided into $w$ parts of length $d$ in the same way we did earlier for $A$, as

$$\hat{A} = \hat{a}_1 \ldots \hat{a}_d \hat{a}_{d+1} \ldots \hat{a}_2d \ldots \hat{a}_{(w-1)d+1} \hat{a}_{(w-1)d+2} 0 \ldots 0.$$  

Note that $\hat{A}$ is padded with $wd - (n-1)/2$ zeros in the most significant word. In the first clock cycle, the first bits of every word, i.e., $a_1, a_{d+1}, \ldots, a_{(w-1)d+1}$ form an input set to the multiplier. In the second clock cycle, the inputs would be the set of second bits of every word, $a_2, a_{d+2}, \ldots, a_{(w-1)d+2}$, so on and so forth.

For given $j$ and $k$, in each clock cycle, the variable of function $\varphi$ in $b_{\varphi(j-kd-\ell)}$ decreases by one in Step 5. An $n$-bit circular shift register can be used, as shown in Fig. 2 by $R_1$, to generate the required coefficients in Step 5. This circular shift register should be initially loaded as, from left to right, $b_{n-1}, b_{n-2}, \ldots, b_0$. On the contrary, the variable of function $\varphi$ in $b_{\varphi(j+kd+\ell)}$ in Step 6 increases by one in each clock cycle. In this case, a similar circular shift register, namely, $R_2$, with the same initial contents but with the opposite shift direction should be utilized to produce the required coefficients.

**Lemma 3:** Assuming the required conditions of symmetry property explained in Corollary 1 are satisfied, only one circular shift register of length $n$ would suffice to facilitate both the operations of Steps 5 and 6.

Let the upper half of register $R$ be initialized with equivalent coordinates from the lower half of operand $B$ in the way shown in Fig. 2. Since $\varphi(i) = \varphi(n-i)$, an increase/decrease...
by one in the variable of function \( \varphi \) within the range of \((n + 1)/2 \) to \( n \) would be equal to a decrease/increase by one within the range of \((n - 1)/2 \) to 0. Also, since \( \varphi(-i) = \varphi(i) \) an increase/decrease by one in the variable of function \( \varphi \) within the range of \(-(n + 1)/2 \) to 0 would be equal to a decrease/increase by one within the range of 0 to \((n - 1)/2 \). Consequently, the lower half of register \( R \) is used when an initial decrease in the lower half or an initial increase in the upper half of \( B \) is required and the upper half of register \( R \) is used when an initial increase in the lower half or an initial decrease in the upper half of \( B \) is needed.

Take an example in which \( j = 5 \) and \( k = 0 \). At the first clock cycle, the value of function \( \varphi \) in Step 5 is equal to \( \varphi(5 - 1) = 4 \). It decreases by one in each clock cycle up to the fifth cycle and will increase by one at each cycle afterward: \( \varphi(5 - 2) = 3, \ldots, \varphi(5 - 5) = 0, \varphi(5 - 6) = \varphi(1) = 1 \). The similar function value in Step 6, is initially equal to \( \varphi(5 - 1) = 6 \) at the first clock cycle. It increases by one in each clock cycle up to the \((n - 1)/2 \)th cycle, and will decrease by one at each cycle afterward: \( \varphi(5 + 2) = 7, \ldots, \varphi(n + 1)/2 = (n - 1)/2, \varphi(n + 3)/2 = (n - 3)/2, \ldots \). As a result, \( R_4 \) and \( R_{n-6} \) should be used to produce \( p_{j,k}^{(e)} \) and \( q_{j,k}^{(e)} \), respectively.

As can be seen in Fig. 1, the number of AND gates exceeds the number of flip-flops in register \( R \). The role of wire expansion module with \( n \) inputs and \( w(n-1) \) outputs is to receive input bits from register \( R \) and to deliver them to AND gates as follows: for given \( j \) and \( k \), inputs to \( p_{j,k}^{(e)} \) and \( q_{j,k}^{(e)} \) should be connected to \( R_{p(j-kd-\ell)-1} \) and \( R_{q(j-kd+\ell)+1} \), respectively. It is evident that what wire expansion module does, is nothing but permuting and reordering the input bits and that it does not contain any logic gates.

Depending on the choices of \( n \) and \( w \), the complexity of the multiplier may be reduced one step further. Recall that \( d = \lceil (n - 1)/2w \rceil \), so \((n - 1)/2w \leq d < (n - 1)/2w + 1 \). If \( w \) is chosen such that \( d(w-1) > (n-1)/2 \), then both \( p_{j,w-1}^{(d)} \) and \( q_{j,w-1}^{(d)} \) in (16) become zero for all the values of \( j \), \( 1 \leq j \leq (n - 1)/2 \). The reason lies in the fact that under the above-mentioned condition (or in other words, the use of \( 2w \) instead of \( w \) in the denominator of \( d \)), subscript \( kd + \ell \) of \( \tilde{a} \) for \( w = 0 \) becomes greater than \((n - 1)/2 \) for all the values of \( \ell \) from 1 to \( d \). Consequently, the last pair of accumulation units along with their respective AND gates can be discarded from all modules \( M_j \). For example, when \( n = 233 \), \( n = 2m + 1 = 467 \) and \( w = 32 \), \( d = \lceil 466/64 \rceil = 8 \) and \((w-1)d = 248 \) which is greater than 233. Likewise, \((w-2)d = 240 \) and is still greater than 233. In this case, each module \( M_j \) only needs to contain \( w - 2 \) pairs of AND–XOR-flop units to generate \( p_{j,k}^{(e)} \) and \( q_{j,k}^{(e)} \), \( 0 \leq k \leq w - 3 \). In the rest of this paper, the notation \( \tilde{w} \) will be used in place of \( w \) to denote the number of the parallel branches required in each module \( M_j \). \( \tilde{w} \) can be defined as follows:

\[
\tilde{w} = \arg \max_{w \in \mathbb{N}} \left( (w - 1)d \right) \left( (w - 1)d < \frac{n - 1}{2} \right). \tag{18}
\]

A noteworthy feature of the proposed architecture is that the critical path of the multiplier is independent of the field size \( (m) \), the degree of the cyclotomic field \( (n) \), and the digit size \( (w) \). The length of critical path in terms of the number of logic gates used remains constant regardless of the number of flip-flops in register \( R \) and the values of \( j \) and \( k \). As the wire expansion module does not require logic cells, the critical path is composed of one AND gate and one XOR gate. Assuming \( T_A \) and \( T_X \) denotes the time delays required by a two-input AND gate and a two-input XOR gate, respectively, the critical path delay is equal to \( T_{cp} = T_A + T_X \). Note that the XOR network shown in Fig. 1 (bottom) is not part of the critical path of the multiplier as the summation in Step 12 of Algorithm 1 is only needed to be performed once at the end of the multiplication operation.

In other words, the proposed architecture can be viewed as a sequential circuit followed by a combinational circuit. In the sequential part (which contains the whole circuit excluding the XOR tree), partial products \( p_{j,k}^{(e)} \) and \( q_{j,k}^{(e)} \) are recursively generated and stored in the flip-flops at each clock cycle. Note that during the first \( d \) clock cycles, the output of XOR trees are not required to be stored as they do not play any role in the computations performed in the sequential circuit. However, the product coordinates will not be available immediately after \( d \) clock cycles. It takes another time delay of \( (\log_2 2w)T_X \) associated with the binary tree of \((2n - 1)\) two-input XOR gates (combinational circuit) before the product coordinates can be read from the output end. To avoid the combinational circuit from becoming the critical path, this step should be performed in multicycles. A common solution would be the use of intermediate flip-flops to break a long path into smaller pieces. However, the use of extra flip-flops can be avoided provided that the inputs of the combinational circuit are kept unchanged so that the combinational circuit has enough time to generate valid outputs. In the proposed architecture, this is done by padding each input sequence \( \hat{A}_0, \hat{A}_1, \ldots, \hat{A}_{w-1} \) with \( d_{ex} \) zeroes, where \( d_{ex} \) is the number of extra clock cycles needed after the operations of Steps 5 and 6 are done. \( d_{ex} \) can be calculated as

\[
d_{ex} = \left\lceil \frac{(\log_2 2w)T_X}{T_{clock}} \right\rceil \tag{19}
\]

where \( T_{clock} \) refers to the clock period. If the clock period is chosen to be equal to the critical path delay to achieve the maximum operation frequency, \( T_{clock} \) should be replaced with \( T_{cp} \) in (19). Finally, the total number of clock cycles needed for a single multiplication operation is equal to \( d + d_{ex} \).

C. New Multiplier Architecture, DL-SRB-b

At the expense of a slight increase in the critical path delay, the number of logic gates and flip-flops used in the architecture of Fig. 1 can be significantly reduced. Starting from the closed formula of (13), instead of the decomposition shown in (14), define two intermediate signals \( s_{j,k}^{(e)} \) and \( r_{j,k}^{(e)} \), \( j = 1, 2, \ldots, (n - 1)/2 \) and \( k = 0, 1, \ldots, w - 1 \) for \( \ell = 1, 2, \ldots, d \) as

\[
\begin{align*}
s_{j,k}^{(e)} &= b_p(j - kd - \ell) + b_p(j + kd + \ell) \\
r_{j,k}^{(e)} &= 0 \quad \text{and} \quad r_{j,k}^{(e)} = r_{j,k}^{(e-1)} + a_{kd+\ell} s_{j,k}^{(e)}.
\end{align*}
\tag{20}
\]
$r_{j,k}^{(d)}$ holds the value of signal $r$ after $d$ clock and is equal to

$$r_{j,k}^{(d)} = \sum_{\ell=1}^{d} \hat{a}_{(kd+f)} [b_{\phi(j-kd-f)} + b_{\phi(j+kd+f)}].$$

Comparing (20) with (13), product coordinates $c_j$ can be expressed as

$$c_j = \sum_{k=0}^{w-1} r_{j,k}^{(d)}.$$  

The new algorithm can be obtained by replacing Steps 5–12 in Algorithm 1 with the following steps:

5: $s_{j,k}^{(d)} = [b_{\phi(j-kd-f)} + b_{\phi(j+kd+f)}]$
6: $r_{j,k}^{(d)} = r_{j,k}^{(d-1)} + \hat{a}_{(kd+f)} s_{j,k}^{(d)}$
7: end for
8: end for
9: for all values of $j = 1, 2, \ldots, \frac{n-1}{2}$, compute in parallel
10: for all values of $k = 0, 1, \ldots, w - 1$, compute in serial
11: $c_j = \sum_{k=0}^{w-1} r_{j,k}^{(d)}$.
12: end for

Note that in each clock cycle, Steps 5 and 6 should be computed in serial. Fig. 3 shows the modified architecture referred to as DL-SRB-$b$. As can be seen from Fig. 3, the new architecture is similar to the previously proposed architecture, DL-SRB-$a$, in the sense that it utilizes the same wire expansion module and the same $n$-bit circular shift register to store operand $B$. Operand $A$ is also fed into the multiplier in the same way as earlier. The main difference between the two architectures originates from the difference between the two modules shown inside the dotted boxes in Figs. 1 and 3. In type-$a$ architecture, one bit of operand $B$ is multiplied by one bit of operand $A$, and the resulting partial product is stored separately in its respective accumulation unit. On the contrary, in type-$b$ architecture, two bits of operand $B$ are first added together before they enter the AND gate and be fed into the accumulation unit. As a result, the critical path delay of the new architecture changes from $T_A + T_X$ to $T_A + 2T_X$. In the recent architecture, the number of accumulation units and AND gates are reduced by half from $w(n-1)$ to $w(n-1/2)$ each. Since half of the addition operations are performed before the accumulation units, the size of the binary XOR tree is also reduced from $2^w - 1$ to $w - 1$.

Similar to DL-SRB-$a$, the multiplication delay of DL-SRB-$b$ is composed of two parts: $d$ and $d_{ex}$. The first part corresponds to Steps 5 and 6 of the algorithm caused by modules $M_j$ during $d$ clock cycles. The second part corresponds to the time delay of a $w$-input XOR gate or a binary tree of $(w-1)$ two-input XOR gates. Assuming that a binary tree of two-input XOR gates is used, the total number of clock cycles required to complete a single multiplication...
can be calculated as
\[ d + \left\lceil \frac{[\log_2 w] T_X}{T_{\text{clock}}} \right\rceil. \]  

(23)

IV. ARCHITECTURAL COMPLEXITIES AND COMPARISON

The area complexities of the proposed architectures can be readily calculated from Figs. 1 and 3. In the case of type-\(a\) structure shown in Fig. 1, the circular shift register contains \(n\) flip-flops to store the coordinates of operand \(B\). As described earlier, the structure utilizes \((n - 1)/2\) identical modules \(M_j\), \(j = 1, 2, \ldots, (n - 1)/2\), each of which employing \(2w\) flip-flops to store the values of signals \(p_{j,k}^{(t)}\) and \(d_{j,k}^{(t)}\) at each clock cycle for all the values of \(k\) from 0 to \(w - 1\). In total, the number of required flip-flops comes to \(w(n - 1)/2 + n\). There are also \(w(n - 1)\) two-input AND gates, each followed immediately by a two-input XOR gate. Assuming that the XOR network at the bottom of the Fig. 3 is only made of two-input XOR gates, the architecture requires \((4w - 1)(n - 1/2)\) XOR gates altogether.

In the case of type-\(b\) architecture, each module \(M_i\) only contains \(w\) parallel branches instead of \(2w\) in type-\(a\) counterpart. As a result, the number of AND gates and the total number of flip-flops decrease to \(w(n - 1)/2\) and \(w(n - 1)/2 + n\), respectively. XOR gates appear in two separate layers in the structure of Fig. 3. The first layer consists of \(w(n - 1)/2\) gates and the layer at the bottom most part of the structure requires \((w - 1)(n - 1)/2\). In sum, a total of \((2w - 1)(n - 1)/2\) XOR gates is used in the structure of type-\(b\) multiplier.

ONBs are the most efficient classes of Gaussian NBs (GNBs) [16], [24]. To achieve smaller area and time overheads when using NBs over binary extension fields, it is recommended to use a GNB with the least possible type. The least possible type for a GNB is equal to 2 and type-2 GNB is also known as type-II ONB. Since ONB is the most efficient class of NB, it should be interesting to have a complexity comparison between the proposed multipliers and several recently proposed ONB multipliers. Table II draws a comparison between the hardware complexities of the two proposed multipliers, those of existing digit-level RB multipliers and several ONB multipliers. The comparison has been made in terms of the number of logic cells used, critical path delay (\(T_{\text{cp}}\)), and multiplication delay. Among the architectures listed in the Table II, the three architectures presented in [13], [14], and [21] are based on a LFSR structure, whereas the others have non-LFSR structural designs. The architecture most comparable to that being proposed is the “comb-style” architecture presented in [12]. Although the overall structure of the two architectures might seem similar, there are two important differences between them. First, the comb-style architecture in [12] implements a general RB multiplier and does not utilize the symmetry property even if applicable.

Second, for each output coordinate in this architecture, the results of partial bitwise products are added together first, and then, the resultant value enters the accumulation unit. The addition operation is applied over \(w\) partial products together with the current data stored in the accumulation unit before updating the output flip-flop at each clock cycle.

As a result, the critical path contains the XOR chain, thus causing additional delay cost.

As can be seen in the Table II, the second proposed architecture, DL-SRB-\(b\), requires the smallest number of gate counts compared with the other RB multipliers. In terms of maximum operating frequency, PS-III has the smallest critical path delay. However, the reduction in critical path delay is achieved by utilizing a layer of flip-flops between the AND gates and the pipelined XOR tree in the structure of PS-III [15, Fig. 7] at the cost of using about \(w\) times more flip-flops and significantly longer multiplication delay. The proposed structure DL-SRB-\(a\) together with “High-speed” structure in [14] has the second smallest critical path delay amongst all the structures under comparison.

In order to enable a better comparison, the area and delay complexities of the multipliers listed in Table II have been calculated and tabulated in Table III as a case study. Among the five field sizes recommended by the National Institute of Standards and Technology for elliptic curve applications [19], \(m = 233\) is the only one for which a type-II ONB exists. For this reason, in all the calculations made for Table III, the field size was selected as \(m = 233\). Note that \(F_{233}\) can be embedded into cyclotomic field \(F_{2^{233}}\). As mentioned earlier, accommodating ring type operations is a unique feature of redundant representation which not only provides a cost-free squaring operation but also eliminates the need for modular reduction in finite field operations. However, these remarkable advantages are achieved at the cost of a certain level of redundancy in the number of bits required to represent field elements. It should be noted that the appropriate choice of representation system generally depends on the overall specifications of the cryptographic system being implemented, such as field size (security level), the frequency of using multiplication and exponentiation operations, the overhead of using basis conversions, fault-tolerance, and so on. Although numerical comparison can reveal that the proposed architectures can effectively reduce the area-delay complexity of RB multipliers (by almost half) for 60% of all field sizes, the main focus of this paper is placed on about 20% of the fields for which \(T\) is equal to 2. In that case, not only the complexity of the proposed RB multipliers become comparable to that of ONB multipliers, but as suggested in Table III, they may even outperform ONB multipliers.

For each multiplier listed in the Table III, the calculations were made for three practical digit sizes 8, 16, and 32 based on the following assumptions. The required areas for an AND gate, an XOR gate, and a D-type flip-flop are assumed to be equal to \(\delta_A\), \(\delta_X\), and \(\delta_R\) square units, respectively.\(^1\) Parameter \(r\) in the second row of Table II represents the number of output product bits generated simultaneously in each clock cycle. To make a fair comparison in terms of gate counts and multiplication delay, the value of parameter \(r\) is assumed to be equal to \(w\) in Table III where needed. It is also assumed that the propagation delay of an XOR

\(^1\)In CMOS065LP standard cell library (from STMicroelectronics) a two-input AND gate, two-input XOR gate, and D-type flip-flop with set/reset are implemented by 6, 12, and 28–30 transistors and the area requirements for pertaining standard cells are reported in [25].
TABLE II

<table>
<thead>
<tr>
<th>Multiplier</th>
<th>Basis</th>
<th># AND</th>
<th># XOR</th>
<th># Reg</th>
<th>Critical Path Delay ($T_{cp}$)</th>
<th>Multiplication Delay</th>
</tr>
</thead>
<tbody>
<tr>
<td>WLMO [7]</td>
<td>ONB</td>
<td>$w(m-1)$</td>
<td></td>
<td>2$m$</td>
<td>$T_A + {\log_2(w(m-1))}T_X$</td>
<td>${\log_2(m+1)}T_{cp}$</td>
</tr>
<tr>
<td>DL-SPO [20]</td>
<td>ONB</td>
<td>$w(m-1)$</td>
<td></td>
<td>2$m$</td>
<td>$T_A + {\log_2(w(m-1))}T_X$</td>
<td>${\log_2(m+1)}T_{cp}$</td>
</tr>
<tr>
<td>WLSB [11]</td>
<td>ONB</td>
<td>$w(m-1)$</td>
<td></td>
<td>2$m$</td>
<td>$T_A + {\log_2(w(m-1))}T_X$</td>
<td>${\log_2(m+1)}T_{cp}$</td>
</tr>
<tr>
<td>Comb [BSWP] [12]</td>
<td>ONB</td>
<td>$w(m-1)$</td>
<td></td>
<td>3$m$</td>
<td>$T_A + {\log_2(w(m-1))}T_X$</td>
<td>${\log_2(m+1)}T_{cp}$</td>
</tr>
<tr>
<td>Mehri (1-folding) [22]</td>
<td>ONB</td>
<td>$2\left(\frac{1}{2}\right)wlog_2^3 \left(\frac{1}{8}(m+3)w^m + 3w + \frac{m}{2} + m\right)$</td>
<td></td>
<td>3$m$</td>
<td>$T_A + {2 + 2\log_2(m)}T_X$</td>
<td>${\log_2(m+1)}T_{cp}$</td>
</tr>
<tr>
<td>Hybrid PISO [8]</td>
<td>RB</td>
<td>$T_{wrm} + w$</td>
<td>$T_{wrm}$</td>
<td>$T_{wrm} + m$</td>
<td>$T_A + {\log_2(w(m+1))}T_X$</td>
<td>${\log_2(m+1)}T_{cp}$</td>
</tr>
<tr>
<td>PISO [25]</td>
<td>RB</td>
<td>$T_{wrm} + m$</td>
<td>$T_{wrm} + r$</td>
<td>$T_{wrm} + m + r$</td>
<td>$T_A + {\log_2(w(m+1))}T_X$</td>
<td>${\log_2(m+1)}T_{cp}$</td>
</tr>
<tr>
<td>Comb [12]</td>
<td>RB</td>
<td>$T_{wrm} + w$</td>
<td>$T_{wrm} + w$</td>
<td>$2(T_{wrm} + m)$</td>
<td>$T_A + {\log_2(w(m+1))}T_X$</td>
<td>${\log_2(m+1)}T_{cp}$</td>
</tr>
<tr>
<td>DL-LFSR [13]</td>
<td>RB</td>
<td>$T_{wrm} + w$</td>
<td>$(w+1)(T_{wrm} + 1)$</td>
<td>$(w+1)(T_{wrm} + 1)$</td>
<td>$T_A + {\log_2(w(m+1))}T_X$</td>
<td>${\log_2(m+1)}T_{cp}$</td>
</tr>
<tr>
<td>High-Speed [14]</td>
<td>RB</td>
<td>$T_{wrm} + w + \left[\left[\frac{T_{wrm}}{2}\right]\right]$</td>
<td>$(2w-1)(T_{wrm} + 1) + \left[\left[\frac{T_{wrm}}{2}\right]\right]$</td>
<td>$(2w-1)(T_{wrm} + 1) + \left[\left[\frac{T_{wrm}}{2}\right]\right]$</td>
<td>$T_A + T_X$</td>
<td>$\frac{1}{2}T_{cp} + \log_2 w\frac{T_X}{T_{cp}}$</td>
</tr>
<tr>
<td>PS-II ($d=2$) [15]</td>
<td>RB</td>
<td>$T_{wrm} + w$</td>
<td>$T_{wrm} + w$</td>
<td>$T_{wrm} + (w+1)$</td>
<td>$T_A + {1 + \log_2(d)}T_X$</td>
<td>$(\log_2 w + {\log_2 w + 1})T_{cp}$</td>
</tr>
<tr>
<td>PS-III [15]</td>
<td>RB</td>
<td>$T_{wrm} + w$</td>
<td>$T_{wrm} + w$</td>
<td>$T_{wrm} + (w+1)$</td>
<td>$T_A + {1 + \log_2(d)}T_X$</td>
<td>$(\log_2 w + {\log_2 w + 1})T_{cp}$</td>
</tr>
<tr>
<td>Proposed (DL-SRB-a)</td>
<td>RB</td>
<td>$w^{\frac{1}{m-1}}$</td>
<td>$(\bar{w}-1)\frac{T_{w}}{T} \left(\frac{\bar{w}}{2} + 1\right)$</td>
<td>$(\bar{w}-1)(T_{wrm} + 1) - \bar{w}$</td>
<td>$T_A + T_X$</td>
<td>$\frac{1}{\log_2 w}\left[\log_2 T_{cp} + \log_2 \frac{5\log_2 w}{T_{cp}}\right]$</td>
</tr>
<tr>
<td>Proposed (DL-SRB-b)</td>
<td>RB</td>
<td>$w^{\frac{1}{m-1}}$</td>
<td>$(\bar{w}-1)\frac{T_{w}}{T} \left(\frac{\bar{w}}{2} + 1\right)$</td>
<td>$(\bar{w}-1)(T_{wrm} + 1) - \bar{w}$</td>
<td>$T_A + T_X$</td>
<td>$\frac{1}{\log_2 w}\left[\log_2 T_{cp} + \log_2 \frac{5\log_2 w}{T_{cp}}\right]$</td>
</tr>
</tbody>
</table>

Note: $T > 1$ denotes the type of Gaussian Normal Basis (GNB). $\bar{w} = w - \frac{d}{(w-1)d} < \frac{m}{2}$, otherwise $\bar{w} = \arg\max_{x \in R} \left(\left(\frac{m}{2} - 1\right)d(\frac{m}{2} - 1)d < \frac{m}{2}\right)$.

Gate is twice as long as the delay of an AND gate. The column entitled “Area Cost” in the Table III shows the total area required by logic gates and registers for each multiplier. Assuming that the propagation delay of an AND gate is equal to 1 delay unit, the column entitled “Delay Cost” presents the relative multiplication delays in proportion to the delay of an AND gate.

As shown in Table III, DL-SRB-a offers much lower delay costs compared with the other multipliers. DL-SRB-b stands at the second position, having the second lowest delay cost except for only one case in which PS-III shows a slightly better performance when the digit size is equal to 8. In the design of digit-level finite field multipliers, there is always a tradeoff between delay and area costs as two important design factors and reducing one them generally results in an increase in the other one. To achieve a fair comparison, the area-delay product of the multipliers has been calculated and listed in the rightmost column of the Table III. As can be seen, both of the proposed architectures show much lower area-delay costs than all the existing RB multipliers for all digit sizes listed in the Table III. In the case of DL-SRB-b, the area-cost delay is 53%, 51%, and 47% lower than the most comparable architecture when $w = 8$, 16, and 32, respectively. In comparison with ONB multipliers, DL-SRB-b architecture offers 24%, 29%, and 7% area-delay improvement when the digit size changes from 8 to 16 and finally, 32, respectively.

It has been proven that if there exist a type-II ONB for representing field elements in $F_{2^n}$, then a cyclotomic field of degree $2m + 1$ ($T = 2$) always exists [8]. However, the inverse statement is not always true. The existence of a cyclotomic field of degree $n = 2m + 1$ for $F_{2^n}$ does not necessarily imply that a type-II ONB for that particular field size exists. As a result, the advantage of using the proposed multipliers would become more distinct when $T = 2$ but no ONB exists; for example, in the case of $m = 200, 204$, or 224.

V. HARDWARE IMPLEMENTATION

In order to verify the theoretical results, both the proposed multipliers were implemented in hardware as separate application specific integrated circuit (ASIC) modules for three digit sizes 8, 16, and 32. Multipliers have been realized for the binary extension field of degree 233. Note that in this case $T$ is equal to 2 and the result of Corollary 1 is applicable to the cyclotomic field of degree $n = 467$. All the implementations were carried out in seven-metal layer 65-nm CMOS process from STMicroelectronics with CMOS065LP standard cell library.

Fig. 4 shows the design flow used to realize each multiplier. The implementation process started with writing a
TABLE III
NUMERICAL COMPLEXITY COMPARISON OF DIGIT-LEVEL RB AND ONB MULTIPLIERS IN F_{2^{233}}
(EMBEDDED IN F_{2^{467}} IN THE CASE OF RB) FOR DIFFERENT VALUES OF \( w \)

<table>
<thead>
<tr>
<th>Multiplier</th>
<th>Basis</th>
<th>( w )</th>
<th>#AND ((N_A))</th>
<th>#XOR ((N_X))</th>
<th>#Register ((N_R))</th>
<th>Area Cost ((\delta_A \times N_A + \delta_X \times N_X + \delta_R \times N_R))</th>
<th>Delay Cost ((T_A = 1, T_X = 2))</th>
<th>Area x Delay</th>
</tr>
</thead>
<tbody>
<tr>
<td>WLMO [7]</td>
<td>ONB</td>
<td>8</td>
<td>3729</td>
<td>3712</td>
<td>466</td>
<td>35.755</td>
<td>309FA + 270FX = 570</td>
<td>20,380,450</td>
</tr>
<tr>
<td>DL-SIP [20]</td>
<td>RB</td>
<td>8</td>
<td>1861</td>
<td>6525</td>
<td>466</td>
<td>54.285</td>
<td>309FA + 150FX = 330</td>
<td>17,924,050</td>
</tr>
<tr>
<td>WLB [11]</td>
<td>ONB</td>
<td>8</td>
<td>1861</td>
<td>3726</td>
<td>466</td>
<td>60.096</td>
<td>309FA + 150FX = 330</td>
<td>9,301,980</td>
</tr>
<tr>
<td>Comb [BSWP] [12]</td>
<td>[22] (1-Golded)</td>
<td>8</td>
<td>1620</td>
<td>7310</td>
<td>659</td>
<td>32.481</td>
<td>159FA + 120FX = 255</td>
<td>12,835,680</td>
</tr>
<tr>
<td>Hybrid PISO [8]</td>
<td>RB</td>
<td>8</td>
<td>3736</td>
<td>3728</td>
<td>467</td>
<td>50.336</td>
<td>592FA + 513FX = 1121</td>
<td>40,242,779</td>
</tr>
<tr>
<td>PS [23]</td>
<td>RB</td>
<td>16</td>
<td>7472</td>
<td>7472</td>
<td>467</td>
<td>30.125</td>
<td>309FA + 270FX = 570</td>
<td>17,301,250</td>
</tr>
<tr>
<td>PS [23]</td>
<td>RB</td>
<td>32</td>
<td>7472</td>
<td>7472</td>
<td>467</td>
<td>40.978</td>
<td>594FA + 236FX = 531</td>
<td>21,665,207</td>
</tr>
<tr>
<td>DL-LFSR [13]</td>
<td>RB</td>
<td>8</td>
<td>3728</td>
<td>4203</td>
<td>467</td>
<td>38.869</td>
<td>594FA + 236FX = 531</td>
<td>20,373,499</td>
</tr>
<tr>
<td>High-Speed [14]</td>
<td>RB</td>
<td>8</td>
<td>3728</td>
<td>7005</td>
<td>3742</td>
<td>87,018</td>
<td>592FA + 223FX = 183</td>
<td>15,924,294</td>
</tr>
<tr>
<td>PS II ((g = 2)) [15]</td>
<td>RB</td>
<td>8</td>
<td>3736</td>
<td>3736</td>
<td>4235</td>
<td>55.468</td>
<td>629FA + 1247FX = 310</td>
<td>17,164,080</td>
</tr>
<tr>
<td>PS III [15]</td>
<td>RB</td>
<td>8</td>
<td>3736</td>
<td>3736</td>
<td>4506</td>
<td>118.506</td>
<td>607FA + 487FX = 126</td>
<td>16,816,816</td>
</tr>
<tr>
<td>DL-SRR-a</td>
<td>RB</td>
<td>8</td>
<td>3728</td>
<td>7223</td>
<td>5195</td>
<td>92.819</td>
<td>309FA + 343FX = 98</td>
<td>9,096,262</td>
</tr>
<tr>
<td>DL-SRR-b</td>
<td>RB</td>
<td>8</td>
<td>1864</td>
<td>3495</td>
<td>2331</td>
<td>48.232</td>
<td>309FA + 637FX = 156</td>
<td>7,524,192</td>
</tr>
</tbody>
</table>

Verilog code to describe the multiplier in hardware description language. 
\( C \) language was used to generate netlist blocks describing the numerous interconnections between logic gates as the main part of the RTL code. Then, the RTL design was synthesized to an optimal gate level design using Design Compiler from Synopsys. In the final stage, the netlist was imported to the Cadence SoC Encounter to perform floorplanning, cell placement, clock tree synthesis, reset net synthesis, and routing tasks. Three rounds of simulations were also carried out after RTL design, synthesis, and place&route stages to ensure the correct functionality of the multiplier. A set of golden results was initially created by simulating the multiplier with a large set of randomly generated input operands in MATLAB. Then, in each round of simulation, the same set of input data was fed into the multiplier and the product values were compared against the golden set. To obtain accurate power estimation, the final netlist was generated by Encounter and then simulated for 1000 pairs of random input vectors by NCSim to extract and store the switching activity information of all internal nets in value change dump (VCD) format. The switching activity information was fed into Encounter afterward to calculate the power consumption values.
The main characteristics of ASIC implementations for the proposed multipliers are listed in Table IV. It should be noted that the gap between the critical path delays measured in the post-synthesis stage and the post-place&route stage increases as the value of $\omega$ changes from 8 to 32. These changes in critical path delay stem from two facts. First, increasing the level of parallelism in the architectures of multipliers can significantly increase the capacitive load of certain nets, such as input $A$, reset, and clock. Consequently, a longer buffer chain is required to be able to properly drive logic cells connected to the high-fan-out nets, thus causing additional delay. Second, contributing factors, such as interconnect and parasitic capacitances, can only be taken into account for timing analysis after place&route when the layout is fully routed. Such factors eventually lead to a longer critical path delay.

VI. CONCLUSION

Two new digit-level SIPO finite field multipliers using redundant representation have been proposed. For about 60% of the field sizes within the practical range for ECC applications, the relationship between extension degree $m$ and the size of the smallest cyclotomic field, $(n)$, in which $\mathbb{F}_{2^m}$ can be embedded is expressed as $n = Tm + 1$ for $T$ even and greater than or equal to 2 [18]. In this case, a specific feature of redundant representation was used to alleviate the redundancy problem in this representation system. Numerical complexity comparison showed that both new architectures have the lowest delay cost compared with the existing RB architectures. One of the proposed architectures achieved at least 2.12 times higher performance (for different digit sizes over $\mathbb{F}_{2^{233}}$) in comparison with the most comparable RB architecture when considering area-delay complexity as a measure of performance. In about 20% of cases where $T = 2$, the proposal can show better performance than ONB multipliers, if existed, and can show much better performance than NB multipliers when $T = 2$ but no ONB exists (e.g., field sizes 200, 204, and 224). VLSI implementation of the proposed architectures for binary extension field of 233 and three practical digit sizes in 65-nm CMOS technology was also presented.

REFERENCES


Parham Hosseinzadeh Namin (S’09) received the B.Sc. degree in electrical engineering from the Islamic Azad University of Karaj, Alborz, Iran, in 2006, and the M.Sc. degree in telecommunication engineering from the University of Tabriz, Tabriz, Iran, in 2009. He is currently pursuing the Ph.D. degree with the Department of Electrical and Computer Engineering, University of Windsor, Windsor, ON, Canada.

His current research interests include digital and analog integrated circuits, architectures in finite fields, the hardware implementation of cryptosystems, spread spectrum communications, and cognitive radio networks.
Roberto Muscedere (S’95–M’96) was born in Windsor, ON, Canada, in 1973. He received the B.A.Sc., M.A.Sc., and Ph.D. degrees from the University of Windsor, Windsor, in 1996, 1999, and 2003, respectively, all in electrical engineering.

He managed the microelectronics computing environment with the Research Center for Integrated Microsystems, University of Windsor from 1996 to 2001. His current research interests include the implementation of high performance and low power VLSI circuits, full and semicustom VLSI design, computer arithmetic, HDL synthesis, and digital signal processing.

Majid Ahmadi (S’75–M’77–SM’84–F’02–LF’14) received the B.Sc. degree in electrical engineering from the Sharif University of Technology, Tehran, Iran, in 1971, and the Ph.D. degree in electrical engineering from the Imperial College of Science, Technology and Medicine, London, U.K., in 1977.

He has been with the Department of Electrical and Computer Engineering, University of Windsor, Windsor, ON, Canada, since 1980 and he is currently a Distinguished University Professor and an Associate Dean of Engineering for Research and Graduate Studies. He has co-authored the book Digital Filtering in One-D and Two-Dimensions; Design and Applications (New York, Plennum, 1989) and has authored over 500 articles in these areas. His current research interests include digital signal processing, machine vision, pattern recognition, neural network architectures, applications, and VLSI implementation, computer arithmetic, and MEMS.

Dr. Ahmadi is a fellow of IET. He was a recipient of an Honorable Mention Award from the Editorial Board of the Journal of Pattern Recognition in 1992, and the best paper award from the 2011 IEEE International Electro/Information Technology Conference. He received the Distinctive Contributed Paper Award from the Multiple-Valued Logic Conference Technical Committee and the IEEE Computer Society in 2000, the Distinguished University Professorship in 2003, the Faculty of Engineering Deans Special Recognition Award in 2007, and the University of Windsor Award for Excellence in Scholarship, Research, and Creative Activity in 2008. He was the IEEE-CAS representative on the Neural Network Council and the Chair of the IEEE Circuits and Systems Neural Systems Applications Technical Committee in 2000. He has served on the Editorial Board of the Journal of Circuits, Systems, and Computers as an Associate Editor and a Regional Editor from 1992 to 2012, an Associate Editor for the Journal of Pattern Recognition since 1992.