Journal of Computers Vol. 28, No. 1, 2017, pp. 87-103
doi:10.3966/199115592017022801008

A Novel Scalable Dual Basis GF(2^m) Multiplier Architecture

Liang-Hwa Chen[1], Yen-Ching Chang[2], Chiou-Yng Lee[1], and Po-Lun Chang[3]

[1] Department of Computer Information and Network Engineering,
Lunghwa University of Science and Technology, Taoyuan City 33306, Taiwan
{whallis2000, pp010}@mail.lhu.edu.tw

[2] Department of Medical Informatics, Chung Shan Medical University,
Taichung City 40201, Taiwan
nicholas@csmu.edu.tw

[3] Department of Electrical Engineering, Lunghwa University of Science and Technology,
Taoyuan City 33306, Taiwan
whc1223@ms7.hinet.net

Received 2 September 2015; Revised 16 December 2015; Accepted 6 April 2016

Abstract. Modern mobile communications and Internet transactions heavily rely on cryptosystems to ensure their security. These cryptosystems such as ECDSA usually rely on arithmetic operations over finite field GF(2^m). The most important arithmetic operation is multiplication, thus leading to the high demand for design of efficient multiplier. In this paper, based on the Hankel matrix-vector representation of dual basis multiplication, a novel low-complexity scalable digital circuit architecture for dual basis GF(2^m) multiplication is derived and proposed. This scalable architecture adopts most significant digit (MSD) first scheme and can perform *m*-bit multiplications with substantially smaller *d*-bit digits, and thus is feasible for implementing ECC cryptosystems such as ECDSA in resource-constrained environments such as embedded systems and smart cards. Analytical results exhibit that both area and time-area complexities of the proposed scalable architecture are substantially lower than those of the non-scalable architectures. Moreover, due to its features of regularity, modularity and concurrency, the proposed architecture is highly promising for VLSI implementations.

Keywords: elliptic curve cryptography (ECC), finite field, dual basis, Hankel matrix-vector, scalable multiplier

1 Introduction

Mobile communications and Internet transactions have become more and more popular in recent years, leading to increasing concern over the issue about integrity of transmitted data, against eavesdropping or unauthorized data altering. The advances in cryptography research thus have received increasing interest as well [1-3]. Cryptography and coding theory frequently involve finite field arithmetic operations, especially for the binary finite field GF(2^m). In particular, the elliptic curve cryptography (ECC) [4], which has become increasingly popular owing to its ability to realize a robust cryptosystem in smart cards, mobile phones and other resource-constrained environments, requires finite field arithmetic operations. The National Institute for Standards and Technology (NIST) has recommended the elliptic curve digital signature algorithm (ECDSA) standard [5], in which five GF(2^m), i.e. for $m = 163$, 233, 283, 409 and 571, are used to achieve adequate security. Among the basic arithmetic operations over GF(2^m), multiplication is the most important and time-consuming computation. Other complex arithmetic operations, including exponentiation, division and multiplicative inversion, can be performed by repeating multiplications, subsequently explaining the high demand for efficient design and implementation of the finite field multiplier with a low complexity.

For finite field $\mathrm{GF}(2^m)$, three conventionally adopted bases are polynomial basis (PB), normal basis (NB) and dual basis (DB) to represent its elements. Each representation has its own distinct advantages. For instance, the polynomial basis multiplier does not require basis conversion and is characterized by its regularity and simplicity. The normal basis multiplier is quite effective in performing squaring an element in the finite field. The dual basis multiplier needs the least number of gates that leads to the smallest chip area demand for VLSI implementation [6]. Pertinent literature in recent years has generally classified most finite field multipliers on these bases as bit-serial [7,8] and bit-parallel architectures [9-12]. Cryptography applications such as the above-mentioned ECDSA heavily rely on finite fields of large values of *m* (160 or higher) to achieve a sufficient level of security. To implement such cryptosystems, although the bit-serial architecture requires less chip area, its throughput is too slow. Conversely, the bit-parallel architecture is typically faster yet more complex and requires a significantly larger chip area and power consumption. Digit-serial architectures [13-15] have been proposed to facilitate the trade-off between throughput performance and hardware complexity. These architectures are based on a cut-set systolization scheme to accelerate the computational process. However, such an architecture has a space complexity similar to that of the original bit-level multiplier design, making it infeasible for implementing cryptosystems that require a large number of operand bits in resource-constrained environments. Scalable architectures [16-18] have subsequently been developed to satisfy the demands for implementing ECC cryptosystems such as ECDSA in resource-constrained environments. As a combination of serial and parallel schemes, the scalable architecture requires smaller chip area than that of the digit-serial architecture. In the scalable architecture, each *m*-bit operand word in the $\mathrm{GF}(2^m)$ field is separated into $k = \lceil m/d \rceil$ *d*-bit sub-words which are also termed as digits. Two operand words are multiplied in digits with a serial scheme while two digits are multiplied with a parallel scheme. Digit size *d* is normally selected to be substantially smaller than the operand bit length *m*. Thus, a substantially smaller hardware area multiplier (for *d*-bit operands) that can be easily implemented in resource-constrained environments is sufficient to perform all of the *m*-bit multiplications. Additionally, considering the trade-off between throughput performance and hardware complexity allows the scalable architecture to generate an optimal realization in hardware implementation by selecting an appropriate scalable factor, i.e. digit size *d*, especially in cases involving implementation of ECC cryptosystems in resource-constrained environments. Furthermore, the scalable architecture is flexible in terms of re-usage. Assume that a multiplier has been designed for 768-bit operand words. This multiplier cannot be applied directly to a system with an operand length of 1024 bits. The non-scalable (bit-parallel) multiplier must be re-designed to comply with system requirements. Conversely, the scalable architecture does not require a change in the core multiplier. By simply adjusting the size of the register array to satisfy the required longer operand length and reusing the core multiplier, the scalable multiplier can be applied to the new system. Doing so also makes it possible to achieve multiplications over the five $\mathrm{GF}(2^m)$ used in ECDSA with only one size (such as $d = 20$) of the core multiplier in order to obtain different levels of security. This feature is highly attractive for implementing ECDSA in resource-constrained environments such as smart cards and handset devices.

Scalable multiplication architectures have been developed for PB [17] and NB [18]. Correspondingly, this work presents a novel low-complexity scalable digital circuit architecture for dual basis (DB) multiplication over $\mathrm{GF}(2^m)$. We first propose an algorithm for dual basis $\mathrm{GF}(2^m)$ multiplication based on the Hankel matrix-vector representation. Then, the corresponding low-complexity scalable and systolic digital multiplier circuit architecture is designed and proposed. This scalable multiplier architecture adopts most significant digit (MSD) first scheme and is feasible for finite fields $\mathrm{GF}(2^m)$ generated by irreducible trinomials. Analytical results indicate that both area and time-area complexities of the proposed architecture are significantly lower than those of conventional digit-serial and bit-parallel multipliers. Additionally, for a given field size *m* of $\mathrm{GF}(2^m)$, the optimal digit size *d* can also be easily obtained from the analytical results to achieve the optimal trade-off between throughput performance and hardware complexity.

The rest of this paper is organized as follows: Section 2 briefly reviews the dual basis multiplication over $GF(2^m)$. Section 3 then introduces the proposed novel scalable algorithm and architecture for dual basis multiplication over $GF(2^m)$. Next, Section 4 discusses its time and space complexities. Conclusions are finally drawn in Section 5.

2 Preliminaries

As is well known, the finite field $GF(2^m)$ can be viewed as a vector space of dimension *m* over $GF(2)$, i.e. {0, 1}, where the field is generated by the irreducible polynomial $F(x) = f_0 + f_1 x + \ldots + f_{m-1} x^{m-1} + x^m$ of degree *m*, where the coefficients $f_i \in GF(2)$ for $i = 0, 1, \cdots, m-1$. Assume that α denotes a root of $F(x)$. Then, any element *A* in the finite field $GF(2^m)$ can be represented as $A = a_0 + a_1\alpha + a_2\alpha^2 + \cdots + a_{m-1}\alpha^{m-1}$, where the coordinates $a_i \in GF(2)$ for $i = 0, 1, \cdots, m-1$. The set $\left\{1, \alpha, \alpha^2, \ldots, \alpha^{m-1}\right\}$ is called the polynomial basis (PB) in $GF(2^m)$.

Definition 1. A basis $\{\beta_0, \beta_1, \ldots, \beta_{m-1}\}$ in $GF(2^m)$ is called the dual basis (DB) to the polynomial basis (PB) $\left\{1, \alpha, \alpha^2, \ldots, \alpha^{m-1}\right\}$ if the following condition is satisfied [7]:

$$Tr(\gamma\alpha^i\beta_j) = \begin{cases} 1, & if\ i = j \\ 0, & if\ i \neq j \end{cases} \quad \textbf{(1)}$$

where $Tr(\bullet)$ is the trace function from $GF(2^m)$ to $GF(2)$, and $\gamma \in GF(2^m), \gamma \neq 0$. □

For any two elements *A* and *B* in $GF(2^m)$ represented in polynomial and dual basis respectively, i.e.

$$A = a_0 + a_1\alpha + a_2\alpha^2 + \cdots + a_{m-1}\alpha^{m-1} = \sum_{i=0}^{m-1} a_i\alpha^i \quad \textbf{(2)}$$

$$B = b_0\beta_0 + b_1\beta_1 + \cdots + b_{m-1}\beta_{m-1} = \sum_{i=0}^{m-1} b_i\beta_i \quad \textbf{(3)}$$

where the coordinates $a_i, b_i \in GF(2)$. Their product $C = AB$ represented in dual basis, i.e.

$$C = c_0\beta_0 + c_1\beta_1 + \cdots + c_{m-1}\beta_{m-1} = \sum_{i=0}^{m-1} c_i\beta_i \quad \textbf{(4)}$$

where the coordinates $c_i \in GF(2)$, can be computed with the following discrete-time Wiener-Hopf equation (DTWHE) [19]:

$$\begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_{m-1} \end{bmatrix} = \begin{bmatrix} b_0 & b_1 & \cdots & b_{m-1} \\ b_1 & b_2 & \cdots & b_m \\ \vdots & \vdots & \ddots & \vdots \\ b_{m-1} & b_m & \cdots & b_{2m-2} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_{m-1} \end{bmatrix} \quad \textbf{(5)}$$

where $b_{m+i} = f_0 b_i + f_1 b_{i+1} + \cdots + f_{m-1} b_{i+m-1}$, for $i = 0, 1, \cdots, m-1$. The above equation is derived as follows: First, the coordinates b_i of *B* can be obtained as $b_i = Tr(\gamma\alpha^i B)$ because

$$\begin{aligned} Tr(\gamma\alpha^i B) &= Tr(\gamma\alpha^i(b_0\beta_0 + b_1\beta_1 + \cdots + b_{m-1}\beta_{m-1})) \\ &= b_0 Tr(\gamma\alpha^i\beta_0) + b_1 Tr(\gamma\alpha^i\beta_1) + \cdots + b_{m-1} Tr(\gamma\alpha^i\beta_{m-1}) \\ &= b_i, \quad \text{for } i = 0, 1, \cdots, m-1 \end{aligned}$$

from Definition 1. Next, since α is a root of $F(x) = f_0 + f_1 x + \ldots + f_{m-1}x^{m-1} + x^m$, i.e. $F(\alpha) = 0$, thus,

$$\begin{aligned} \alpha^m &= f_0 + f_1\alpha + f_2\alpha^2 + \cdots + f_{m-1}\alpha^{m-1} \\ \alpha^{m+i} &= f_0\alpha^i + f_1\alpha^{i+1} + f_2\alpha^{i+2} + \cdots + f_{m-1}\alpha^{i+m-1} = \sum_{j=0}^{m-1} f_j\alpha^{i+j} \end{aligned} \qquad \textbf{(6)}$$

Let us define b_{m+i} as $Tr(\gamma\alpha^{m+i}B)$. Then, according to (6), we get

$$\begin{aligned} b_{m+i} &\triangleq Tr(\gamma\alpha^{m+i}B) = Tr\left(\gamma(\sum_{j=0}^{m-1} f_j\alpha^{i+j})B\right) = \sum_{j=0}^{m-1} f_j Tr(\gamma\alpha^{i+j}B) \\ &= \sum_{j=0}^{m-1} f_j b_{i+j} = f_0 b_i + f_1 b_{i+1} + f_2 b_{i+2} + \cdots + f_{m-1} b_{i+m-1}, \quad \text{for } i = 0, 1, \cdots, m-1 \end{aligned} \qquad \textbf{(7)}$$

Based on Definition 1, the coordinates c_i of C can also be obtained as $c_i = Tr(\gamma\alpha^i C)$. Given that $C = AB$, $A = \sum_{i=0}^{m-1} a_i\alpha^i$ and, according to (7), we obtain

$$\begin{aligned} c_i &= Tr(\gamma\alpha^i C) = Tr(\gamma\alpha^i AB) = Tr\left(\gamma\alpha^i(\sum_{j=0}^{m-1} a_j\alpha^j)B\right) = \sum_{j=0}^{m-1} a_j Tr(\gamma\alpha^{i+j}B) \\ &= \sum_{j=0}^{m-1} a_j b_{i+j} = a_0 b_i + a_1 b_{i+1} + a_2 b_{i+2} + \cdots a_{m-1} b_{i+m-1}, \quad \text{for } i = 0, 1, \cdots, m-1. \end{aligned} \qquad \textbf{(8)}$$

The DTWHE in (5) is then obtained by expressing (8) in matrix form. Next, if we define the following vectors: $\boldsymbol{A} = [a_0, a_1, \cdots, a_{m-1}]$, $\boldsymbol{F} = [f_0, f_1, \cdots, f_{m-1}]$ and $\boldsymbol{B}^{(i)} = [b_i, b_{i+1}, \cdots, b_{i+m-1}]$, for $i = 0, 1, \cdots, m-1$, then, (7) and (8) can also be expressed as

$$b_{m+i} = \boldsymbol{B}^{(i)} \odot \boldsymbol{F} \qquad \textbf{(9)}$$

$$c_i = \boldsymbol{B}^{(i)} \odot \boldsymbol{A} \qquad \textbf{(10)}$$

where "$\odot$" denotes the inner product operation of two vectors. Notably, $\boldsymbol{B}^{(0)} = \boldsymbol{B} = [b_0, b_1, \cdots, b_{m-1}]$.

Here, an example is given for illustration: Let $m = 3$, then, $\boldsymbol{A} = [a_0, a_1, a_2]$, $\boldsymbol{F} = [f_0, f_1, f_2]$ and $\boldsymbol{B}^{(0)} = \boldsymbol{B} = [b_0, b_1, b_2]$. Next, according to the above derivations, we get $\boldsymbol{B}^{(1)} = [b_1, b_2, b_3]$, $\boldsymbol{B}^{(2)} = [b_2, b_3, b_4]$, where $b_3 = \boldsymbol{B}^{(0)} \odot \boldsymbol{F} = f_0 b_0 + f_1 b_1 + f_2 b_2$ and $b_4 = \boldsymbol{B}^{(1)} \odot \boldsymbol{F} = f_0 b_1 + f_1 b_2 + f_2 b_3$. Thus, the product vector

$$\begin{aligned} \boldsymbol{C} &= [c_0, c_1, c_2] = [\boldsymbol{B}^{(0)} \odot \boldsymbol{A},\ \boldsymbol{B}^{(1)} \odot \boldsymbol{A},\ \boldsymbol{B}^{(2)} \odot \boldsymbol{A}] \\ &= [a_0 b_0 + a_1 b_1 + a_2 b_2,\quad a_0 b_1 + a_1 b_2 + a_2 b_3,\quad a_0 b_2 + a_1 b_3 + a_2 b_4] \end{aligned}$$

is obtained.

Based on (9) and (10), Lee *et al.* [9] developed a bit-parallel systolic DB multiplier circuit architecture consisting of m^2 cell, which comprises the following digital logic gates: one AND gate, one XOR gate and two 1-bit latches. Owing to the regularity of its architecture, this DB multiplier is feasible for VLSI implementation. However, for a large field size of binary finite fields such as the five GF(2^m), $m = 163, 233, 283, 409$ and 571, adopted by ECDSA, the corresponding large space complexity $O(m^2)$ makes such a multiplier infeasible for implementing in constrained hardware environments such as smart cards and mobile handsets. To overcome this problem, the next section introduces a novel scalable scheme for DB multiplication that divides the *m*-bit operand word into several *d*-bit digits and, then, iteratively applies a smaller scale multiplier to achieve the complete *m*-bit multiplication.

3 Proposed Scalable Dual Basis Multiplier over GF(2^m)

Deriving the scalable architecture of DB multiplier initially requires introducing the Hankel matrix-vector representation.

3.1 Hankel Matrix-vector Representation

Definition 2. An $m \times n$ matrix H is referred to as a Hankel matrix if it satisfies the relation $\mathbf{H}(p,q) = \mathbf{H}(p+1,q-1)$, for $0 \le p \le m-2$, $1 \le q \le n-1$, where $\mathbf{H}(p,q)$ represents the element in the intersection of row *i* and column *j*. □

A Hankel matrix can be determined entirely by the $m+n-1$ entries that are located on its first row and last column. Namely, the matrix can be defined by the corresponding Hankel vector $\overline{\boldsymbol{H}} = [h_0, h_1, \cdots, h_{m+n-2}]$. For example, the Hankel matrix

$$\mathbf{H} = \begin{bmatrix} h_0\, h_1\, h_2 \\ h_1\, h_2\, h_3 \\ h_2\, h_3\, h_4 \\ h_3\, h_4\, h_5 \end{bmatrix}$$

has the corresponding Hankel vector $\overline{\boldsymbol{H}} = [h_0, h_1, h_2, h_3, h_4, h_5]$. With the Hankel *matrix-vector representation*, the product of a Hankel matrix $\mathbf{H}$ and a vector $\boldsymbol{V} = [v_0, v_1, \cdots v_{n-1}]$, i.e. $\mathbf{H}\boldsymbol{V}^T$ where T denotes transpose operator, is denoted as $\overline{\boldsymbol{H}} \otimes \boldsymbol{V}$. With such notation, the DB multiplication in (5) can be expressed as

$$\boldsymbol{C}^T = \mathbf{B}\boldsymbol{A}^T = \overline{\boldsymbol{B}} \otimes \boldsymbol{A} \qquad \textbf{(11)}$$

where $\overline{\boldsymbol{B}} = [b_0, b_1, \cdots, b_{m-1}, b_m, \cdots, b_{2m-2}]$ represents the corresponding Hankel vector of the Hankel matrix in (5).

3.2 Proposed Algorithm

For the elements *A*, *B* and *C* defined in (2), (3) and (4), repectively, select the digit size as *d*, and define $k = \lceil m/d \rceil$. If the word length *m* is divisible by *d*, then $m = kd$. Otherwise, $(k-1)d < m < kd$. In this case, for simplifying the derivation, we can extend *A* with appending zero bits $a_m, \cdots, a_{kd-1}$ and get the new word length $m' = kd$ which is divisible by *d*. The additionally required bits $b_{2m-1}, \cdots, b_{2m'-2}$ in (5) now are still calculated according to (7). In the resulting *C*, bits $c_m, \cdots, c_{m'-1}$ are redundant and should be disregarded. Thus, in the following, we only consider the case where *m* is divisible by *d*.

For the element *A* expressed in (2), we can rewrite it as the combination of the *k* sub-elements $A_0, A_1, \cdots, A_{k-1}$:

$$A = A_{k-1}\alpha^{(k-1)d} + A_{k-2}\alpha^{(k-2)d} + \cdots + A_2\alpha^{2d} + A_1\alpha^d + A_0 \qquad \textbf{(12)}$$

where

$$\begin{aligned} A_0 &= a_0 + a_1\alpha + a_2\alpha^2 + \cdots + a_{d-1}\alpha^{d-1} \\ A_1 &= a_d + a_{d+1}\alpha + a_{d+2}\alpha^2 + \cdots + a_{2d-1}\alpha^{d-1} \\ &\vdots \\ A_{k-1} &= a_{(k-1)d} + a_{(k-1)d+1}\alpha + \cdots + a_{kd-1}\alpha^{d-1} \end{aligned} \qquad \textbf{(13)}$$

Then, the product of *A* and *B*, i.e. *C* can be expressed as:

$$\begin{aligned} C = BA &= B(A_{k-1}\alpha^{(k-1)d} + A_{k-2}\alpha^{(k-2)d} + \cdots + A_2\alpha^{2d} + A_1\alpha^d + A_0) \\ &= BA_{k-1}\alpha^{(k-1)d} + BA_{k-2}\alpha^{(k-2)d} + \cdots + BA_1\alpha^d + BA_0 \\ &= E_{k-1}\alpha^{(k-1)d} + E_{k-2}\alpha^{(k-2)d} + \cdots + E_1\alpha^d + E_0 \\ &= ((\cdots(E_{k-1}\alpha^d + E_{k-2})\alpha^d + \cdots + E_1)\alpha^d + E_0 \end{aligned} \quad \textbf{(14)}$$

where E_i denotes the product of B and the sub-element A_i, i.e. BA_i which is a dual base element:

$$E_i \triangleq BA_i = e_{i,0}\beta_0 + e_{i,1}\beta_1 + \cdots + e_{i,m-1}\beta_{m-1}, \quad \text{for } i = 0,1,\cdots,k-1. \quad \textbf{(15)}$$

The j-th coordinate of E_i, i.e. $e_{i,j} \in \mathrm{GF}(2)$, can be obtained based on Definition 1 as:

$$\begin{aligned} e_{i,j} &= Tr(\gamma\alpha^j E_i) = Tr(\gamma\alpha^j BA_i) \\ &= Tr(\gamma\alpha^j B(a_{id} + a_{id+1}\alpha + \cdots + a_{id+d-1}\alpha^{d-1})) \\ &= a_{id}Tr(\gamma\alpha^j B) + a_{id+1}Tr(\gamma\alpha^{j+1}B) + \cdots + a_{id+d-1}Tr(\gamma\alpha^{j+d-1}B) \\ &= a_{id}b_j + a_{id+1}b_{j+1} + \cdots + a_{id+d-1}b_{j+d-1}, \quad \text{for } j = 0,1,\cdots,m-1 \end{aligned} \quad \textbf{(16)}$$

Expressing (16) in matrix form, we thus get the corresponding vector of E_i, i.e. $\boldsymbol{E}_i^T \triangleq [e_{i,0}, e_{i,1}, \cdots, e_{i,m-1}]^T = \mathbf{B}\boldsymbol{A}_i^T$, where

$$\boldsymbol{A}_i = [a_{id}, a_{id+1}, \cdots, a_{id+d-1}], \quad \text{for } i = 0,1,\cdots,k-1 \quad \textbf{(17)}$$

are the corresponding vectors of sub-elements A_i and the corresponding $m \times d$ Hankel matrix of element B is

$$\mathbf{B} = \begin{bmatrix} b_0 & b_1 & b_2 & \cdots & b_{d-1} \\ b_1 & b_2 & b_3 & \cdots & b_d \\ b_2 & b_3 & b_4 & \cdots & b_{d+1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ b_{m-1} & b_m & b_{m+1} & \cdots & b_{m+d-2} \end{bmatrix} \quad \textbf{(18)}$$

For the case that $m = 9$, $d = 3$ and then $k = \lceil m/d \rceil = \lceil 9/3 \rceil = 3$ as an example, the above $\boldsymbol{A}_i$'s are $\boldsymbol{A}_0 = [a_0, a_1, a_2]$, $\boldsymbol{A}_1 = [a_3, a_4, a_5]$, $\boldsymbol{A}_2 = [a_6, a_7, a_8]$ and the 9×3 Hankel matrix $\mathbf{B}$ is

$$\mathbf{B} = \begin{bmatrix} b_0 & b_1 & b_2 \\ b_1 & b_2 & b_3 \\ b_2 & b_3 & b_4 \\ \vdots & \vdots & \vdots \\ b_8 & b_9 & b_{10} \end{bmatrix}$$

The above $\boldsymbol{E}_i$'s are then $\boldsymbol{E}_0^T = \mathbf{B}\boldsymbol{A}_0^T$, $\boldsymbol{E}_1^T = \mathbf{B}\boldsymbol{A}_1^T$ and $\boldsymbol{E}_2^T = \mathbf{B}\boldsymbol{A}_2^T$.

The matrix expression corresponding to (14) is thus obtained as

$$\begin{aligned} \boldsymbol{C}^T &= [c_0, c_1, \cdots, c_{m-1}]^T = (\mathbf{B}\boldsymbol{A}_{k-1}^T)\alpha^{(k-1)d} + (\mathbf{B}\boldsymbol{A}_{k-2}^T)\alpha^{(k-2)d} + \cdots + (\mathbf{B}\boldsymbol{A}_1^T)\alpha^d + \mathbf{B}\boldsymbol{A}_0^T \\ &= \boldsymbol{E}_{k-1}^T\alpha^{(k-1)d} + \boldsymbol{E}_{k-2}^T\alpha^{(k-2)d} + \cdots + \boldsymbol{E}_1^T\alpha^d + \boldsymbol{E}_0^T \\ &= ((\cdots(\boldsymbol{E}_{k-1}^T\alpha^d + \boldsymbol{E}_{k-2}^T)\alpha^d + \cdots + \boldsymbol{E}_1^T)\alpha^d + \boldsymbol{E}_0^T \end{aligned} \quad \textbf{(19)}$$

The corresponding Hankel vector of the above matrix $\mathbf{B}$ (eq. (18)) is then

$$\overline{\boldsymbol{B}} = [b_0, b_1, \cdots, b_{d-1}, b_d, \cdots, b_{m-1}, b_m, \cdots, b_{m+d-2}] \quad \textbf{(20)}$$

With the Hankel matrix-vector representation, eq. (19) can be rewritten as

$$\begin{aligned} C^T &= (\overline{B} \otimes A_{k-1})\alpha^{(k-1)d} + (\overline{B} \otimes A_{k-2})\alpha^{(k-2)d} + \cdots + (\overline{B} \otimes A_1)\alpha^d + \overline{B} \otimes A_0 \\ &= ((\cdots(\overline{B} \otimes A_{k-1}\alpha^d + \overline{B} \otimes A_{k-2})\alpha^d + \cdots + \overline{B} \otimes A_1)\alpha^d + \overline{B} \otimes A_0 \\ &= ((\cdots(E_{k-1}{}^T\alpha^d + E_{k-2}{}^T)\alpha^d + \cdots + E_1{}^T)\alpha^d + E_0{}^T \end{aligned} \quad \textbf{(21)}$$

where $E_i^T = \mathbf{B}A_i^T = \overline{B} \otimes A_i$, for $i = 0,1,\cdots,k-1$. Notably, all contents in all parentheses in (21) are dual basis element vectors. The whole computation thus involves k Hankel multiplications ($\overline{B} \otimes A_i$, $i = 0,1,\cdots,k-1$) and k nested multiplications of α^d to dual basis element vectors. For the case that $m = 9$, $d = 3$ and then $k = 3$, the Hankel vector is

$$\overline{B} = [b_0, b_1, b_2, b_3, b_4, b_5, b_6, b_7, b_8, b_9, b_{10}]$$

and the product vector C is

$$C^T = (\overline{B} \otimes A_2)\alpha^6 + (\overline{B} \otimes A_1)\alpha^3 + \overline{B} \otimes A_0 = (E_2{}^T\alpha^3 + E_1{}^T)\alpha^3 + E_0{}^T$$

The effect of multiplying a dual basis element vector $E \triangleq [e_0, e_1, \cdots, e_{m-1}]$ by α^d is just a d-bit left-shifting operation on vector E. That is,

$$E\alpha^d = [e_d, e_{d+1}, \cdots, e_{m-1}, e_m, \cdots, e_{d+m-1}] \quad \textbf{(22)}$$

The latter d shifted-in bits, i.e. $e_m, e_{m+1}, \cdots, e_{m+d-1}$, can be reformed as

$$[e_m, e_{m+1}, \cdots, e_{m+d-1}] = [(e_n + e_0), (e_{n+1} + e_1), \cdots, (e_{n+d-1} + e_{d-1})]$$

when the generating function of GF(2^m) is an irreducible trinomial, i.e. $F(x) = x^m + x^n + 1$. Thus, these d shifted-in bits can be generated from the original bits of vector E, i.e. $e_0, e_1, \cdots, e_{m-1}$, with d XOR gates. The whole $E\alpha^d$ vector can thus be expressed as

$$E\alpha^d = [e_d, e_{d+1}, \cdots, e_{m-1},\ (e_n + e_0), (e_{n+1} + e_1), \cdots, (e_{n+d-1} + e_{d-1})] \quad \textbf{(23)}$$

The detailed derivation is described in Appendix A. For the case that $m = 9$, $d = 3$ and $n = 3$ as an example, $E\alpha^3$ is just a 3-bit left-shifting operation on vector E, that is,

$$E\alpha^3 = [e_3, e_4, e_5, e_6, e_7, e_8, e_9, e_{10}, e_{11}] = [e_3, e_4, e_5, e_6, e_7, e_8, (e_3 + e_0), (e_4 + e_1), (e_5 + e_2)].$$

Next, we divide $\overline{B} \otimes A_i = \mathbf{B}A_i^T$ into k parts in order to adopt a smaller size of DB multiplier architecture. From (17) and (18), we can rewrite $\overline{B} \otimes A_i = \mathbf{B}A_i^T$ as follows:

$$\overline{B} \otimes A_i = \mathbf{B}A_i^T = \begin{bmatrix} \mathbf{B}_0 \\ \mathbf{B}_1 \\ \vdots \\ \mathbf{B}_{k-1} \end{bmatrix} A_i^T = \begin{bmatrix} \mathbf{B}_0 A_i^T \\ \mathbf{B}_1 A_i^T \\ \vdots \\ \mathbf{B}_{k-1} A_i^T \end{bmatrix} = \begin{bmatrix} \overline{B}_0 \otimes A_i \\ \overline{B}_1 \otimes A_i \\ \vdots \\ \overline{B}_{k-1} \otimes A_i \end{bmatrix} \quad \textbf{(24)}$$

where

$$\mathbf{B}_j = \begin{bmatrix} b_{jd} & b_{jd+1} & b_{jd+2} & \cdots & b_{jd+d-1} \\ b_{jd+1} & b_{jd+2} & b_{jd+3} & \cdots & b_{jd+d} \\ b_{jd+2} & b_{jd+3} & b_{jd+4} & \cdots & b_{jd+d+1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ b_{jd+d-1} & b_{jd+d} & b_{jd+d+1} & \cdots & b_{jd+2d-2} \end{bmatrix}, \quad \text{for } j = 0,1,\cdots,k-1 \quad \textbf{(25)}$$

are $d \times d$ Hankel matrices whose corresponding $1 \times (2d-1)$ Hankel vectors are

$$\overline{\boldsymbol{B}}_j = [b_{jd}, b_{jd+1}, \cdots, b_{jd+d-1}, b_{jd+d}, \cdots, b_{jd+2d-2}], \quad \text{for } j = 0,1,\cdots,k-1 \qquad \textbf{(26)}$$

From (24), it reveals that each Hankel multiplication, $\overline{\boldsymbol{B}} \otimes \boldsymbol{A}_i$, can be composed of k smaller size of Hankel multiplications, i.e. $\overline{\boldsymbol{B}}_j \otimes \boldsymbol{A}_i$, $j = 0,1,\cdots,k-1$. For the case that $m = 9$, $d = 3$ and then $k = 3$, the above 1×5 Hankel vectors $\overline{\boldsymbol{B}}_j$'s are

$$\overline{\boldsymbol{B}}_0 = [b_0, b_1, b_2, b_3, b_4], \quad \overline{\boldsymbol{B}}_1 = [b_3, b_4, b_5, b_6, b_7], \quad \overline{\boldsymbol{B}}_2 = [b_6, b_7, b_8, b_9, b_{10}]$$

and the Hankel multiplication $\overline{\boldsymbol{B}} \otimes \boldsymbol{A}_i$ can be expressed as:

$$\overline{\boldsymbol{B}} \otimes \boldsymbol{A}_i = \mathbf{B}\boldsymbol{A}_i^T = \begin{bmatrix} \mathbf{B}_0 \\ \mathbf{B}_1 \\ \mathbf{B}_2 \end{bmatrix} \boldsymbol{A}_i^T = \begin{bmatrix} \overline{\boldsymbol{B}}_0 \otimes \boldsymbol{A}_i \\ \overline{\boldsymbol{B}}_1 \otimes \boldsymbol{A}_i \\ \overline{\boldsymbol{B}}_2 \otimes \boldsymbol{A}_i \end{bmatrix}$$

Next, for convenience, let $\overline{\boldsymbol{b}}_j$ denotes the d-bit length sub-vectors:

$$\overline{\boldsymbol{b}}_j \triangleq [b_{jd}, b_{jd+1}, \cdots, b_{jd+d-1}], \quad \text{for } j = 0,1,\cdots,k \qquad \textbf{(27)}$$

That is, $\overline{\boldsymbol{b}}_0 \triangleq [b_0, b_1, \cdots, b_{d-1}]$, $\overline{\boldsymbol{b}}_1 \triangleq [b_d, b_{d+1}, \cdots, b_{2d-1}]$, …, $\overline{\boldsymbol{b}}_{k-1} \triangleq [b_{m-d}, b_{m-d+1}, \cdots, b_{m-1}]$ and $\overline{\boldsymbol{b}}_k \triangleq [b_m, b_{m+1}, \cdots, b_{m+d-1}]$. Then, the $(2d-1)$-bit length Hankel vectors $\overline{\boldsymbol{B}}_j$ (eq. (26)) can be further expressed as:

$$\overline{\boldsymbol{B}}_j = [\overline{\boldsymbol{b}}_j, \ \overline{\boldsymbol{b}}_{j+1} \setminus b_{jd+2d-1}], \quad \text{for } j = 0,1,\cdots,k-1 \qquad \textbf{(28)}$$

where $\overline{\boldsymbol{b}}_{j+1} \setminus b_{jd+2d-1}$ denotes removing the lattermost bit $b_{jd+2d-1}$ from $\overline{\boldsymbol{b}}_{j+1}$. Beside, the original Hankel vector $\overline{\boldsymbol{B}}$ (eq. (20)) can be composed of $k+1$ sub-vectors as follows:

$$\overline{\boldsymbol{B}} = [\overline{\boldsymbol{b}}_0, \ \overline{\boldsymbol{b}}_1, \ \cdots, \ \overline{\boldsymbol{b}}_{k-1}, \overline{\boldsymbol{b}}_k] \qquad \textbf{(29)}$$

For the case that $m = 9$, $d = 3$ and then $k = 3$, the above $\overline{\boldsymbol{b}}_j$'s are $\overline{\boldsymbol{b}}_0 \triangleq [b_0, b_1, b_2]$, $\overline{\boldsymbol{b}}_1 \triangleq [b_3, b_4, b_5]$, $\overline{\boldsymbol{b}}_2 \triangleq [b_6, b_7, b_8]$ and $\overline{\boldsymbol{b}}_3 \triangleq [b_9, b_{10}, b_{11}]$. The above $\overline{\boldsymbol{B}}_j$'s are then:

$$\begin{aligned} \overline{\boldsymbol{B}}_0 &= [b_0, b_1, b_2, b_3, b_4] = [\overline{\boldsymbol{b}}_0, \ \overline{\boldsymbol{b}}_1 \setminus b_5] \\ \overline{\boldsymbol{B}}_1 &= [b_3, b_4, b_5, b_6, b_7] = [\overline{\boldsymbol{b}}_1, \ \overline{\boldsymbol{b}}_2 \setminus b_8] \\ \overline{\boldsymbol{B}}_2 &= [b_6, b_7, b_8, b_9, b_{10}] = [\overline{\boldsymbol{b}}_2, \ \overline{\boldsymbol{b}}_3 \setminus b_{11}] \end{aligned}$$

Notably, all $\overline{\boldsymbol{b}}_0, \ \overline{\boldsymbol{b}}_1, \ \cdots, \ \overline{\boldsymbol{b}}_{k-1}$ can be obtained directly from the coordinate bits of the original input vector $\boldsymbol{B}$, i.e. $b_0, b_1, \cdots, b_{m-1}$. However, $\overline{\boldsymbol{b}}_k \triangleq [b_m, b_{m+1}, \cdots, b_{m+d-1}]$ has to be computed from $b_0, b_1, \cdots, b_{m-1}$. As described in Appendix B, when the generating function of GF(2^m) is an irreducible trinomial, i.e. $F(x) = x^m + x^n + 1$, $\overline{\boldsymbol{b}}_k$ can be reformed as

$$\overline{\boldsymbol{b}}_k = [(b_n + b_0), (b_{n+1} + b_1), \cdots, (b_{n+d-1} + b_{d-1})] \qquad \textbf{(30)}$$

Thus, the d bits of $\overline{\boldsymbol{b}}_k$ can be generated from $b_0, b_1, \cdots, b_{m-1}$ with d XOR gates. For the case that $m = 9$, $d = 3$, $n = 3$ and then $k = 3$, the 3 bits of the vector $\overline{\boldsymbol{b}}_3$ can then be generated from $b_0, b_1, \cdots, b_8$ with 3 XOR gates, i.e. $\overline{\boldsymbol{b}}_3 \triangleq [b_9, b_{10}, b_{11}] = [(b_3 + b_0), (b_4 + b_1), (b_5 + b_2)]$.

Next, a MSD (most significant digit) first scalable dual-basis multiplication algorithm with digit size d is derived based on the above derivations as follows:

```
Algorithm 1:
Input: $\boldsymbol{A}=[a_0, a_1, \cdots, a_{m-1}]$, $\boldsymbol{B}=[b_0, b_1, \cdots, b_{m-1}]$
Output: $\boldsymbol{C}=[c_0, c_1, \cdots, c_{m-1}]=\boldsymbol{AB}$
1. Initial step:
  1.1 Clear output vector $\boldsymbol{C}$
  1.2 Construct vectors $\boldsymbol{A}_i$, $i=0,1,\cdots,k-1$, from $\boldsymbol{A}$ according to (17)
  1.3 Construct vectors $\overline{\boldsymbol{b}}_j$, $j=0,1,\cdots,k-1$ and $\overline{\boldsymbol{b}}_k$ from $\boldsymbol{B}$ according to (27) and (30), respectively
2. Multiplication step:
  2.1 For $i=1$ to $k$ do
  2.2     Clear vector $\boldsymbol{E}=[e_0, e_1, \cdots, e_{m-1}]=[\boldsymbol{E}_0, \boldsymbol{E}_1, \cdots, \boldsymbol{E}_{k-1}]$ where $\boldsymbol{E}_0, \cdots, \boldsymbol{E}_{k-1}$ are $d$-bit sub-vectors
  2.3     For $j=0$ to $k-1$ do
  2.4         Set $\overline{\boldsymbol{B}}_j=[\overline{\boldsymbol{b}}_j, \overline{\boldsymbol{b}}_{j+1} \setminus b_{jd+2d-1}]$
  2.5         Compute $\boldsymbol{E}_j=\left(\overline{\boldsymbol{B}}_j \otimes \boldsymbol{A}_{k-i}\right)^T$
  2.6     Endfor
  2.7     Set $\boldsymbol{E}=[\boldsymbol{E}_0, \boldsymbol{E}_1, \cdots, \boldsymbol{E}_{k-1}]=\left[\left(\overline{\boldsymbol{B}}_0 \otimes \boldsymbol{A}_{k-i}\right)^T, \left(\overline{\boldsymbol{B}}_1 \otimes \boldsymbol{A}_{k-i}\right)^T, \cdots, \left(\overline{\boldsymbol{B}}_{k-1} \otimes \boldsymbol{A}_{k-i}\right)^T\right]=\left(\overline{\boldsymbol{B}} \otimes \boldsymbol{A}_{k-i}\right)^T$
  2.8     Set $\boldsymbol{C}=\boldsymbol{C}\alpha^d$ by performing the $d$-bit left-shifting operation on $\boldsymbol{C}$ according to (23)
  2.9     Set $\boldsymbol{C}=\boldsymbol{C}+\boldsymbol{E}=\boldsymbol{C}+\left(\overline{\boldsymbol{B}} \otimes \boldsymbol{A}_{k-i}\right)^T$
  2.10 Endfor
3. Return $\boldsymbol{C}$
```

Algorithm 1 implements eq. (21) in MSD first scheme (for PB element $\boldsymbol{A}$). The PB element $\boldsymbol{A}$ is divided into k sub-vectors (digits) $\boldsymbol{A}_i, i=0,1,\cdots,k-1$. The DB element $\boldsymbol{B}$ is divided into k sub-vectors $\overline{\boldsymbol{b}}_j$, $j=0,1,\cdots,k-1$, and $\overline{\boldsymbol{b}}_k$ is generated. The complete product result vector $\boldsymbol{C}$ is obtained after all k rounds of computations (outer loop) are performed. In each computation round, k smaller size of Hankel multiplications ($\overline{\boldsymbol{B}}_j \otimes \boldsymbol{A}_{k-i}$) are performed (inner loop) to construct the required bigger size of Hankel multiplication ($\overline{\boldsymbol{B}} \otimes \boldsymbol{A}_{k-i}$). Note that the k sub-vectors $\boldsymbol{A}_i$ are fed into the computation process in MSD first manner (from $\boldsymbol{A}_{k-1}$ to $\boldsymbol{A}_0$). Next, an illustrative example for Algorithm 1 is provided as follows:

Example 1. Assume that finite field $\mathrm{GF}(2^9)$ is constructed from $F(x)=x^9+x^3+1$ and the digit size d is set to 3. Namely, $m=9$, $n=3$ and $k=\lceil m/d \rceil=\lceil 9/3 \rceil=3$. The original binary input vectors are $\boldsymbol{A}=[a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8]$, $\boldsymbol{B}=[b_0, b_1, b_2, b_3, b_4, b_5, b_6, b_7, b_8]$. Then, according to Algorithm 1, we get

$$\boldsymbol{A}_0=[a_0, a_1, a_2], \quad \boldsymbol{A}_1=[a_3, a_4, a_5], \quad \boldsymbol{A}_2=[a_6, a_7, a_8]$$

and

$$\overline{\boldsymbol{b}}_0=[b_0, b_1, b_2], \quad \overline{\boldsymbol{b}}_1=[b_3, b_4, b_5], \quad \overline{\boldsymbol{b}}_2=[b_6, b_7, b_8]$$

$$\overline{\boldsymbol{b}}_3=[b_9, b_{10}, b_{11}]=[(b_3+b_0), (b_4+b_1), (b_5+b_2)]$$

It leads to

$$\overline{\boldsymbol{B}}_0=[\overline{\boldsymbol{b}}_0, \overline{\boldsymbol{b}}_1 \setminus b_5]=[b_0, b_1, b_2, b_3, b_4]$$

$$\overline{\boldsymbol{B}}_1=[\overline{\boldsymbol{b}}_1, \overline{\boldsymbol{b}}_2 \setminus b_8]=[b_3, b_4, b_5, b_6, b_7]$$

$$\overline{\boldsymbol{B}}_2 = [\overline{\boldsymbol{b}}_2,\ \overline{\boldsymbol{b}}_3 \setminus b_{11}] = [b_6, b_7, b_8, b_9, b_{10}].$$

Next, $k = 3$ computation rounds, according to Algorithm 1, are required to get the desired binary result vector $\boldsymbol{C} = \boldsymbol{AB}$. The vector $\boldsymbol{C}$ is initially set to be a zero vector, i.e. $\boldsymbol{C}^{(0)} = [0,0,0,0,0,0,0,0,0]$ where the superscript indicates the serial number of computation round. In round 1, $\overline{\boldsymbol{B}} \otimes \boldsymbol{A}_2$ is computed at first:

$$\boldsymbol{E}^{(1)} = \left(\overline{\boldsymbol{B}} \otimes \boldsymbol{A}_2\right)^T = [\boldsymbol{E}_0^{(1)}, \boldsymbol{E}_1^{(1)}, \boldsymbol{E}_2^{(1)}] = \left[\left(\overline{\boldsymbol{B}}_0 \otimes \boldsymbol{A}_2\right)^T, \left(\overline{\boldsymbol{B}}_1 \otimes \boldsymbol{A}_2\right)^T, \left(\overline{\boldsymbol{B}}_2 \otimes \boldsymbol{A}_2\right)^T\right]$$

$$= \left[\left(\begin{bmatrix} b_0 & b_1 & b_2 \\ b_1 & b_2 & b_3 \\ b_2 & b_3 & b_4 \end{bmatrix}\begin{bmatrix} a_6 \\ a_7 \\ a_8 \end{bmatrix}\right)^T, \left(\begin{bmatrix} b_3 & b_4 & b_5 \\ b_4 & b_5 & b_6 \\ b_5 & b_6 & b_7 \end{bmatrix}\begin{bmatrix} a_6 \\ a_7 \\ a_8 \end{bmatrix}\right)^T, \left(\begin{bmatrix} b_6 & b_7 & b_8 \\ b_7 & b_8 & b_9 \\ b_8 & b_9 & b_{10} \end{bmatrix}\begin{bmatrix} a_6 \\ a_7 \\ a_8 \end{bmatrix}\right)^T\right]$$

Next, take a 3-bit left-shifting operation on $\boldsymbol{C}^{(0)}$:

$$\boldsymbol{C}^{(0)}\alpha^3 = [0, 0, \cdots, 0]\alpha^3 = [0, 0, \cdots, 0]$$

which is still a zero vector due to that $\boldsymbol{C}^{(0)}$ is a zero vector. The result of round 1 is the summation of the above two computation results:

$$\boldsymbol{C}^{(1)} = [c_0^{(1)}, c_1^{(1)}, c_2^{(1)}, \cdots, c_8^{(1)}] = \boldsymbol{C}^{(0)}\alpha^3 + \boldsymbol{E}^{(1)} = \boldsymbol{E}^{(1)}$$

Next, in round 2, similar procedures are performed: $\overline{\boldsymbol{B}} \otimes \boldsymbol{A}_1$ is computed at first:

$$\boldsymbol{E}^{(2)} = \left(\overline{\boldsymbol{B}} \otimes \boldsymbol{A}_1\right)^T = [\boldsymbol{E}_0^{(2)}, \boldsymbol{E}_1^{(2)}, \boldsymbol{E}_2^{(2)}] = \left[\left(\overline{\boldsymbol{B}}_0 \otimes \boldsymbol{A}_1\right)^T, \left(\overline{\boldsymbol{B}}_1 \otimes \boldsymbol{A}_1\right)^T, \left(\overline{\boldsymbol{B}}_2 \otimes \boldsymbol{A}_1\right)^T\right]$$

$$= \left[\left(\begin{bmatrix} b_0 & b_1 & b_2 \\ b_1 & b_2 & b_3 \\ b_2 & b_3 & b_4 \end{bmatrix}\begin{bmatrix} a_3 \\ a_4 \\ a_5 \end{bmatrix}\right)^T, \left(\begin{bmatrix} b_3 & b_4 & b_5 \\ b_4 & b_5 & b_6 \\ b_5 & b_6 & b_7 \end{bmatrix}\begin{bmatrix} a_3 \\ a_4 \\ a_5 \end{bmatrix}\right)^T, \left(\begin{bmatrix} b_6 & b_7 & b_8 \\ b_7 & b_8 & b_9 \\ b_8 & b_9 & b_{10} \end{bmatrix}\begin{bmatrix} a_3 \\ a_4 \\ a_5 \end{bmatrix}\right)^T\right]$$

$\boldsymbol{C}^{(1)}$ is then left-shifted with 3-bits:

$$\boldsymbol{C}^{(1)}\alpha^3 = [c_0^{(1)}, c_1^{(1)}, c_2^{(1)}, \cdots, c_8^{(1)}]\alpha^3 = [c_3^{(1)}, c_4^{(1)}, c_5^{(1)}, \cdots, c_8^{(1)}, (c_3^{(1)} + c_0^{(1)}), (c_4^{(1)} + c_1^{(1)}), (c_5^{(1)} + c_2^{(1)})].$$

Then, take the summation to get the result of round 2:

$$\boldsymbol{C}^{(2)} = [c_0^{(2)}, c_1^{(2)}, c_2^{(2)}, \cdots, c_8^{(2)}] = \boldsymbol{C}^{(1)}\alpha^3 + \boldsymbol{E}^{(2)}$$

Next, in round 3, $\overline{\boldsymbol{B}} \otimes \boldsymbol{A}_0$ is computed:

$$\boldsymbol{E}^{(3)} = \left(\overline{\boldsymbol{B}} \otimes \boldsymbol{A}_0\right)^T = [\boldsymbol{E}_0^{(3)}, \boldsymbol{E}_1^{(3)}, \boldsymbol{E}_2^{(3)}] = \left[\left(\overline{\boldsymbol{B}}_0 \otimes \boldsymbol{A}_0\right)^T, \left(\overline{\boldsymbol{B}}_1 \otimes \boldsymbol{A}_0\right)^T, \left(\overline{\boldsymbol{B}}_2 \otimes \boldsymbol{A}_0\right)^T\right]$$

$$= \left[\left(\begin{bmatrix} b_0 & b_1 & b_2 \\ b_1 & b_2 & b_3 \\ b_2 & b_3 & b_4 \end{bmatrix}\begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix}\right)^T, \left(\begin{bmatrix} b_3 & b_4 & b_5 \\ b_4 & b_5 & b_6 \\ b_5 & b_6 & b_7 \end{bmatrix}\begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix}\right)^T, \left(\begin{bmatrix} b_6 & b_7 & b_8 \\ b_7 & b_8 & b_9 \\ b_8 & b_9 & b_{10} \end{bmatrix}\begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix}\right)^T\right]$$

$\boldsymbol{C}^{(2)}$ is left-shifted with 3-bits:

$$\boldsymbol{C}^{(2)}\alpha^3 = [c_0^{(2)}, c_1^{(2)}, c_2^{(2)}, \cdots, c_8^{(2)}]\alpha^3 = [c_3^{(2)}, c_4^{(2)}, c_5^{(2)}, \cdots, c_8^{(2)}, (c_3^{(2)} + c_0^{(2)}), (c_4^{(2)} + c_1^{(2)}), (c_5^{(2)} + c_2^{(2)})].$$

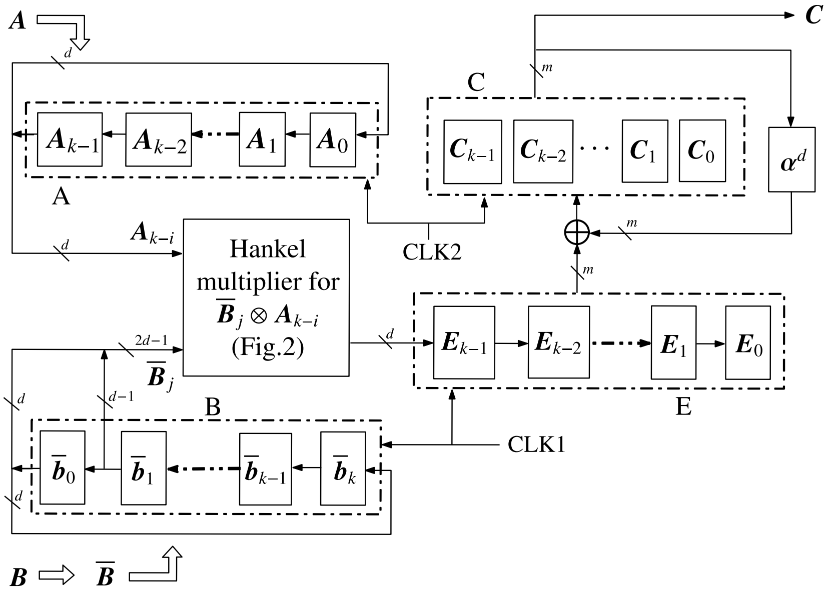
The result of round 3 is then obtained:

$$\boldsymbol{C}^{(3)} = [c_0^{(3)}, c_1^{(3)}, c_2^{(3)}, \cdots, c_8^{(3)}] = \boldsymbol{C}^{(2)}\alpha^3 + \boldsymbol{E}^{(3)}$$

This result, i.e. $\boldsymbol{C}^{(3)}$, is just the desired product vector $\boldsymbol{C} = \boldsymbol{AB}$.

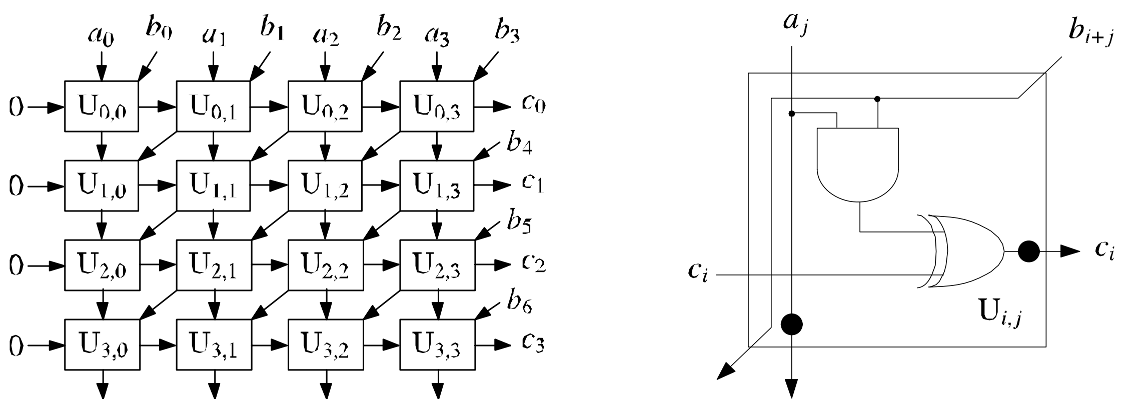
3.3 Proposed Architecture

Based on Algorithm 1, the proposed MSD scalable circuit architecture for dual-basis multiplication over $GF(2^m)$ is constructed (Fig. 1). This architecture is mainly composed of four registers for storing input vectors ***A***, ***B***, output vector ***C*** and product result $\overline{\boldsymbol{B}} \otimes \boldsymbol{A}_{k-i}$ respectively, one kernel $d \times d$ systolic Hankel multiplier, one summation circuit ($\oplus$) and one transformation circuit (α^d block) for vector ***C***. The kernel $d \times d$ systolic Hankel multiplier (Fig. 2 (a)) is applied to perform the Hankel matrix-vector multiplication $\overline{\boldsymbol{B}}_j \otimes \boldsymbol{A}_{k-i}$, and is composed of d^2 U-cell. Each U-cell (Fig. 2 (b)) consists of the following digital logic gates: one AND gate, one XOR gate and two 1-bit latches. Here, the AND gate and XOR gate are used to multiply and add two bits in $GF(2)$, respectively. The latches are used for synchronization of the computation. A corresponding LSD first scalable multiplier architecture had been proposed in [20] which is a little more complicated then this MSD architecture due to the unavoidable usage of multiplexers and switch circuits.



A
d
A
A_{k-1} A_{k-2} A_1 A_0
A_{k-i}
d
Hankel multiplier for $\overline{B}_j \otimes A_{k-i}$ (Fig.2)
2d−1
$\overline{B}_j$
d−1
d
d
B
$\overline{b}_0$ $\overline{b}_1$ $\overline{b}_{k-1}$ $\overline{b}_k$
$B \Rightarrow \overline{B}$
C
m
C
C_{k-1} C_{k-2} C_1 C_0
α^d
CLK2
m
m
d
E_{k-1} E_{k-2} E_1 E_0
E
CLK1

Fig. 1. Proposed MSD scalable DB $GF(2^m)$ multiplier architecture



a_0 b_0 a_1 b_1 a_2 b_2 a_3 b_3
0 $U_{0,0}$ $U_{0,1}$ $U_{0,2}$ $U_{0,3}$ c_0
b_4
0 $U_{1,0}$ $U_{1,1}$ $U_{1,2}$ $U_{1,3}$ c_1
b_5
0 $U_{2,0}$ $U_{2,1}$ $U_{2,2}$ $U_{2,3}$ c_2
b_6
0 $U_{3,0}$ $U_{3,1}$ $U_{3,2}$ $U_{3,3}$ c_3
a_j
b_{i+j}
c_i
c_i
$U_{i,j}$
(a)
(b)

Fig. 2. (a) $d \times d$ Systolic Hankel multiplier applied to perform $\overline{\boldsymbol{B}}_j \otimes \boldsymbol{A}_{k-i}$ for $d = 4$,
(b) Detailed circuit of a U-cell (symbol • denotes 1-bit latch)

In the proposed MSD multiplier architecture, the sub-vector $\boldsymbol{A}_{k-i}$ is sent into the kernel $d \times d$ systolic Hankel multiplier (Fig. 2 (a)) through the d bits $[a_0, a_1, \cdots, a_{d-1}]$ on the upper side to perform the Hankel multiplication $\overline{\boldsymbol{B}}_j \otimes \boldsymbol{A}_{k-i}$. In the mean time, the Hankel vector $\overline{\boldsymbol{B}}_j$ is sent into the kernel multiplier through $2d-1$ bits $[b_0, b_1, \cdots, b_{d-1}, b_d, \cdots, b_{2d-2}]$ on the upper and right sides. Meanwhile, a *d*-bit zero vector is sent into the multiplier from the left side. The *d*-bit product outcome $\overline{\boldsymbol{B}}_j \otimes \boldsymbol{A}_{k-i}$ is then obtained from the *d* bits $[c_0, c_1, \cdots, c_{d-1}]$ on the right side of this kernel multiplier.

Back to Fig. 1, register A consists of *k* *d*-bit latches ($\boldsymbol{A}_0, \boldsymbol{A}_1, \cdots, \boldsymbol{A}_{k-1}$) and register B consists of $k+1$ *d*-bit latches ($\overline{\boldsymbol{b}}_0, \overline{\boldsymbol{b}}_1, \cdots, \overline{\boldsymbol{b}}_{k-1}, \overline{\boldsymbol{b}}_k$). They all function as circular-shift registers. Register E also comprises *k* *d*-bit latches which are used to store the outputs of the kernel $d \times d$ systolic Hankel multiplier, i.e. $\overline{\boldsymbol{B}}_0 \otimes \boldsymbol{A}_{k-i}$, $\overline{\boldsymbol{B}}_1 \otimes \boldsymbol{A}_{k-i}$, …, $\overline{\boldsymbol{B}}_{k-1} \otimes \boldsymbol{A}_{k-i}$. The α^d block consists of *d* XOR gates and is applied to perform the α^d transformation for a dual basis vector as described in (23). Register C is then responsible for accumulating and outputting the result vector $\boldsymbol{C}$.

As the computation process begins, register C is initially cleared (step 1.1). Register A is stored the *k* sub-vectors $\boldsymbol{A}_0, \boldsymbol{A}_1, \cdots, \boldsymbol{A}_{k-1}$ which are divided from input vector $\boldsymbol{A}$ (step 1.2). Register B is stored $k+1$ sub-vectors, $\overline{\boldsymbol{b}}_0, \overline{\boldsymbol{b}}_1, \cdots, \overline{\boldsymbol{b}}_{k-1}, \overline{\boldsymbol{b}}_k$ which are divided and generated from input vector $\boldsymbol{B}$ (step 1.3). Next, owing to the frequency of the clock signal CLK1 for registers B and E is set to be *k* times that of the clock signal CLK2 for registers A and C, *k* CLK1 cycles occur during one CLK2 cycle. As a result, the *k* Hankel multiplication results, i.e. $\overline{\boldsymbol{B}}_0 \otimes \boldsymbol{A}_{k-1}$, $\overline{\boldsymbol{B}}_1 \otimes \boldsymbol{A}_{k-1}$, …, $\overline{\boldsymbol{B}}_{k-1} \otimes \boldsymbol{A}_{k-1}$, are sequentially outputted from the kernel Hankel multiplier and stored into register E during one CLK2 cycle (step 2.3-2.7). When the next CLK2 cycle starts, register C has been set to the sum of the α^d transformation of its previous content and the content of register E (step 2.8-2.9). Meanwhile, $\boldsymbol{A}_{k-2}$ is shifted to the top of register A and sent into the kernel Hankel Multiplier; the second computation round begins. After *k* computation rounds completed, the desired result vector $\boldsymbol{C}^T = \overline{\boldsymbol{B}} \otimes \boldsymbol{A}_0 + (\overline{\boldsymbol{B}} \otimes \boldsymbol{A}_1)\alpha^d + \cdots + (\boldsymbol{B} \otimes \boldsymbol{A}_{k-1})\alpha^{(k-1)d}$ (eq. (21)) is then accumulated in register C.

4 Complexity Analysis

4.1 Time and Space Complexities

The proposed MSD scalable DB multiplier architecture contains one kernel $d \times d$ systolic Hankel multiplier (Fig. 2 (a)) which consists of d^2 U-cells. Each U-cell (Fig. 2 (b)) consists of one AND gate, one XOR gate and two 1-bit latches. The summation circuit ($\oplus$) and the α^d block consist of *m* XOR gates and *d* XOR gates, respectively. Additionally, registers A, C and E are all composed of *k* *d*-bit latches, respectively. Registers B then consists of $k+1$ *d*-bit latches. Thus, for the proposed MSD scalable DB multiplier architecture, totally d^2 AND gates, d^2+d+kd XOR gates and $4kd+2d^2+d$ 1-bit latches are required. Table 1 lists these gate counts.

Table 1. Comparisons between various multiplier architectures over GF(2^m)

Multiplier	Ibrahim et al. [14]	Chang et al. [15]	Lee et al. [9]	Proposed (Fig.1)
Basis	Dual	Dual	Dual	Dual
Architecture	Digit-serial	Digit-serial	Bit-parallel	Scalable
Space complexity:				
# 2-input AND gate	$2kd^2$	kd^2	m^2	d^2
# 2-input XOR gate	$2kd^2$	kd^2+kd+d	m^2	d^2+d+kd
# 1-bit latch	$6kd$	$2kd^2+3kd+d-1$	$2m^2+m$	$4kd+2d^2+d$
# 1to2 SW	0	0	m	0
# 2to1 MUX	d	0	m	0
Critical path delay time	$T_A+2dT_X+T_{MUX}+T_L$	$T_A+T_X+T_L$	$T_A+T_X+T_L$	$T_A+T_X+T_L$
Latency (unit = cycle)	$2k$	$kd+d+k-1$	$2m$	k^2+2d-2

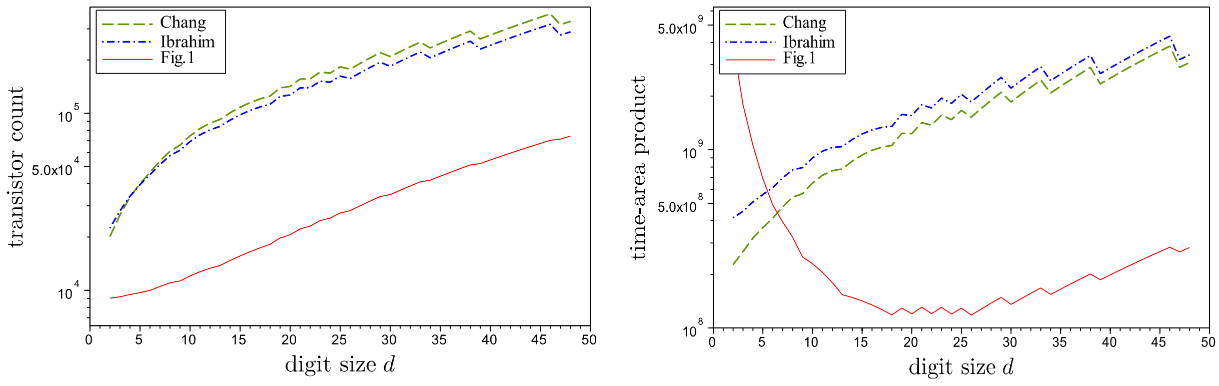
$k = \lceil m/d \rceil$, *d*: the selected digit size

'#' denotes "count of "; SW: switch circuit, MUX: multiplexer

As for the computation latency of the proposed scalable multiplier, Algorithm 1 clearly indicates that k computation rounds and then k Hankel matrix-vector computations in each round are required. Hence, totally k^2 Hankel matrix-vector computations are required to perform a complete m-bit multiplication. Each Hankel matrix-vector computation performed with the kernel $d \times d$ systolic Hankel multiplier (Fig. 2 (a)) requires a latency of $2d-1$ clock cycles (CLK1). Moreover, due to the feedback structure of the summation circuit for register C, the final resultant vector $\boldsymbol{C}$ is obtained after k computation rounds. Each computation round involves k clock cycles to let register E be filled with $\overline{\boldsymbol{B}} \otimes \boldsymbol{A}_{k-i}$ (i.e., $\overline{\boldsymbol{B}}_0 \otimes \boldsymbol{A}_{k-i}$, $\overline{\boldsymbol{B}}_1 \otimes \boldsymbol{A}_{k-i}$, …, $\overline{\boldsymbol{B}}_{k-1} \otimes \boldsymbol{A}_{k-i}$) and then send this content forward to register C. Thus, the total latency required to obtain the desired complete result of vector $\boldsymbol{C}$ is k^2+2d-2 clock cycles. Additionally, the critical path delay is the time duration required by each U-cell in the kernel $d \times d$ systolic Hankel multiplier. This time duration is $T_A+T_X+T_L$ where T_A, T_X and T_L refer to the time delay of a 2-input AND gate, a 2-input XOR gate and an 1-bit latch, respectively. Table 1 summarizes the above space complexity (count of logic gates) and time complexity (latency and critical path delay) of the proposed scalable multiplier architecture where "# 2-input AND gate" means "count of 2-input AND gates". Similarly, "# 2-input XOR gate", "# 1-bit latch", "# 1to2 SW (switch)" and "# 2to1 MUX (multiplexer)" mean respectively the counts of these 4 kinds of logic gates. Table 1 also compares the proposed multiplier architecture with other non-scalable multipliers (bit-parallel [9] and digit-serial ones [14,15]). According to this table, the proposed scalable multiplier architecture has a lower space complexity in terms of total gate count ($O(d^2)$) than the non-scalable architectures ($O(m^2)$ and $O(kd^2)$ for bit-parallel and digit-serial ones, respectively).

4.2 Tradeoff between Throughput and Hardware Complexity

Next, we more closely examine the trade-off between throughput performance and hardware complexity for a given word length m, where the time-area complexity in terms of product of transistor count and latency in ns is calculated. The 2-input XOR gate, 2-input AND gate, 1to2 SW (switch) circuit, 2to1 MUX (multiplexer) and 1-bit latch are composed of 6, 6, 6, 6 and 8 transistors, respectively [21]. Real circuits such as XOR gate M74HC86 ($T_X = 12\text{ns}$), AND gate M74HC08 ($T_A = 7\text{ns}$), Latch gate M74HC279 ($T_L = 13\text{ns}$) and MUX gate M74HC257 ($T_{MUX} = 11\text{ns}$) are utilized to estimate the latency in ns [22]. Thus, according to Table 1, the total transistor count of the proposed multiplier architecture is calculated as $28d^2+38kd+14d$. Its latency in ns, i.e. the product of critical path delay and latency in cycles, is $(7+12+13)(k^2+2d-2)$ ns. In the same manner, the total transistor counts of the digit-serial multipliers of Chang [15] and Ibrahim [14] are calculated as $28kd^2+30kd+14d-8$ and $24kd^2+48kd+6d$, respectively. Their latencies are $32(kd+d+k-1)$ ns and $2k(7+24d+11+13)$ ns, respectively. To illustrate the effectiveness of our proposed scalable architecture in performing large size of m-bit multiplication with substantially smaller d-bit digits in order to achieve adequate security in resource-constrained environments such as embedded systems, the five finite fields adopted by ECDSA, i.e. $\text{GF}(2^{163})$, $\text{GF}(2^{233})$, $\text{GF}(2^{283})$, $\text{GF}(2^{409})$ and $\text{GF}(2^{571})$ are tested. Fig. 3 (a) shows the total transistor counts versus selected digit size d for $\text{GF}(2^{233})$, indicating that the transistor count of the proposed scalable multiplier architecture is always lower than that of the other two digit-serial multipliers. Fig. 3 (b) illustrates the time-area complexity, i.e. the product of total transistor count and latency in ns, versus selected digit size d for $\text{GF}(2^{233})$. According to this figure, the proposed scalable multiplier architecture has a lower time-area complexity than the other two digit-serial multiplier architectures when digit size $d > 6$. A minor limitation is that once d is chosen to be too small (such as <12), the latency increases rapidly. For the other four finite fields, i.e. $\text{GF}(2^{163})$, $\text{GF}(2^{283})$, $\text{GF}(2^{409})$ and $\text{GF}(2^{571})$, similar type of complexity plots can be obtained, e.g. Figs. 4 (a) and (b) are the plots for $\text{GF}(2^{409})$. The superiority of the proposed multiplier architecture over conventional ones is thus clearly demonstrated. Moreover, according to Figs. 3 (b) and 4 (b), the optimal digit size d can be easily selected in practice ($d = 18$ for $\text{GF}(2^{233})$, and 32 for $\text{GF}(2^{409})$, respectively) to achieve the minimal time-area complexity, i.e. the optimal trade-off between throughput performance and hardware complexity for the proposed scalable multiplier architecture. Table 2 lists the optimal digit size *d* of the proposed architecture to achieve the minimal time-area complexity for the five $\mathrm{GF}(2^m)$ adopted by ECDSA. For efforts to implement ECDSA in resource-constrained environments such as smart cards and embedded systems with the proposed scalable multiplier architecture, Table 2 provides a valuable reference.

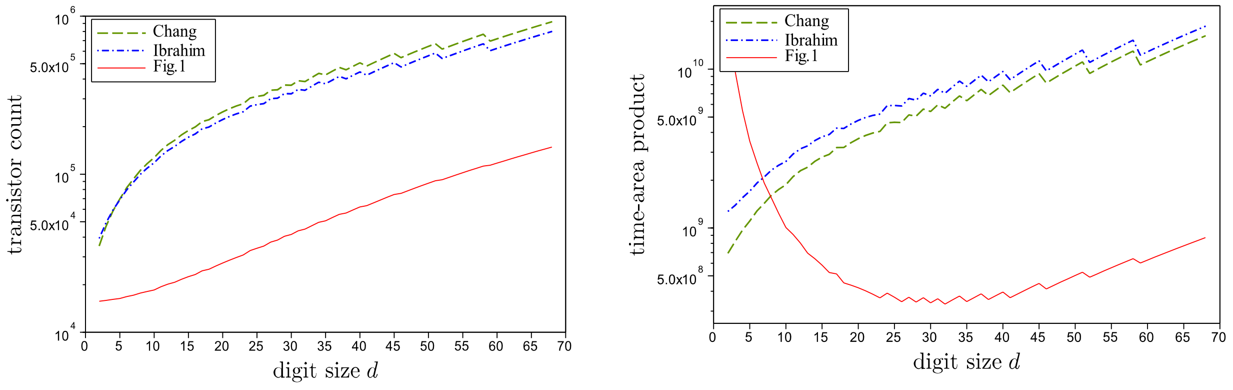


(a) (b)

Fig. 3. (a) Transistor count versus digit size *d* for $\mathrm{GF}(2^{233})$,
(b) Time-area product versus digit size *d* for $\mathrm{GF}(2^{233})$

Table 2. Optimal digit size *d* of the proposed multiplier for $\mathrm{GF}(2^m)$ adopted by ECDSA

m	163	233	283	409	571
d_{opt}	15	18	26	32	36



(a) (b)

Fig. 4. (a) Transistor count versus digit size *d* for $\mathrm{GF}(2^{409})$,
(b) Time-area product versus digit size *d* for $\mathrm{GF}(2^{409})$

5 Conclusions

This work develops a novel low-complexity scalable architecture for dual basis multiplications over $\mathrm{GF}(2^m)$. The proposed multiplier architecture is derived from the Hankel matrix-vector representation of dual basis multiplication. Due to its scalability, this architecture can perform *m*-bit multiplications with substantially smaller *d*-bit digits, and thus is feasible for implementing ECC cryptosystems such as ECDSA in resource-constrained environments such as embedded systems and smart cards. Analytical results exhibit that both area and time-area complexities of the proposed scalable architecture are substantially lower than those of the non-scalable architectures. Besides, this scalable architecture can achieve a satisfactory trade-off between hardware complexity and throughput performance by appropriately selecting its digit size. The proposed architecture is also highly promising for VLSI implementations due to its regularity, modularity and concurrency. As for the future work, we would like to implement our scalable architecture on a real FPGA development board and verify its effectiveness. Furthermore, a real VLSI chip implementation is also an interesting future work.

References

[1] D.E.R. Denning, Cryptography and Data Security, Addison-Wesley Longman Publishing Co., Reading, 1983.

[2] A. Menezes, P.V. Oorschot, S. Vanstone, Handbook of Applied Cryptography, CRC Press, 1996.

[3] M.Y. Rhee, Cryptography and Secure Communications, McGraw-Hill, Singapore, 1994.

[4] N. Kobliz, Elliptic curve cryptography, Math. Computation 48(1987) 203-209.

[5] National Institute for Standards and Technology, Digital Signature Standard, FIPS Publication 186-2, 2000.

[6] I.S. Hsu, T.K. Truong, L.J. Deutsch, I.S. Reed, A comparison of VLSI architecture of finite field multipliers using dual, normal, or standard bases, IEEE Transactions on Computers 37(1988) 735-739.

[7] S.T.J. Fenn, M. Benaissa, D. Taylor, GF(2^m) multiplication and division over the dual basis, IEEE Transactions on Computers 45(1996) 319-327.

[8] C.W. Wu, M.K. Chang, Bit-level systolic arrays for finite-field multiplications, The Journal of VLSI Signal Processing 10(1995) 85-92.

[9] C.Y. Lee, J.S. Horng, I.C. Jou, Low-complexity bit-parallel multiplier over GF(2^m) using dual basis representation, Journal of Computer Science and Technology 21(2006) 887-892.

[10] C.Y. Lee, C.W. Chiou, Efficient design of low-complexity bit-parallel systolic Hankel multipliers to implement multiplication in normal and dual bases of GF(2^m), IEICE Trans Fundamentals E88-A(2005) 3169-3179.

[11] A. Reyhani-Masoleh, M.A. Hasan, Low complexity bit-parallel architectures for polynomial basis multiplication over GF(2^m), IEEE Transactions on Computers 53(2004) 945-959.

[12] C.Y. Lee, Low complexity bit-parallel systolic multiplier over GF(2^m) using irreducible trinomials, IEE Proceedings Computers and Digital Techniques 150(2003) 39-42.

[13] C.H. Kim, C.P. Hong, S. Kwon, A digit-serial multiplier for finite field GF(2^m), IEEE Transactions on Very Large Scale Integration (VLSI) Systems 13(2005) 476-483.

[14] M.K. Ibrahim, A. Aggoun, Dual basis digit serial GF(2^m) multiplier, International Journal of Electronics 89(2002) 517-523.

[15] P.L. Chang, L.H. Chen, C.Y. Lee, Low-complexity dual basis digit serial GF(2^m) multiplier, ICIC Express Letters 3(2009) 1113-1118.

[16] A.F. Tenca, C.K. Koç, A scalable architecture for modular multiplication based on Montgomery's algorithm, IEEE Transactions on Computers 52(2003) 1215-1221.

[17] C.Y. Lee, C.W. Chiou, J.M. Lin, C.C. Chang, Scalable and systolic montgomery multiplier over GF(2^m) generated by trinomials, IET Circuits, Devices & Systems 1(2007) 477-484.

[18] C.Y. Lee, C.W. Chiou, Scalable Gaussian normal basis multipliers over GF(2^m) using Hankel matrix-vector representation, Journal of Signal Processing Systems 69(2012) 197-211.

[19] M. Morii, M. Kasahara, D.L. Whiting, Efficient bit-serial multiplication and the discrete-time wiener-Hopf equation over finite fields, IEEE Transactions on Information Theory 35(1989) 1177-1183.

[20] L.H. Chen, P.L. Chang, C.Y. Lee, Y.K. Yang, Scalable and systolic dual basis multiplier over GF(2^m), International Journal of Innovative Computing, Information and Control 7(2011) 1193-1208.

[21] S.M. Kang, Y. Leblebici, CMOS Digital Integrated Circuits: Analysis and Design, McGraw-Hill Professional, New York, 2002.

[22] STMicroelectronics, Logic Selection Guide. <http://www.st.com>

Appendix A: Effect of Multiplying Dual Basis Vector by α^d

The effect of multiplying a dual basis element vector by α^d is derived as follows: Suppose that there is a dual basis element vector $\boldsymbol{E} \triangleq [e_0, e_1, \cdots, e_{m-1}]$ whose corresponding dual basis element is $E \triangleq e_0\beta_0 + e_1\beta_1 + \cdots + e_{m-1}\beta_{m-1}$. Now, multiplying vector $\boldsymbol{E}$ by α^d, i.e. $\boldsymbol{E}\alpha^d$, implies multiplying element E by α^d, i.e. $E\alpha^d = (e_0\beta_0 + e_1\beta_1 + \cdots + e_{m-1}\beta_{m-1})\alpha^d$. The *j*-th coordinate of $E\alpha^d$ can thus be obtained, based on Definition 1, as

$$(E\alpha^d)_j = Tr(\gamma\alpha^j(E\alpha^d)) = Tr(\gamma\alpha^{d+j}E) = e_{d+j}, \quad \text{for } j = 0,1,\cdots,m-1$$

Thus, we get the vector $\boldsymbol{E}\alpha^d$ as

$$\boldsymbol{E}\alpha^d = [e_d, e_{d+1}, \cdots, e_{m-1}, e_m, \cdots, e_{d+m-1}]$$

From the above equation, it is clear that the effect of multiplying vector $\boldsymbol{E}$ by α^d is just a *d*-bit left-shifting operation on vector $\boldsymbol{E}$. The latter part of coordinates, i.e. the *d* shifted-in bits, $e_m, e_{m+1}, \cdots, e_{m+d-1}$, are defined as $e_{m+i} \triangleq Tr(\gamma\alpha^{m+i}E), \quad i = 0,1,\cdots,d-1$. When the generating function of GF(2^m) is an irreducible trinomial, i.e. $F(x) = x^m + x^n + 1$, these bits can be computed as follows: Let α be the root of $F(x)$, then $\alpha^m = \alpha^n + 1$ which leads to

$$\begin{aligned} e_{m+i} &= Tr(\gamma\alpha^{m+i}E) = Tr(\gamma\alpha^m\alpha^iE) = Tr\left(\gamma(\alpha^n + 1)\alpha^iE\right) \\ &= Tr(\gamma\alpha^{n+i}E) + Tr(\gamma\alpha^iE) = e_{n+i} + e_i, \quad \text{for } i = 0,1,\cdots,d-1 \end{aligned} \qquad \textbf{(A1)}$$

If *n* and *d* are selected to satisfy the condition $0 < n,d \le \lceil m/2 \rceil$, then we get $n + d - 1 \le m - 1$. As a result, from (A1), the lattermost *d* bits of $\boldsymbol{E}\alpha^d$ can be reformed as

$$[e_m, e_{m+1}, \cdots, e_{m+d-1}] = [(e_n + e_0), (e_{n+1} + e_1), \cdots, (e_{n+d-1} + e_{d-1})]$$

Namely, these *d* bits can be generated from the coordinate bits of the original vector $\boldsymbol{E}$, i.e. $e_0, e_1, \cdots, e_{m-1}$, with *d* XOR gates since the addition of two bits in GF(2) can be performed by a XOR gate. The whole $\boldsymbol{E}\alpha^d$ vector can thus be expressed as

$$\boldsymbol{E}\alpha^d = [e_d, e_{d+1}, \cdots, e_{m-1}, \ (e_n + e_0), (e_{n+1} + e_1), \cdots, (e_{n+d-1} + e_{d-1})]$$

Appendix B: Generating $\overline{\boldsymbol{b}}_k$ from $b_0, b_1, \cdots, b_{m-1}$

For sub-vector $\overline{\boldsymbol{b}}_k = [b_m, b_{m+1}, \cdots, b_{m+d-1}]$, when the generating function of GF(2^m) is an irreducible trinomial, i.e. $F(x) = x^m + x^n + 1$, then

$$\begin{aligned} b_{m+i} &= Tr(\gamma\alpha^{m+i}B) = Tr(\gamma\alpha^m\alpha^iB) \\ &= Tr\left(\gamma(\alpha^n + 1)\alpha^iB\right) = Tr(\gamma\alpha^{n+i}B) + Tr(\gamma\alpha^iB) = b_{n+i} + b_i, \quad \text{for } i = 0, 1, \cdots, d-1 \end{aligned}$$

where α is the root of $F(x)$. Thus,

$$\overline{\boldsymbol{b}}_k = [b_m, b_{m+1}, \cdots, b_{m+d-1}] = [(b_n + b_0), (b_{n+1} + b_1), \cdots, (b_{n+d-1} + b_{d-1})]$$

When *n* and *d* are selected as $0 < n,d \le \lceil m/2 \rceil$, we get $n + d - 1 \le m - 1$ which leads to the *d* bits of $\overline{\boldsymbol{b}}_k$ can be generated from $b_0, b_1, \cdots, b_{m-1}$ with *d* XOR gates.