A VLSI Pipeline Design of a Fast Prime Factor DFT on a Finite Field

T. K. Truong, I. S. Hsu, and H. M. Shao
Communications Systems Research Section

I. S. Reed and H. C. Shyu
Department of Electrical Engineering
University of Southern California

In this article, a conventional prime factor discrete Fourier transform (DFT) algorithm is used to realize a discrete Fourier-like transform on the finite field, GF(q^n). A pipeline structure is used to implement this prime factor DFT over GF(q^n). This algorithm is developed to compute cyclic convolutions of complex numbers and to decode Reed-Solomon codes. Such a pipeline fast prime factor DFT algorithm over GF(q^n) is regular, simple, expandable, and naturally suitable for VLSI implementation. An example illustrating the pipeline aspect of a 30-point transform over GF(q^n) is presented.

I. Introduction

Let GF(q^n) be a finite field, where q is a prime number. A discrete Fourier-like transform over GF(q^n) is defined by

\[ A_k = \sum_{n=0}^{d-1} a_n a^{nk} \mod q, \quad 0 \leq k \leq d - 1 \quad (1) \]

where \( A_k, a_n \in GF(q^n) \), \( a \) is the \( d \)-th root of unity in \( GF(q^n) \) and \( d | q^n - 1 \).

In some applications of finite field transforms to coding and digital filters, the transform length \( d \) is not a power of two. Thus, the usual FFT algorithm cannot be used to calculate transforms of as many as \( d \) points.

In this article it will be shown that if \( d = d_1 \cdot d_2 \ldots d_r \), where \( (d_i, d_j) = 1 \) for \( i \neq j \), then the transform in Eq. (1) and its inverse can be calculated by the fast prime factor DFT algorithm (Refs. 1–3). The prime factor DFT algorithm is based on the use of the Chinese remainder theorem (Ref. 4). In this algorithm, the one-dimensional \( d \)-point Fourier transform over GF(q^n) is converted into an \( r \)-dimensional, \( d_i \)-point, \( (i = 1, 2, \ldots , r) \) Fourier transforms over GF(q^n). This algorithm is called a prime factor DFT algorithm over GF(q^n).

A primary advantage of the prime factor DFT is that the VLSI processor for computing a \( d \)-point transform over GF(q^n) is composed only of \( d_1 + d_2 + \ldots + d_r \) basic cells. Each basic cell performs a sum and product operation in GF(q^n).

It is well known that the systolic array of a DFT computation requires \( d \) basic cells (Ref. 5). Therefore, the number of basic cells of a prime factor DFT algorithm over GF(q^n) are
substantially fewer than the systolic array implementation of the DFT. Of course, this advantage must be weighed against the difficulty of the rearrangement of two input and output sequences of \( d \) samples in the VLSI implementation.

In this article, a pipeline structure for rearranging input and output data is developed for a prime factor DFT over \( GF(q^n) \). As a consequence, a pipeline fast prime factor DFT over \( GF(q^n) \) can be developed. Finally, it is shown that a pipeline fast prime factor DFT over \( GF(2^8) \) can be used to decode a \((255,223)\) Reed-Solomon Code (Ref. 6) and that a pipeline fast prime factor DFT over \( GF(q^n) \) can be developed. Finally, it is shown that a pipeline fast prime factor DFT over \( GF(q^n) \) based on the Chinese remainder theorem, is illustrated in the following example.

### II. A Pipeline Fast Prime Factor DFT Over \( GF(q^n) \)

In this section a method for computing a \( d \)-point DFT over \( GF(q^n) \), based on the Chinese remainder theorem, is investigated.

Let \( d = d_1 \cdot d_2 \ldots d_r \), where \( (d_i, d_j) = 1 \) for \( i \neq j \), and also let \( k \) and \( n \) in Eq. (1) be represented by the \( r \)-tuple \((k_1, k_2, \ldots, k_r)\) and \((n_1, n_2, \ldots, n_r)\), respectively, where \( k_j \equiv k \mod d_j \) and \( n_j \equiv n \mod d_j \) for \( 1 \leq j \leq r \). By the use of the Chinese remainder theorem, it is shown (Ref. 4) that a \( d \)-point DFT over \( GF(q^n) \) in Eq. (1) can be decomposed into an \( r \), \( d_i \)-point, DFTs \((i = 1, 2, \ldots, r)\). In other words, the computation of \( A_k \) given in Eq. (1) can be written explicitly in terms of the \( d_i \)-th root of unit in \( GF(q^n) \) as

\[
A_k = A_{k_1,k_2,\ldots,k_r} = \sum_{n_1=0}^{d_1-1} \sum_{n_2=0}^{d_2-1} \cdots \sum_{n_r=0}^{d_r-1} a_{n_1,n_2,\ldots,n_r} \alpha_{n_1}^{k_1} \alpha_{n_2}^{k_2} \cdots \alpha_{n_r}^{k_r} \quad \text{for } 1 \leq k_j \leq d_j
\]  

where

\[
\alpha_j = \alpha^{(d_1,d_2,\ldots,d_{j-1},d_{j+1},\ldots,d_r)} d_j^{-1}
\]

with

\[
(d_1d_2\ldots d_{j-1} d_{j+1} \ldots d_r) \cdot d_j^{-1} \equiv 1 \mod d_j
\]

and \((d_j^{-1}, d_j) = 1\) (for \( j = 1, 2, \ldots, r \)) is a \( d_j \)-th root of unity in \( GF(q^n) \) (Ref. 6).

From Eq. (2), this algorithm consists of the following \( r \) stages:

**Stage 1**

\[
A_{k_1,k_2,\ldots,k_r} = \sum_{n_1=0}^{d_1-1} \sum_{n_2=0}^{d_2-1} \cdots \sum_{n_r=0}^{d_r-1} a_{n_1,n_2,\ldots,n_r} \alpha_{n_1}^{k_1} \alpha_{n_2}^{k_2} \cdots \alpha_{n_r}^{k_r} , \quad 0 \leq k_j \leq d_j - 1
\]  

**Stage 2**

\[
A_{k_1,k_2,\ldots,k_r} = \sum_{i_2=0}^{d_2-1} A_{k_1,i_2}^{d_1} \alpha_{i_2}^{k_1} , \quad 0 \leq k_2 \leq d_2 - 1
\]

**Stage \( r \)**

\[
A_{k_1,k_2,\ldots,k_r} = \sum_{i_r=0}^{d_r-1} A_{k_1,k_2,\ldots,k_{r-1},i_r}^{d_r-1} \alpha_{i_r}^{k_r} , \quad 0 \leq k_r \leq d_r - 1
\]

Instead of computing Eq. (3) directly, an alternate pipeline technique can be developed to compute Eq. (3). To see this, let \( d = d_1 \cdot d_2, \ldots, d_r = d_1^{(1)} \cdot d_2^{(1)} \), where \( d_1^{(1)} = d_1 \cdot d_2, \ldots, d_{r-1}^{(1)} = d_r \) and \( (d_2^{(1)}, d_2^{(2)}) = 1 \). Thus, by Eq. (3) a one-dimensional transform of \( d \) samples can be converted into two \( d_1^{(1)} \)-point transforms, for \( i = 1, 2 \). In a similar fashion, the one-dimensional transform for \( d_1^{(1)} \) can be converted into two \( d_2^{(2)} \)-point transforms, for \( i = 1, 2 \), where \( d_2^{(2)} = d_1 \cdot d_2, \ldots, d_{r-2}^{(2)} = d_{r-1} \), and so forth. This algorithm is called a pipeline prime factor DFT algorithm on \( GF(q^n) \). It is shown in the next section that this new algorithm can be readily implemented in VLSI technology.

### III. VLSI Design for Computing a Pipeline Fast Prime Factor DFT Over \( GF(q^n) \)

A VLSI processor for computing Eq. (1), composed of \( d \) basic cells, is now developed, using the direct approach. Each basic cell performs a sum and product operation in \( GF(q^n) \). That is, a \(-a + b \cdot c\), where "-" denotes the operation "is replaced by." The VLSI architecture of the DFT over \( GF(q^n) \) using a method similar to the systolic array technique is illustrated in the following example.

**Example 1.** Architecture of a 6-point DFT using the direct systolic array technique.
A 6-point DFT over $GF(q^n)$ is defined by

$$A_k = \sum_{n=0}^{6-1} a_n a_n^{nk}$$

where $a$ is a 6th root of unity in $GF(q^n)$. A VLSI architecture structure for computing this transform over $GF(q^n)$ is shown in Fig. 1. This architecture contains six basic cells. The principal function of each basic cell is to perform the operation $a \rightarrow a + b \cdot c$. In this example $a^{nk}$, which is stored in the registers $A_k$ for $0 \leq k \leq 5$, is assumed to be precomputed.

First, the complex symbol sequence $a_n$ for $0 \leq n \leq 5$ is sent to all of the cells, simultaneously. Assume initially that all registers are set to zero. After the input data are entered, the components of $A_k$ in Eq. (1) are contained in registers $B_k$ for $0 \leq k \leq 5$. The values computed in this manner are shifted sequentially out of register $B_0$.

The application of Eq. (3) to the VLSI design of a $d$-point DFT is illustrated in the following example.

**Example 2.** Architecture of a 6-point DFT given in Eq. (1) using Eq. (3).

Since $d = d_1 \cdot d_2 = 3 \cdot 2$, by Eq. (3) one obtains a 6-point DFT as

$$A^1_{(n_1,k_2)} = \sum_{n_2=0}^{2-1} a_{(n_1,n_2)} a_{n_2}^{n_2 k_2}$$  \hspace{1cm} (4a)

$$A_k = A^2_{(k_1,k_2)} = \sum_{n_1=0}^{3-1} A^1_{(n_1,k_2)} a_{n_1}^{n_1 k_1}$$  \hspace{1cm} (4b)

where $a_3 = a^3$ and $a_4 = a^4$ are the 2nd and 3rd roots of unity in $GF(q^n)$.

To compute Eq. (4a), first, one maps the input index $n$ into a pair of indices $n_1$ and $n_2$, where $n_1 = n \mod 3$ and $n_2 = n \mod 2$. That is, $a_0 \rightarrow a_{(0,0)}$, $a_1 \rightarrow a_{(1,1)}$, $a_2 \rightarrow (a_{(2,0)}$, $a_3 \rightarrow (a_{(0,1)}, a_4 \rightarrow a_{(1,0)}$, and $a_5 \rightarrow a_{(2,1)}$. The method used to convert $a_n$ to $a_{(n_1,n_2)}$ is described in Example A.1 in the appendix.

The pipeline structure for computing the 6-point DFT given above in Example 2 is shown in Fig. 2. This pipeline architecture makes possible an efficient design with fewer cells and a small loss in throughput rate.

In this figure, the input sequence $a_n$, first, is shifted sequentially into the memory $A$, which is composed of six registers $A_0$, $A_1$, $\ldots$, and $A_5$. Then, each column of memory $A$ is shifted right out of the last column into the unit of the 2-point DFT. Then each column of $A$ is cyclically shifted up by one symbol, except for the first column.

Simultaneously, a 2-point transform of the first column is computed by the direct systolic array method. These operations are continued repeatedly until the first column of memory $A$ is shifted out of the circuit. The components of $A^1_{(n_1,k_1)}$ are obtained in memory $B$.

Finally, two 3-point transforms of the first and second rows of memory $B$ are calculated by Eq. (4b). The components of $A^2_{(k_1,k_2)}$ are obtained in memory $C$ (see Fig. 2).

The method used to rearrange the output data from $A_{(k_1,k_2)}$ back to $A_k$ is illustrated for Example A.1 in the appendix. Each column of memory $C$ is shifted up and out of the first row and then into the last stage. After this, each row of $C$ is cyclically shifted right by one symbol except for the third row. These operations are continued repeatedly until the last row of memory $C$ is shifted out of the circuit.

The first three samples, $A_0$ through $A_2$, are obtained by switching them to the output line of memory $C$. The next three samples are obtained by switching them to the other output line with a three-stage delay element (see Fig. 2).

If a structure similar to that used in Example 2 is used recursively, the general pipeline fast prime factor DFT algorithm over $GF(q^n)$ can be obtained. This is illustrated by the following example.

**Example 3.** Architecture of a $d = d_1 \cdot d_2 = 5 \times 6$-point DFT over $GF(q^n)$ using Eq. (3)

To compute this 30-point DFT over $GF(q^n)$, first, one computes the two-dimensional $5 \times 6$ DFT over $GF(q^n)$ by

$$A^1_{(n_2,k_2)} = \sum_{n_2=0}^{6-1} a_{(n_1,n_2)} a_{n_2}^{n_2 k_2}$$  \hspace{1cm} (5a)

and

$$A_{(k_1,k_2)} = \sum_{n_1=0}^{5-1} A^1_{(n_1,k_2)} a_{n_1}^{n_1 k_1}$$  \hspace{1cm} (5b)
where \( \alpha_1 = \alpha^6 \) and \( \alpha_2 = \alpha^{25} \) are the 5th and 6th roots of unity in \( GF(q^n) \), respectively.

To compute Eq. (5a), one needs to convert \( a_n \) into \( a_{(n_1,n_2)} \), where \( n_1 \equiv n \mod 5 \) and \( n_2 \equiv n \mod 6 \). The procedure for rearranging this input sequence is given in Example A.2 in the appendix. Then, the 6-point DFT over \( GF(q^n) \) in Eq. (5a) can be decomposed into a two-dimensional \( 3 \times 2 \) transform. This 2-D transform can be implemented by Example 2.

A 5-point DFT over \( GF(q^n) \) in Eq. (5b) can be computed by the direct systolic array method. Finally, the resulting data \( A_{(k_1,k_2)} \) is arranged in the reverse manner back to \( A_k \) (see Example A.2 in the appendix). A pipeline architecture for implementing a prime factor 30-point DFT over \( GF(q^n) \) is shown in Fig. 3.

In Fig. 3, one observes that one needs to permute the index \( n_2 \) of the last column of \( d_2 = 6 \) symbols in matrix \( A \) and the index \( k_2 \) of the first row of \( d_2 = 6 \) symbols in matrix \( C \), where \( n_2 \equiv n \mod 6 \) and \( k_2 \equiv k \mod 6 \). Thus, in order to reduce the number of permutations in the hardware, one should choose \( d_2 < d_1 \), where \( d = d_1 \cdot d_2 \). Then, one needs only to permute the index \( n_2 \) of the sequence of \( d_2 \) symbols.

In Example 3, if one chooses \( d = d_1 \cdot d_2 = 6 \cdot 5 \), then one does not need to permute the index \( n_2 \) of the last column of \( d_2 = 5 \) symbols in a \( 6 \times 5 \) matrix. In general, if \( d_1 - d_2 = 1 \), then no permutations are needed for index \( n_2 \).

IV. Applications

In this section it is shown that a pipeline fast prime factor DFT over \( GF(q^n) \) has application to future designs of coding and digital signal processors.

A. Coding

It is shown (Ref. 6) that a 255-point transform over \( GF(2^8) \) can be used to decode a (255,223) Reed-Solomon Code. Since the transform length of this transform is \( d = 255 = 17 \times 5 \times 3 = 17 \times 15 = d_1 \cdot d_2 \), the transform can be computed by an algorithm similar to that used in Example 2. Thus, a pipeline fast prime factor DFT algorithm over \( GF(2^8) \) can be used to decode a (255,223) Reed-Solomon Code.

The arithmetic needed to compute the 255-point transform over \( GF(2^8) \) in RS decoder requires only \( 3 + 5 + 17 = 25 \) basic cells. Each basic cell performs a sum and product operation over \( GF(2^8) \).

B. Digital Signal Processor

Recently the authors (Ref. 7) defined transforms over \( GF(q^n) \) of the form

\[
A_k \equiv \sum_{n=0}^{d-1} a_n \alpha^{nk} \mod q
\]

where \( q = 2p - 1 \) is a Mersenne prime, \( \alpha \) is a \( d \)th root of unit in \( GF(q^2) \) and \( d|q^2 - 1 \). They showed that the convolution of two finite sequences of \( d \) samples can be obtained as the inverse transform of the product of their transforms.

Nussbaumer in Ref. 8 showed that the complex integer \((1 + i)\) is an \( 8 \cdot p \)th root of unity in \( GF(q^2) \). Multiplications by powers of \((1 + i)\) can be performed by simple bit rotations. As a consequence, the operations needed to compute an \( 8 \cdot p \)-point transform over \( GF(q^2) \) requires only cyclic shifts and additions.

If one uses the direct systolic array method, the VLSI processor needed to compute the above \( 8 \cdot p \)-point transform over \( GF(q^2) \) would require \( 8 \cdot p \) basic cells. Each basic cell of this transform performs only a sum and cyclic shift in \( GF(q^2) \).

Before developing further the pipeline fast prime factor DFT over \( GF(q^2) \), consider further some properties of the finite field, \( GF(q^n) \).

Theorem 1. Let \( GF(q^n) \) be a finite field. Also let \( d = d_1^n \cdot d_2^{n_2} \cdot \ldots \cdot d_r^{n_r} \), where \( d_i \neq 1 \) for \( i \neq j \) and \( d|q^n - 1 \). Then, \( \alpha \) is an element of order \( d \) in \( GF(q^n) \) if and only if \( \alpha^{d/d_i} \neq 1 \mod q \) for \( i = 1, 2, \ldots, r \).

Proof. If \( \alpha \) is an element of order \( d \), then \( d \) is the smallest integer such that

\[
\alpha^d \equiv 1 \mod q
\]

This implies \( \alpha^{d/d_i} \neq 1 \mod q \).

Assume \( \alpha \) is not an element of order \( d \). Then, the order of \( \alpha \) is \( 0(\alpha) \neq d \), where \( 0(\alpha) \) denotes an order of \( \alpha \). Thus, there exists \( h \neq 1 \) such that

\[
d = d_1^n \cdot d_2^{n_2} \cdot \ldots \cdot d_r^{n_r} = h \cdot 0(\alpha)
\]

By (Ref. 6), one observes that \( d_i|h \) for some \( i \). Therefore,

\[
\alpha^{d/d_i} \equiv 1 \mod q
\]
Corollary 1. If \( \alpha \) is an element of order of \( d \) in \( GF(q^n) \), then \( \alpha^j \) is also an element of order \( d \) for \( (j, d) = 1 \).

Proof. Since \( (j, d) = 1 \), then \( d \mid j \), where \( d \) is a prime factor of \( d \). Thus,
\[
(\alpha^j)^d = \alpha^d = (\alpha^j)^d \mod q
\]
where \( \alpha^j = \alpha^d \) is an element of order \( d \) and \( a = d/d_i \).

Assume
\[
\alpha^j \equiv 1 \mod q
\]
Then, \( j \) must be a multiple of \( d_i \). But \( (j, d_i) = 1 \). Hence \( (\alpha^j)^d \not\equiv 1 \mod q \), i.e., \( (\alpha^j)^d \not\equiv 1 \mod q \). By Theorem 1, \( \alpha^j \) is an element of order of \( \alpha \) for \((j, d) = 1 \).

From Corollary 1, one observes that the number of elements of order \( d \) in \( GF(q^n) \) is \( \phi(d) \), where \( \phi(d) \) denotes Euler's Function.

Corollary 2. Let \( GF(q^2) \) be a Galois field, where \( q = 2^p - 1 \) is a Mersenne prime, an \( 8 \cdot p \)th root of unity in \( GF(q^2) \) is \( 1 + i \).

Proof. First note the identity
\[
(1+i)^{8p} \equiv (1+i)^8 \equiv 2^4 \not\equiv 1 \mod q
\]
and
\[
(1+i)^{8p/2} \equiv (1+i)^4 \equiv -2^2 \equiv -2^2 \mod q
\]
Thus, by Theorem 1, it follows that \( (1+i) \) is an \( 8 \cdot p \)th root of unity in \( GF(q^2) \).

Corollary 4. Let \( GF(q^2) \) be a Galois field, where \( q = 2^p - 1 \) is a Mersenne prime. An \( 8 \)th root of unity in \( GF(q^2) \) is in one of forms \( \pm 2^{(p-1)/2}(1 \pm i) \mod q \).

Proof. First note the identity
\[
(2^{(p-1)/2}(1+i))^4 \equiv 2^{2(p-1)}(1+i)^4 \equiv 2^{-2}(2^{-2}) \equiv -1 \mod q
\]
Thus, by Theorem 1, it follows that \( 2^{(p-1)/2}(1+i) \) is a \( 8 \)th root of unity in \( GF(q^2) \). Also, by Corollary 1, \( 2^{(p-1)/2}(1+i) \) for \( j = 3, 5, 7 \), are \( 8 \)th roots of unity in \( GF(q^2) \). That is,
\[
\left(2^{(p-1)/2}(1+i)^j\right)^3 \equiv 2^{(p-1)/2}(1+i)^3 \equiv 2^{(p-1)/2}(1+i)^2 \equiv 2^{(p-1)/2}(1+i)^2 \mod q
\]
and
\[
\left(2^{(p-1)/2}(1+i)^j\right)^5 \equiv 2^{(p-1)/2}(1+i)^5 \equiv 2^{(p-1)/2}(1+i)^4 \equiv 2^{(p-1)/2}(1+i)^4 \mod q
\]
Thus, by Corollary 2, for \( p = 31 \), \( (1+i) \) is a \( 31 \cdot 8 \)th root of unity in \( GF(q^2) \). If we let \( d = d_1 \cdot d_2 = 31 \cdot 8 \), then a \( 31 \cdot 8 \)-point transform over \( GF(q^2) \) is
\[
A_k = \sum_{n=0}^{31-8-1} a_n \alpha^n \quad (7)
\]
where \( \alpha = (1+i) \) is a \( 8 \cdot p \)th root of unity in \( GF(q^2) \).

From Eq. (3), the transform given in Eq. (7) consists of the following two stages:
\[
A_{(n_1,k_2)}^1 \equiv \sum_{n_2=0}^{8-1} a_{(n_1,n_2)} \alpha_2^n \mod q \quad (8a)
\]
and
\[
A_k = \sum_{n_1=0}^{31-1} A_{(n_1,k_2)}^1 \alpha_1^n \mod q \quad (8b)
\]
where $\alpha_2 = (1 + i)^{217} = 2^{15}(1-i)$ and $\alpha_1 = (1 + i)^{32} = 2^{12}$. 

If one uses an algorithm similar to that used in Example 3, a pipeline structure can be used to implement Eqs. (8a) and (8b). For $p = 31$, the VLSI processor for computing an $8 \cdot p$-point transform over $GF(q^2)$ is only composed of $8 + 31 = 39$ basic cells. Each basic cell in this transform performs only a summation and cyclic shifts.

**V. Concluding Remarks**

It has been shown that a conventional prime factor discrete Fourier transform (DFT) algorithm can be used to realize a discrete Fourier-like transform on the field $GF(q^n)$. This algorithm can also be used to compute cyclic convolutions of complex numbers and to decode Reed-Solomon codes. A primary advantage of the prime factor DFT is that the number of VLSI processors needed is substantially fewer than that of the more conventional systolic array implementation. The difficulty of the rearrangement of two input and output sequences of samples in the VLSI implementation was shown to be a soluble problem. As a consequence, a pipeline fast prime factor DFT over $GF(q^n)$ can be developed.

It is possible to use the pipeline fast prime factor DFT over $GF(2^n)$ in decoding a Reed-Solomon code. Furthermore, a pipeline fast prime factor DFT over $GF(q^2)$, where $q = 2^p - 1$ is a Mersenne prime, can be used to compute the cyclic convolutions of two sequences of $8 \cdot p$ symbols.

**Acknowledgment**

The authors wish to thank Dr. L. J. Deutsch of JPL for his helpful suggestions made during the preparation of this article.

**References**


Fig. 1. The systolic array to compute a 6-point DFT using the direct method

Fig. 2. A pipeline architecture for the implementation of a fast prime factor 6-point DFT over GF(q^n)
Fig. 3. A pipeline architecture for the implementation of a fast prime factor 30-point DFT over $GF(q^n)$
Appendix

Algorithm for Rearranging the Input and Output Sequences in the Prime Factor DFT Over GF(q^n)

To perform Eq. (3), one needs to convert input data from \(a_n\) to \(a(n_1,n_2)\) and inversely to convert output data from \(A(k_1,k_2)\) to \(A_n\), where \(0 \leq n, k \leq d = d_1 \cdot d_2 - 1, n_1 \equiv n \mod d_1\) and \(k_1 \equiv k \mod d_1\) for \(i = 1, 2\). To see this, let the one-dimensional input data of DFT be \(a_n\), where \(0 \leq n < d = d_1 \cdot d_2 - 1\). This one-dimensional input data can be shifted sequentially into a \(d_1 \times d_2\) two-dimensional matrix as follows:

\[
\begin{array}{cccc}
  a_{d_1-1} & \ldots & a_2 & a_1 & a_0 \\
  a_{2d_1-1} & \ldots & a_{2d_1+1} & a_1 & a_0 \\
  a_{3d_1-1} & \ldots & a_{3d_1+1} & a_1 & a_0 \\
  \vdots & \vdots & \vdots & \vdots & \vdots \\
  a_{d_2d_1-1} & \ldots & a_{d_2d_1+1} & a_1 & a_0 \\
\end{array}
\]

(A-1)

In (A-1) the input index \(n\) is mapped into the pair of indices \((n_1, n_2)\), where \(n_1 \equiv n \mod d_1\) and \(n_2 \equiv n \mod d_2\). The vector of index \(n_2\) of the last and next to last columns of the matrix in (A-1) are, respectively:

\[
\begin{bmatrix}
  0 \cdot d_1 \mod d_2 & 1 \cdot d_1 \mod d_2 & 2 \cdot d_1 \mod d_2 & \ldots & (d_2 - 1) \cdot d_1 \mod d_2 \\
  d_1 \mod d_2 & \ldots & 1 \cdot d_1 \mod d_2 & 2 \cdot d_1 \mod d_2 & \ldots & (d_2 - 1) \cdot d_1 \mod d_2
\end{bmatrix}^T
\]

(A-2)

and

\[
\begin{bmatrix}
  (0 \cdot d_1 + 1) \mod d_2 & (1 \cdot d_1 + 1) \mod d_2 & (2 \cdot d_1 + 1) \mod d_2 & \ldots & ((d_2 - 1) \cdot d_1 + 1) \mod d_2 \\
  (0 \cdot d_1 + 1) \mod d_2 & \ldots & (1 \cdot d_1 + 1) \mod d_2 & (2 \cdot d_1 + 1) \mod d_2 & \ldots & ((d_2 - 1) \cdot d_1 + 1) \mod d_2
\end{bmatrix}^T
\]

(A-3)

where \(T\) denotes matrix transpose.

Then, one can find 1 in (A-3) and some value \(i\) in (A-2) such that

\[
1 = i \cdot d_1 \mod d_2
\]

(A-4)

Given \(d_1\) and \(d_2\), the solution of congruence (A-4) is \(i\). The index \(n_2\) is periodic with period \(d_2\). If the indices in (A-3) are cyclically shifted up by \(i\) or down by \(d_2 - i\), the index \(n_2\) of the resulting sequence is equal to the index \(n_2\) of the sequence in (A-2). In a similar fashion, if the data of the \(k\)th column of (A-1) is cyclically shifted up or down by \(i\) or by \(d_2 - i\), then the index \(n_2\) of the resulting data of the \(k\)th column is equal to the index \(n_2\) of the data of the \((k + 1)\)th column of the matrix in (A-1). To illustrate this, two examples are presented.

**Example A.1.** Let the length of input sequence be \(d = d_1 \cdot d_2 = 3 \cdot 2\). Convert \(a_n\) into \(a(n_1,n_2)\) and inversely convert \(A(n_1,n_2)\) into \(A_n\), where \(0 \leq n < 5, n_1 \equiv n \mod 3\) and \(n_2 \equiv n \mod 2\). First, the input sequence \(a_n\), \(0 \leq n < 5\), is shifted sequentially into a \(3 \times 2\) matrix as follows:

\[
\begin{array}{ccc}
  a_2 & a_1 & a_0 \\
  a_5 & a_4 & a_3
\end{array}
\]

(A-5)

Then, the index of \(a_n\) is mapped into two pairs of indices \((n_1, n_2)\). Thus, the matrix in (A-5) becomes

\[
\begin{array}{ccc}
  a_{(2,0)} & a_{(1,1)} & a_{(0,0)} \\
  a_{(2,1)} & a_{(1,0)} & a_{(0,1)}
\end{array}
\]

(A-6)

The index \(n_2\) of the last and next to last columns of (A-5) are

\[
\begin{bmatrix}
  0 \cdot 3 \mod 2 & 1 \cdot 3 \mod 2 \\
  0 \cdot 3 \mod 2 & 1 \cdot 3 \mod 2
\end{bmatrix}^T
\]

(A-7)

and

\[
\begin{bmatrix}
  (0 \cdot 3 + 1) \mod 2 & (1 \cdot 3 + 1) \mod 2 \\
  (0 \cdot 3 + 1) \mod 2 & (1 \cdot 3 + 1) \mod 2
\end{bmatrix}^T
\]

(A-8)

respectively.
From (A-4), one obtains the congruence as
\[ 1 \equiv i \cdot 3 \mod 2 \]  
(A-9)

The solution of (A-9) is \( i = 1 \). Let the data of the first and second columns of the 3 \times 2 matrix in (A-6) be cyclically shifted up by two and by one, respectively. Then, the 2-D matrix is arranged in the same order as the index \( n_2 \) of the last column:

\[
\begin{array}{ccc}
  a_{(2,0)} & a_{(1,0)} & a_{(0,0)} \\
  a_{(2,1)} & a_{(1,1)} & a_{(0,1)} \\
\end{array}
\]  
(A-10)

This matrix is used to input data for a prime factor DFT algorithm over \( GF(q^2) \) after taking three 2-point transforms of the columns and then taking two 3-point transforms of the rows in (A-10); then, (A-10) becomes

\[
\begin{array}{ccc}
  A_{(2,0)} & A_{(1,0)} & A_{(0,0)} \\
  A_{(2,1)} & A_{(1,1)} & A_{(0,1)} \\
\end{array}
\]  
(A-11)

To inversely convert 2-D matrix in (A-11) into 1-D \( A_n \) for \( 0 < n \leq 5 \), first, the data of the first and second columns of the 3 \times 2 matrix in (A-11) are cyclically shifted down by two and by one, respectively. Then, the matrix in (A-11) becomes

\[
\begin{array}{ccc}
  A_{(2,0)} & A_{(1,0)} & A_{(0,0)} \\
  A_{(2,1)} & A_{(1,1)} & A_{(0,1)} \\
\end{array}
\]  
(A-12)

\[
\begin{array}{cccccc}
  a_{(4,4)} & a_{(3,3)} & a_{(2,2)} & a_{(1,1)} & a_{(0,0)} \\
  a_{(4,3)} & a_{(3,2)} & a_{(2,1)} & a_{(1,0)} & a_{(0,5)} \\
  a_{(4,2)} & a_{(3,1)} & a_{(2,0)} & a_{(1,5)} & a_{(0,4)} \\
  a_{(4,1)} & a_{(3,0)} & a_{(2,5)} & a_{(1,4)} & a_{(0,3)} \\
  a_{(4,0)} & a_{(3,5)} & a_{(2,4)} & a_{(1,3)} & a_{(0,2)} \\
  a_{(4,5)} & a_{(3,4)} & a_{(2,3)} & a_{(1,2)} & a_{(0,1)} \\
\end{array}
\]  
(A-13)

In order to arrange the second index of a pair \((n_1, n_2)\), i.e., \( n_2 \) in order, from (A-12), one solves the congruence

\[ 1 \equiv i \cdot 5 \mod 6 \]  
(A-14)

The solution of (A-14) is \( i = 5 \) or \( d_2 - i = 6 - 5 = 1 \). If the data of the 1st, 2nd, 3rd, and 4th columns of (A-3) are cyclically shifted down by 4, 3, 2, and 1, the columns of the 5 \times 6 matrix in (A-13) are rearranged in the same order as the last column of (A-13) as follows:

Example A.2. Let the length of input sequence be \( d = d_1 \cdot d_2 = 5 \cdot 6 \). Convert \( a_n \) into \( a_{(n_1, n_2)} \) and inversely convert from output sequence \( A_{(k_1, k_2)} \) into \( A_k \), where \( 0 \leq n, k \leq 30, n_1 \equiv n \mod 5 \), and \( k_i \equiv k \mod 6 \) for \( i = 1, 2 \).
Finally, if one arranges the rows so that the index $n_2$ is in numerical sequence, (A-15) becomes the desired result, namely,

$$\begin{array}{cccc}
a(4,0) & a(3,0) & a(2,0) & a(1,0) \\
a(4,1) & a(3,1) & a(2,1) & a(1,1) \\
a(4,2) & a(3,2) & a(2,2) & a(1,2) \\
a(4,3) & a(3,3) & a(2,3) & a(1,3) \\
a(4,4) & a(3,4) & a(2,4) & a(1,4) \\
\end{array}$$

(A-15)

To rearrange from $A_{(k_1,k_2)}$ in (A-17) into $A_k$, first one arranges the rows in (A-17) such that the index $n_2$ of $5 \times 6$ matrix in (A-17) are in the same order as the index in (A-15) as follows:

$$\begin{array}{cccccc}
A(4,0) & A(3,0) & A(2,0) & A(1,0) & A(0,0) \\
A(4,1) & A(3,1) & A(2,1) & A(1,1) & A(0,1) \\
A(4,2) & A(3,2) & A(2,2) & A(1,2) & A(0,2) \\
A(4,3) & A(3,3) & A(2,3) & A(1,3) & A(0,3) \\
A(4,4) & A(3,4) & A(2,4) & A(1,4) & A(0,4) \\
\end{array}$$

(A-18)

Then, the 1st, 2nd, 3rd, and 4th columns of the matrix are cyclically shifted up by four, three, two, and one data, respectively. Thus, one obtains

$$\begin{array}{cccccc}
A(4,4) & A(3,3) & A(2,2) & A(1,1) & A(0,0) \\
A(4,1) & A(3,1) & A(2,1) & A(1,0) & A(0,5) \\
A(4,2) & A(3,2) & A(2,0) & A(1,5) & A(0,4) \\
A(4,3) & A(3,3) & A(2,5) & A(1,4) & A(0,3) \\
A(4,4) & A(3,4) & A(2,4) & A(1,3) & A(0,2) \\
\end{array}$$

(A-19)

Finally, $A_k$ is shifted sequentially out from this matrix.

$\begin{array}{cccccc}
A(4,0) & A(3,0) & A(2,0) & A(1,0) & A(0,0) \\
A(4,1) & A(3,1) & A(2,1) & A(1,1) & A(0,1) \\
A(4,2) & A(3,2) & A(2,2) & A(1,2) & A(0,2) \\
A(4,3) & A(3,3) & A(2,3) & A(1,3) & A(0,3) \\
A(4,4) & A(3,4) & A(2,4) & A(1,4) & A(0,4) \\
\end{array}$$

(A-17)