D5-32

N86-28280

13149

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

subn

~~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.

subn

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

$$A_k = \sum_{n=0}^{d-1} a_n \alpha^{nk} \bmod q, 0 \leqslant k \leqslant d-1 \qquad (1)$$

where $A_k, a_n \epsilon GF(q^n)$, α is the dth 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, (for $i = 1, 2, \ldots, r$) 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 $d_1 + d_2 + \cdots + d_r$ 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 $GF(q^2)$, 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.

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 $k_j \equiv k \bmod d_j$ and $n_j \equiv n \bmod d_j$ for $1 \leqslant j \leqslant r$. 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_ith root of unit in $GF(q^n)$ as

$$A_k = A_{(k_1,k_2,\ldots,k_r)} = \sum_{n_1=0}^{d_1-1} \sum_{n_2=0}^{d_2-1} \cdots \sum_{n_r=0}^{d_r-1} a_{(n_1,n_2,\ldots,n_r)}$$

$$\alpha_1^{n_1k_1}\,\alpha_2^{n_2k_2} \cdots \alpha_r^{n_rk_r} \text{ for } 1 \leqslant k_j \leqslant d_j \tag{2}$$

where

$$\alpha_j = \alpha^{(d_1d_2\ldots d_{j-1}d_{j+1}\ldots d_r)d_j^{-1}}$$

with

$$(d_1d_2 \ldots d_{j-1}\, d_{j+1} \ldots d_r) \cdot d_j^{-1} \equiv 1 \bmod d_j$$

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

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

Stage 1

$$A^1_{(k_1,i_2,\ldots,i_r)} = \sum_{n_1=0}^{d_1-1} a_{(n_1,n_2,\ldots,n_r)}\,\alpha_1^{n_1k_1}, 0 \leqslant k_1 \leqslant d_1 - 1 \tag{3}$$

Stage 2

$$A^2_{(k_1,k_2,\ldots,i_r)} = \sum_{i_2=0}^{d_2-1} A^1_{(k_1,i_2,\ldots,i_r)}\,\alpha_2^{n_1k_1},\ 0 \leqslant k_2 \leqslant d_2 - 1$$

$$\cdot$$

$$\cdot$$

Stage r

$$A_j = A^r_{(k_1,k_2,\ldots,k_r)}$$

$$= \sum_{i_r=0}^{d_r-1} A^{r-1}_{(k_1,k_2,\ldots,k_{r-1},i_r)}\,\alpha_r^{n_rk_r},\ 0 \leqslant k_r \leqslant d_r - 1$$

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 $d_1^1 = d_1 \cdot d_2, \ldots, d_{r-1}, d_2^{(1)} = d_r$ and $(d_1^{(1)}, d_2^{(2)}) = 1$. Thus, by Eq. (3) a one-dimensional transform of d samples can be converted into a two $d_i^{(1)}$-point transforms, for $i = 1, 2$. In a similar fashion, the one-dimensional transform for $d_1^{(1)}$ can be converted into two $d_i^{(2)}$-point transforms, for $i = 1, 2$, where $d_1^{(2)} = d_1 \cdot d_2, \ldots, d_{r-2}$ and $d_2^{(2)} = d_{r-1}$, 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, $a \leftarrow a + b \cdot c$, 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

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

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 $a \leftarrow a + b \cdot c$. In this example α^{nk}, which is stored in the registers A_k for $0 \leqslant k \leqslant 5$, is assumed to be precomputed.

First, the complex symbol sequence a_n for $0 \leqslant n \leqslant 5$ 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 $0 \leqslant k \leqslant 5$. 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

$$A^1_{(n_1,k_2)} = \sum_{n_2=0}^{2-1} a_{(n_1,n_2)} \alpha_2^{n_2k_2} \tag{4a}$$

$$A_k = A^2_{(k_1,k_2)} = \sum_{n_1=0}^{3-1} A^1_{(n_1,k_2)} \alpha_1^{n_1k_1} \tag{4b}$$

where $\alpha_2 = \alpha^3$ and $\alpha_1 = \alpha^4$ 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 $n_1 \equiv n \bmod 3$ and $n_2 \equiv n \bmod 2$. That is, $a_0 \to a_{(0,0)}$, $a_1 \to a_{(1,1)}$, $a_2 \to (a_{2,0})$, $a_3 \to (a_{0,1})$, $a_4 \to a_{(1,0)}$, and $a_5 \to a_{(2,1)}$. 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 $A_{(k_1,k_2)}$ 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 X 6 DFT over $GF(q^n)$ by

$$A^1_{(n_2,k_2)} = \sum_{n_2=0}^{6-1} a_{(n_1,n_2)} \alpha_2^{n_2k_2} \tag{5a}$$

and

$$A_{(k_1,k_2)} = \sum_{n_1=0}^{5-1} A^1_{(n_1,\, k_2)} \alpha_1^{n_1k_1} \tag{5b}$$

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 $n_1 \equiv n \bmod 5$ and $n_2 \equiv n \bmod 6$. 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 X 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 $n_2 \equiv n \bmod 6$ and $k_2 \equiv k \bmod 6$. 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 X 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 pipeline prime 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 (255,233) 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 $GF(q^2)$ of the form

$$A_k \equiv \sum_{n=0}^{d-1} a_n \alpha^{nk} \bmod q$$

where $q = 2^p - 1$ is a Mersenne prime, α is a dth 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 $d = d_1^{n_1} \cdot d_2^{n_2}, \ldots, d_r^{n_r}$, where $(d_i, d_j) \neq 1$ for $i \neq j$ and $d | q^n - 1$. Then, α is an element of order d in $GF(q^n)$ if and only if $\alpha^{d/d_i} \not\equiv 1 \bmod q$ for $i = 1, 2, \ldots, r$.

Proof. If α is an element of order d, then d is the smallest integer such that

$$\alpha^d \equiv 1 \bmod q$$

This implies $\alpha^{d/d_i} \not\equiv 1 \bmod q$.

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

$$d = d_1^{n_1} \cdot d_2^{n_2}, \ldots, d_r^{n_r} = h \cdot O(\alpha) \tag{6}$$

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

$$\alpha^{d/d_i} \equiv 1 \bmod q$$

Corollary 1. If α is an element of order of d in $GF(q^n)$, then α^j is also an element of order d for $(j, d) = 1$.

Proof. Since $(j, d) = 1$, then $d_i \nmid j$, where d_i is a prime factor of d. Thus,

$$(\alpha^j)^{d/d_i} \equiv (\alpha^{d/d_i})^j \equiv (\alpha_i)^j \bmod q$$

where $\alpha_i = \alpha^a$ is an element of order d_i and $a = d/d_i$.

Assume

$$\alpha_i^j \equiv 1 \bmod q$$

Then, j must be a multiple of d_i. But $(j, d_i) = 1$. Hence $(\alpha^a)^j \not\equiv 1 \bmod q$, i.e., $(\alpha^j)^{d/d_i} \not\equiv 1 \bmod q$. By Theorem 1, α^j is an element of order of α 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 Golois 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

$$(1+i)^{8p/p} \equiv (1+i)^8 \equiv 2^4 \not\equiv 1 \bmod q$$

and

$$(1+i)^{8p/2} \equiv (1+i)^{4p} \equiv (-2^2)^p \equiv -2^{2p}$$

$$\equiv -1 \not\equiv 1 \bmod q$$

Thus, by Theorem 1, it follows that $(1 + i)$ is an $8 \cdot p$th root of unity in $GF(q^2)$.

Corollary 3. Let $GF(q^2)$ be a Golois field, where $q = 2^p - 1$ is a Mersenne prime, 2^j is a pth root of unity in $GF(q^2)$ for $j = 1, 2, \ldots, p-1$.

Proof. First note the identity

$$2^{p/p} \equiv 2 \not\equiv 1 \bmod q$$

Thus, by Theorem 1, it follows that 2 is a pth root of unity in $GF(q^2)$. Also by Corollary 1, 2^j is also a pth root of unity in $GF(q^2)$ for $(j, p) = 1$, i.e., $j = 1, 2, \ldots, p-1$.

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

Proof. First note the identity

$$(2^{(p-1)/2}(1+i))^4 \equiv 2^{2(p-1)}(1+i)^4 \equiv 2^{-2}(-2^2) \equiv -1 \bmod q$$

Thus, by Theorem 1, it follows that $2^{(p-1)/2}(1 + i)$ is an 8th root of unity in $GF(q^2)$. Also, by Corollary 1, $(2^{(p-1)/2}(1 + i))^j$ for $j = 3, 5, 7$, are 8th roots of unity in $GF(q^2)$. That is,

$$\left(2^{(p-1)/2}(1+i)\right)^3 \equiv 2^{(p-1)/2}(1+i)\,2^{p-1}(1+i)^2$$

$$\equiv 2^{(p-1)/2}(-1+i) \bmod q$$

$$\left(2^{(p-1)/2}(1+i)\right)^5 \equiv 2^{(p-1)/2}(1+i)\left(2^{(p-1)/2}(1+i)\right)^4$$

$$\equiv -2^{(p-1)/2}(1+i) \bmod q$$

and

$$\left(2^{(p-1)/2}(1+i)\right)^7 \equiv 2^{(p-1)/2}(1+i)\left(2^{(p-1)/2}(1+i)\right)^6$$

$$\equiv -2^{(p-1)/2}(-1+i)$$

Therefore, an 8th root of unity in $GF(q^2)$ is in one of forms $\pm 2^{(p-1)/2}(1 \pm i) \bmod q$.

By Corollary 2, for $p = 31$, $(1 + i)$ is a $31 \cdot 8$th 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

$$A_k = \sum_{n=0}^{31 \cdot 8 - 1} a_n \alpha^{nk} \tag{7}$$

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:

$$A^1_{(n_1,k_2)} \equiv \sum_{n_2=0}^{8-1} a_{(n_1,n_2)} \alpha_2^{n_2 k_2} \bmod q \tag{8a}$$

and

$$A_k = A^2_{(k_1,k_2)} \equiv \sum_{n_1=0}^{31-1} A^1_{(n_1,k_2)} \alpha_1^{n_1 k_1} \bmod q \tag{8b}$$

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 $GF(2^8)$ 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

1. Good, I. J., "The Interaction Algorithm and Practical Fourier Analysis," *J. Royal Statist. Soc.*, Ser. B, Vol. 20, pp. 361–372, 1958, "Addendum," Vol. 22 (MR 21 1674; MR 23 A4231), pp. 372–375, 1960.

2. Thomas, L. H., "Using A Computer to Solve Problems in Physics," in *Applications of Digital Computers*, Gion and Co., Boston, Mass., 1963.

3. Kolba, D. P., and Parks, T. W., "A Prime Factor FFT Algorithm Using High-Speed Convolution," *IEEE Trans. on Acoustics, Speed, and Signal Processing*, Vol. ASSP-25, No. 4, pp. 281–294, August 1977.

4. McClellan, J., and Rader, C., "Number Theory in Digital Signal Processing," Prentice-Hall, Englewood Cliffs, N.J., 1979.

5. Mead, C., and Conway, L., *Introduction to VLBI Systems*, Chapter 8, Addison-Wesley Publishing Co., Reading, Mass., 1980.

6. Reed, I. S., Troung, T. K., Miller, R. L., and Huang, J. P., "Fast Transforms for Decoding Reed-Solomon Codes," *IEE Proc.*, Vol. 128, Pt. F, No. 1, February 1981.

7. Reed, I. S., and Truong, T. K., "The Use of Finite Fields to Compute Convolutions," *IEEE Trans. on Information Theory*, Vol. IT-21, No. 2, pp. 208–213, March 1975.

8. Nassbaumer, H. J., "Digital Filtering Using Complex Mersenne Transforms," *IBM Journal of Research and Development*, Vol. 20, No. 5, pp. 498–504, September 1976.

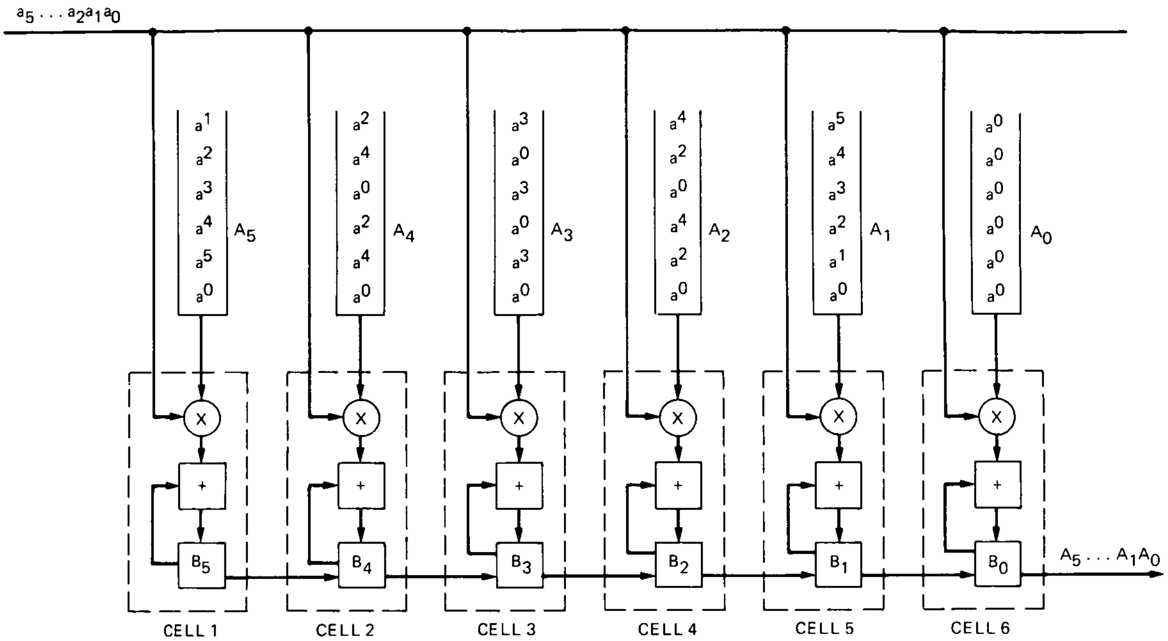
$a_5 \ldots a_2a_1a_0$
a^1 a^2 a^3 a^4 a^5 a^0 A_5
a^2 a^4 a^0 a^2 a^4 a^0 A_4
a^3 a^0 a^3 a^0 a^3 a^0 A_3
a^4 a^2 a^0 a^4 a^2 a^0 A_2
a^5 a^4 a^3 a^2 a^1 a^0 A_1
a^0 a^0 a^0 a^0 a^0 a^0 A_0
X
+
B_5
B_4
B_3
B_2
B_1
B_0
$A_5 \ldots A_1A_0$
CELL 1
CELL 2
CELL 3
CELL 4
CELL 5
CELL 6

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

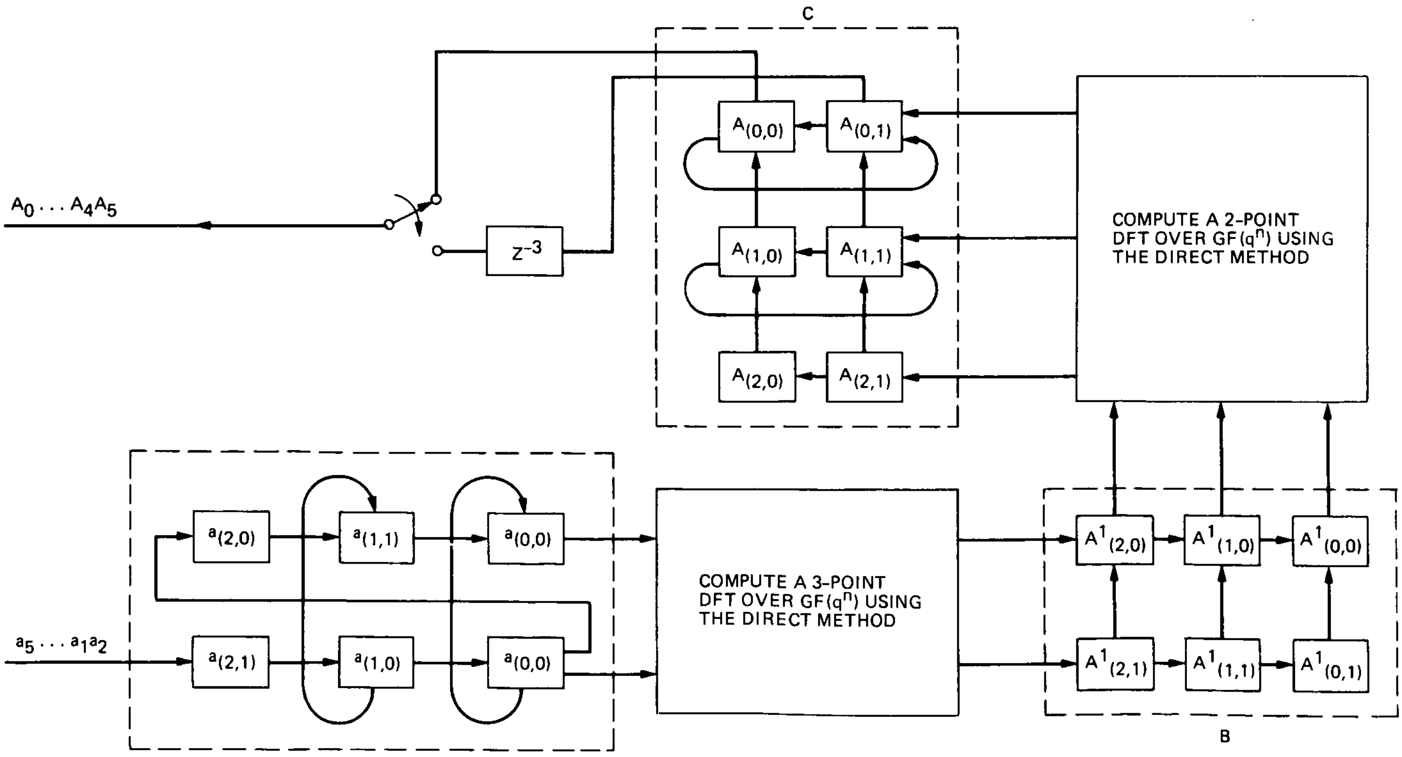

Fig. 2. A pipeline architecture for the implementation of a fast prime factor 6-point DFT over *GF*(q^n)

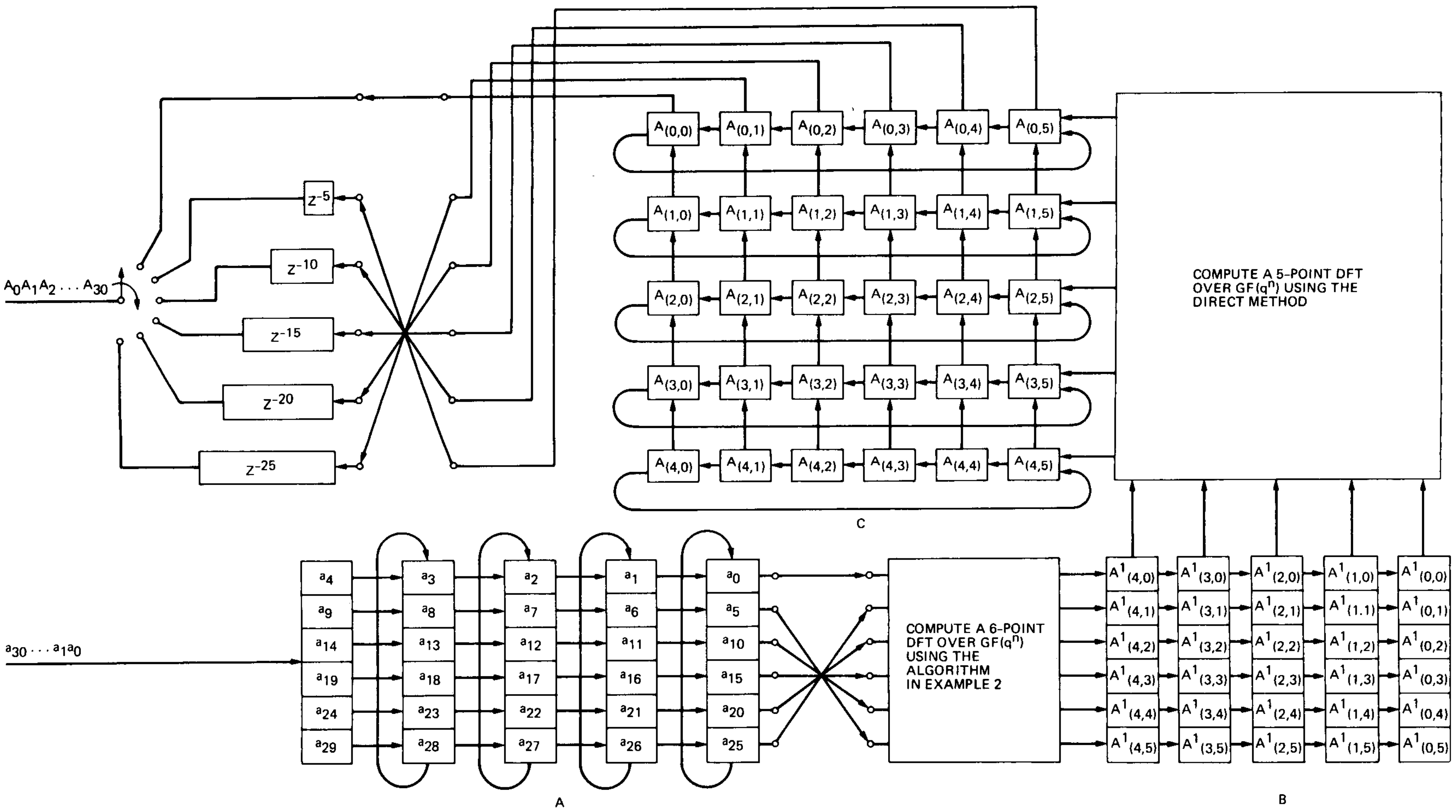
$A_0A_1A_2 \cdots A_{30}$
z^{-5}
z^{-10}
z^{-15}
z^{-20}
z^{-25}
$A_{(0,0)}$ $A_{(0,1)}$ $A_{(0,2)}$ $A_{(0,3)}$ $A_{(0,4)}$ $A_{(0,5)}$
$A_{(1,0)}$ $A_{(1,1)}$ $A_{(1,2)}$ $A_{(1,3)}$ $A_{(1,4)}$ $A_{(1,5)}$
$A_{(2,0)}$ $A_{(2,1)}$ $A_{(2,2)}$ $A_{(2,3)}$ $A_{(2,4)}$ $A_{(2,5)}$
$A_{(3,0)}$ $A_{(3,1)}$ $A_{(3,2)}$ $A_{(3,3)}$ $A_{(3,4)}$ $A_{(3,5)}$
$A_{(4,0)}$ $A_{(4,1)}$ $A_{(4,2)}$ $A_{(4,3)}$ $A_{(4,4)}$ $A_{(4,5)}$
COMPUTE A 5-POINT DFT OVER GF(q^n) USING THE DIRECT METHOD
C
$a_{30} \cdots a_1a_0$
a_4 a_9 a_{14} a_{19} a_{24} a_{29}
a_3 a_8 a_{13} a_{18} a_{23} a_{28}
a_2 a_7 a_{12} a_{17} a_{22} a_{27}
a_1 a_6 a_{11} a_{16} a_{21} a_{26}
a_0 a_5 a_{10} a_{15} a_{20} a_{25}
COMPUTE A 6-POINT DFT OVER GF(q^n) USING THE ALGORITHM IN EXAMPLE 2
$A^1{}_{(4,0)}$ $A^1{}_{(4,1)}$ $A^1{}_{(4,2)}$ $A^1{}_{(4,3)}$ $A^1{}_{(4,4)}$ $A^1{}_{(4,5)}$
$A^1{}_{(3,0)}$ $A^1{}_{(3,1)}$ $A^1{}_{(3,2)}$ $A^1{}_{(3,3)}$ $A^1{}_{(3,4)}$ $A^1{}_{(3,5)}$
$A^1{}_{(2,0)}$ $A^1{}_{(2,1)}$ $A^1{}_{(2,2)}$ $A^1{}_{(2,3)}$ $A^1{}_{(2,4)}$ $A^1{}_{(2,5)}$
$A^1{}_{(1,0)}$ $A^1{}_{(1.1)}$ $A^1{}_{(1,2)}$ $A^1{}_{(1,3)}$ $A^1{}_{(1,4)}$ $A^1{}_{(1,5)}$
$A^1{}_{(0,0)}$ $A^1{}_{(0,1)}$ $A^1{}_{(0,2)}$ $A^1{}_{(0,3)}$ $A^1{}_{(0,4)}$ $A^1{}_{(0,5)}$
A
B

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_k, where $0 \leqslant n, k \leqslant d = d_1 \cdot d_2 - 1$, $n_i \equiv n \bmod d_i$ and $k_i \equiv k \bmod d_i$ for $i = 1, 2$. To see this, let the one-dimensional input data of DFT be a_n, where $0 \leqslant n \leqslant d = d_1 \cdot d_2 - 1$. This one-dimensional input data can be shifted sequentially into a $d_1 \times d_2$ two-dimensional matrix as follows:

$$\begin{matrix} a_{d_1-1} & \ldots, a_2 & a_1 & a_0 \\ a_{2d_1-1} & \ldots, a_{2d_1+2} & a_{d_1+1} & a_{1d_1} \\ a_{3d_1-1} & \ldots, a_{3d_1+2} & a_{2d_1+1} & a_{2d_1} \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ a_{d_2d_1-1} & \ldots, a_{(d_2-1)d_1+2} & a_{(d_2-1)d_1+1} & a_{(d_2-1)d_1} \end{matrix} \tag{A-1}$$

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

$$[0 \cdot d_1 \bmod d_2, 1 \cdot d_1 \bmod d_2, 2 \cdot d_1 \bmod d_2, \ldots, (d_2 - 1) \cdot d_1 \bmod d_2]^T \tag{A-2}$$

and

$$[(0 \cdot d_1 + 1) \bmod d_2, (1 \cdot d_1 + 1) \bmod d_2, (2 \cdot d_1 + 1) \bmod d_2, \ldots, ((d_2 - 1) \cdot d_1 + 1) \bmod d_2]^T \tag{A-3}$$

where T denotes matrix transpose.

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

$$1 \equiv i \cdot d_1 \bmod d_2 \tag{A-4}$$

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 kth 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 kth 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 $0 \leqslant n \leqslant 5$, $n_1 \equiv n \bmod 3$ and $n_2 \equiv n \bmod 2$. First, the input sequence a_n, $0 \leqslant n \leqslant 5$, is shifted sequentially into a 3×2 matrix as follows:

a_2	a_1	a_0
a_5	a_4	q_3

(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

$a_{(2,0)}$	$a_{(1,1)}$	$a_{(0,0)}$
$a_{(2,1)}$	$a_{(1,0)}$	$a_{(0,1)}$

(A-6)

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

$$[0 \cdot 3 \bmod 2, 1 \cdot 3 \bmod 2]^T \tag{A-7}$$

and

$$[(0 \cdot 3 + 1) \bmod 2, (1 \cdot 3 + 1) \bmod 2]^T \tag{A-8}$$

respectively.

From (A-4), one obtains the congruence as

$$1 \equiv i \cdot 3 \bmod 2 \tag{A-9}$$

The solution of (A-9) is $i = 1$. Let the data of the first and second columns of the 3 X 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:

$a_{(2,0)}$	$a_{(1,0)}$	$a_{(0,0)}$
$a_{(2,1)}$	$a_{(1,1)}$	$a_{(0,1)}$

(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

$A_{(2,0)}$	$A_{(1,0)}$	$A_{(0,0)}$
$A_{(2,1)}$	$A_{(1,1)}$	$A_{(0,1)}$

(A-11)

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

$A_{(2,0)}$	$A_{(1,1)}$	$A_{(0,0)}$
$A_{(2,1)}$	$A_{(1,0)}$	$A_{(0,1)}$

or

A_2	A_1	A_0
A_5	A_4	A_3

Finally, a_n is shifted out sequentially from this matrix.

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 $0 \leqslant n, k \leqslant 30$, $n_i \equiv n \bmod 5$, and $k_i \equiv k \bmod 6$ for $i = 1, 2$.

First, the input sequence a_n, $0 \leqslant n \leqslant 30$ is shifted sequentially into a 5 X 6 matrix as follows:

a_4	a_3	a_2	a_1	a_0
a_9	a_8	a_7	a_6	a_5
a_{14}	a_{13}	a_{12}	a_{11}	a_{10}
a_{19}	a_{18}	a_{17}	a_{16}	a_{15}
a_{24}	a_{23}	a_{22}	a_{21}	a_{20}
a_{29}	a_{28}	a_{27}	a_{26}	a_{25}

(A-12)

If the index of a_n is mapped into the pair of indices (n_1, n_2), where $n_1 \equiv n \bmod 5$ and $n_2 \equiv n \bmod 6$, then the matrix in (A-12) becomes

$a_{(4,4)}$	$a_{(3,3)}$	$a_{(2,2)}$	$a_{(1,1)}$	$a_{(0,0)}$
$a_{(4,3)}$	$a_{(3,2)}$	$a_{(2,1)}$	$a_{(1,0)}$	$a_{(0,5)}$
$a_{(4,2)}$	$a_{(3,1)}$	$a_{(2,0)}$	$a_{(1,5)}$	$a_{(0,4)}$
$a_{(4,1)}$	$a_{(3,0)}$	$a_{(2,5)}$	$a_{(1,4)}$	$a_{(0,3)}$
$a_{(4,0)}$	$a_{(3,5)}$	$a_{(2,4)}$	$a_{(1,3)}$	$a_{(0,2)}$
$a_{(4,5)}$	$a_{(3,4)}$	$a_{(2,3)}$	$a_{(1,2)}$	$a_{(0,1)}$

(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

$$1 \equiv i \cdot 5 \bmod 6 \tag{A-14}$$

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 X 6 matrix in (A-13) are rearranged in the same order as the last column of (A-13) as follows:

$a_{(4,0)}$	$a_{(3,0)}$	$a_{(2,0)}$	$a_{(1,0)}$	$a_{(0,0)}$
$a_{(4,5)}$	$a_{(3,5)}$	$a_{(2,5)}$	$a_{(1,5)}$	$a_{(0,5)}$
$a_{(4,4)}$	$a_{(3,4)}$	$a_{(2,4)}$	$a_{(1,4)}$	$a_{(0,4)}$
$a_{(4,3)}$	$a_{(3,3)}$	$a_{(2,3)}$	$a_{(1,3)}$	$a_{(0,3)}$
$a_{(4,2)}$	$a_{(3,2)}$	$a_{(2,2)}$	$a_{(1,2)}$	$a_{(0,2)}$
$a_{(4,1)}$	$a_{(3,1)}$	$a_{(2,1)}$	$a_{(1,1)}$	$a_{(0,1)}$

(A-15)

Finally, if one arranges the rows so that the index n_2 is in numerical sequence,(A-15) becomes the desired result, namely,

$a_{(4,0)}$	$a_{(3,0)}$	$a_{(2,0)}$	$a_{(1,0)}$	$a_{(0,0)}$
$a_{(4,1)}$	$a_{(3,1)}$	$a_{(2,1)}$	$a_{(1,1)}$	$a_{(0,1)}$
$a_{(4,2)}$	$a_{(3,2)}$	$a_{(2,2)}$	$a_{(1,2)}$	$a_{(0,2)}$
$a_{(4,3)}$	$a_{(3,3)}$	$a_{(2,3)}$	$a_{(1,3)}$	$a_{(0,3)}$
$a_{(4,4)}$	$a_{(3,4)}$	$a_{(2,4)}$	$a_{(1,4)}$	$a_{(0,4)}$
$a_{(4,5)}$	$a_{(3,5)}$	$a_{(2,5)}$	$a_{(1,5)}$	$a_{(0,5)}$

(A-16)

After taking five 6-point transforms of the columns and then taking six 5-point transforms of the rows in (A-16), (A-16) becomes

$A_{(4,0)}$	$A_{(3,0)}$	$A_{(2,0)}$	$A_{(1,0)}$	$A_{(0,0)}$
$A_{(4,1)}$	$A_{(3,1)}$	$A_{(2,1)}$	$A_{(1,1)}$	$A_{(0,1)}$
$A_{(4,2)}$	$A_{(3,2)}$	$A_{(2,2)}$	$A_{(1,2)}$	$A_{(0,2)}$
$A_{(4,3)}$	$A_{(3,3)}$	$A_{(2,3)}$	$A_{(1,3)}$	$A_{(0,3)}$
$A_{(4,4)}$	$A_{(3,4)}$	$A_{(2,4)}$	$A_{(1,4)}$	$A_{(0,4)}$
$A_{(4,5)}$	$A_{(3,5)}$	$A_{(2,5)}$	$A_{(1,5)}$	$A_{(0,5)}$

(A-17)

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 × 6 matrix in (A-17) are in the same order as the index in (A-15) as follows:

$A_{(4,0)}$	$A_{(3,0)}$	$A_{(2,0)}$	$A_{(1,0)}$	$A_{(0,0)}$
$A_{(4,5)}$	$A_{(3,5)}$	$A_{(2,5)}$	$A_{(1,5)}$	$A_{(0,5)}$
$A_{(4,4)}$	$A_{(3,4)}$	$A_{(2,4)}$	$A_{(1,4)}$	$A_{(0,4)}$
$A_{(4,3)}$	$A_{(3,3)}$	$A_{(2,3)}$	$A_{(1,3)}$	$A_{(0,3)}$
$A_{(4,2)}$	$A_{(3,2)}$	$A_{(2,2)}$	$A_{(1,2)}$	$A_{(0,2)}$
$A_{(4,1)}$	$A_{(3,1)}$	$A_{(2,1)}$	$A_{(1,1)}$	$A_{(0,1)}$

(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

$A_{(4,4)}$	$A_{(3,3)}$	$A_{(2,2)}$	$A_{(1,1)}$	$A_{(0,0)}$
$A_{(4,3)}$	$A_{(3,2)}$	$A_{(2,1)}$	$A_{(1,0)}$	$A_{(0,5)}$
$A_{(4,2)}$	$A_{(3,1)}$	$A_{(2,0)}$	$A_{(1,5)}$	$A_{(0,4)}$
$A_{(4,1)}$	$A_{(3,0)}$	$A_{(2,5)}$	$A_{(1,4)}$	$A_{(0,3)}$
$A_{(4,0)}$	$A_{(3,5)}$	$A_{(2,4)}$	$A_{(1,3)}$	$A_{(0,2)}$
$A_{(4,5)}$	$A_{(3,4)}$	$A_{(2,3)}$	$A_{(1,2)}$	$A_{(0,1)}$

(A-19)

Finally, A_k is shifted sequentially out from this matrix.