The Integer Transforms Analogous to Discrete Trigonometric Transforms

Soo-Chang Pei, *Fellow, IEEE*, and Jian-Jiun Ding

***Abstract*—Integer transform (such as the Walsh transform) is the discrete transform that all the entries of the transform matrix are integer. It is much easy to implement because the real number multiplication operations can be avoided, but the performance is usually worse. On the other hand, the noninteger transform, such as the DFT and DCT, has good performance, but the real number multiplication is required. In this paper, we will try to derive the integer transforms analogous to some popular noninteger transforms. These integer transforms remain most of the performance quality of the original transform, but the implementation will be much simpler. Especially, for two-dimensional (2-D) block transform in image/video, the saving can be huge for using integer operations. In 1989, Cham had derived the integer cosine transform. Here, we will derive the integer sine, Hartley, and Fourier transforms. We also introduce the general method to derive the integer transform from some noninteger transform. Besides, the integer transform derived by Cham still requires real number multiplication for the inverse transform. We will modify the integer transform introduced by Cham and introduce the complete integer transform. It requires no real number multiplication operation, no matter what the forward or inverse transform. The integer transform we derive would be more efficient than the original transform. For example, for the 8-point DFT and IDFT, there are in total four real numbers and eight fixed-point multiplication operations required, but for the forward and inverse 8-point complete integer Fourier transforms, there are totally 20 fixed-point multiplication operations required. However, for the integer transform, the implementation is simpler, and many of the properties of the original transform will be kept.**

***Index Terms*—Integer cosine transform, integer Fourier transform, integer Hartley transform, integer sine transform, integer transform.**

I. Introduction

SUPPOSE there is a linear discrete transform with the transform matrix $\mathbf{A}$:

$$Y(m) = \sum_{n=0}^{N-1} A(m,n) \cdot X(n). \tag{1}$$

If the values of $A(m,n)$ can all be expressed as

$$A(m,n) = D \cdot 2^{-h} \quad \text{where } D, h \text{ are integer} \tag{2}$$

then we call this discrete transform the *integer transform*. For example, the Walsh transform and Hadamard transform are all integer transforms.

Manuscript received October 12, 1999; revised August 18, 2000. The associate editor coordinating the review of this paper and approving it for publication was Dr. Xiang-Gen Xia.

The authors are with the Department of Electrical Engineering, National Taiwan University, Taipei, Taiwan, R.O.C. (e-mail: pei@cc.ee.ntu.edu.tw).

Publisher Item Identifier S 1053-587X(00)10135-7.

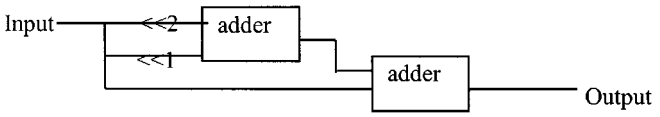

Fig. 1. Implementation of the multiplication of 7 (≪: binary shifting operation).

The most important advantage of the integer transform is that real number multiplication operations are required to implement it. We find that if the transform matrix has some entries that are not in the form of (2), then the real number multiplication operations are required to implement it. The implementation of real number multiplication is usually more complicated and time consuming. In contrast, if all the entries in the transform matrix can be expressed as the form of (2), we only require the fixed-point multiplication operations to implement it. We can implement the fixed-point multiplication operations by the addition and binary shifting operations. For example, if we want to implement the multiplication of 7, we can implement it as Fig. 1.

In addition, for the multiplication of the number not satisfying (2), we cannot implement it by the method of Fig. 1. Besides, for the computer, the multiplication of the noninteger number requires the real number processor and will be more complex and expansive. Thus, if we consider efficiency of hardware implementation and the computation by computer, the integer transform would be better than the noninteger transform.

However, if we consider performance, the noninteger transforms are usually better than the integer transforms. For example, the discrete Fourier transform (DFT) can do the frequency domain analysis, and the discrete cosine transform (DCT) has the high ability of data decorrelation [6]. Thus, although they all require the real number multiplication, they are widely applied.

In this paper, we try to find the integer transforms analogous to some popular noninteger discrete transform, such as the DFT, discrete Hartley transform (DHT), discrete sine transform (DST), and DCT. In [1]–[4], they have derived some types of integer cosine and sine transforms. In this paper, we will improve their methods to derive the integer Fourier and Hartley transforms and other types of integer cosine and sine transforms. Because these integer transforms will approximate to the original transforms, their performance will be very similar to the original transforms, but the real number multiplication can be saved.

Our goal for deriving the integer transforms is not approximating the original transforms. We just want the properties of the original transform to be kept. Then, we can use the transform with much simpler implementation to do the applications

of the original transform. We hope the integer transforms we derive can replace the original transforms in many applications.

In Section II, we will introduce the general method to derive the integer transform from a noninteger transform. We will introduce two types of integer transform. One is near-complete integer transform; it still requires some real number multiplications for the inverse transform. The other is complete integer transform, which requires no real numbers multiplications. From Sections III to V, we will introduce the integer Fourier, Hartley, Sine, and Cosine transforms. We will also compare them with the original noninteger transforms. In Section VI, we will discuss the advantages of the integer transform over the approximation directly and discuss how to choose the parameters. Finally, in Section VII, concludes the paper.

II. General Method to Derive the Integer Transform Analogous to Some Noninteger Transform

A. Conditions for the Integer Transforms

In addition to all the entries being integers, the following requirements must also be satisfied for the integer transform if we want it to be approximated to the original noninteger transform.

a) The equality relations for the entries in each row of the integer transform matrix must be kept exactly the same as the original noninteger transform matrix. That is, if in the original noninteger transform matrix $\mathbf{A}$

$$A(m,n_1)=\tau\cdot A(m,n_2),\quad \tau=1,-1,j,-j \tag{3}$$

then the integer transform matrix $\mathbf{B}$ must have the following relations:

$$B(m,n_1)=\tau\cdot B(m,n_2). \tag{4}$$

b) In addition, the inequality relations for each row must also be the same. That is, if in the original noninteger transform matrix $\mathbf{A}$

$$\begin{aligned}\text{Re}(A(m,n_1))&\geq \text{Re}(A(m,n_2))\\ \text{Im}(A(m,n_3))&\geq \text{Im}(A(m,n_4))\end{aligned} \tag{5}$$

then in the integer transform matrix $\mathbf{B}$

$$\begin{aligned}\text{Re}(B(m,n_1))&\geq \text{Re}(B(m,n_2))\\ \text{Im}(B(m,n_3))&\geq \text{Im}(B(m,n_4)).\end{aligned} \tag{6}$$

For flexibility, we allow the original inequality relation without equality in (5) becomes the inequality relation with equality in (6). With the requirements of (a) and (b), we can assure each row of the integer transform matrix will be of the same shape as the row of the original transform matrix.

c) The sign for each entry must be the same as the original. That is, if $\mathbf{A}$ is the original transform matrix, and $\mathbf{B}$ is the integer transform matrix, then

$$\begin{aligned}\text{sgn}(\text{Re}(A(m,n_1)))&=\text{sgn}(\text{Re}(B(m,n_1)))\\ \text{sgn}(\text{Im}(A(m,n_1)))&=\text{sgn}(\text{Im}(B(m,n_1)))\end{aligned} \tag{7}$$

where

$$\begin{aligned}&\text{sgn}(x)=1 \quad \text{when } x>0\\ &\text{sgn}(x)=-1 \quad \text{when } x<0\\ &\text{sgn}(x)=0 \quad \text{when } x=0.\end{aligned} \tag{8}$$

If this requirement is satisfied, then we can assure the number and the locations of the zero-crossings for each row will remain the same.

d) The basis vectors of the integer transform matrix must also be orthogonal. That is

$$\sum_{k=0}^{N-1} B(m,k)\cdot\overline{B(n,k)}=C_m\cdot\delta_{mn} \tag{9}$$

where $\bar{}$ is the symbol of conjugation. The constraint of (9) can assure that the following relation will be satisfied:

$$\mathbf{B}\cdot\mathbf{B^H}\cdot\mathbf{C^{-1}}=\mathbf{B^H}\cdot\mathbf{C^{-1}}\cdot\mathbf{B}=\mathbf{I} \tag{10}$$

where $\mathbf{H}$ represents the Hermitian operation (transpose and conjugation), and $\mathbf{C}$ is

$$\begin{aligned}&C(m,n)=0 \quad \text{when } m\neq n\\ &C(m,m)=C_m \quad [C_m \text{ is defined as (9)}].\end{aligned} \tag{11}$$

Then, the inverse of the integer transform is $\mathbf{B^H C^{-1}}$. It is just the Hermitian of the forward integer transform matrix with the multiplication operation. The requirement of orthogonality is the key difference between the integer transform with the direct approximation. This will be illustrated in Section VI-A.

If all the four requirements are satisfied, then the integer transform we obtain will have the properties similar to those of the original noninteger transform.

From the above constraints, we can be assured that the forward integer transform only requires the fixed-point multiplication operations. However, since the inverse transform is $\mathbf{B^H C^{-1}}$, if the value of C_m is not as the form of 2^k where k is some integer, some floating-point multiplications are still required for the inverse transform. Thus, if the integer transform satisfies the above constraints, we can only be assured that the number of floating-point multiplications for the inverse transform is less than N (N is the number of points).

To fully avoid the floating-point multiplication operations, we will use another matrix $\mathbf{E^H}$ other than the Hermitian of the forward integer transform matrix as the inverse transform matrix. This is because it is very hard to find the integer matrix $\mathbf{B}$ such that all the value of C_m in (9) will be the form as 2^k. Requirements a), b), and c) for the forward integer matrix $\mathbf{B}$ must also be satisfied for the inverse transform matrix $\mathbf{E}$, and the requirement d) is modified as

$$\begin{aligned}&\sum_{k=0}^{N-1} B(m,k)\cdot\overline{E(n,k)}=D_m\cdot\delta_{mn}\\ &D_m=2^k \quad (k \text{ is an integer}).\end{aligned} \tag{12}$$

Then

$$\mathbf{B}\cdot\mathbf{E^H}\cdot\mathbf{D^{-1}}=\mathbf{E^H}\cdot\mathbf{D^{-1}}\cdot\mathbf{B}=\mathbf{I} \tag{13}$$

where

$$\begin{aligned} D(m,n)&=0 \quad \text{when } m\neq n\\ D(m,n)&=D_m \quad \text{when } m=n. \end{aligned} \tag{14}$$

Therefore, the inverse transform matrix is $\mathbf{E^H D^{-1}}$, and the real number multiplication will be fully not required. We call this type of integer transform the *complete integer transform* and call the former type of integer transform the *near-complete integer transform*.

Sometimes, however, the complete integer transforms are very hard to derive. In these cases, we will either only try to derive near-complete integer transform or relax some requirement in constraints a)–d). For example, we can relax some equality constraint in (4).

Thus, there are two types of integer transforms that can be derived:.

1) Near-Complete Integer Transform: In this case, the inverse of the forward transform matrix $\mathbf{B}$ will be $\mathbf{B^H C^{-1}}$, where $C(m,n)=C_m\cdot\delta_{m,n}$, and C_m is defined in (9). That is, the basis vectors of $\mathbf{B}$ are orthogonal. Some real numbers multiplications are still required (but we can assure they are less then N).

2) Complete Integer Transform: In this case, the inverse of the forward transform matrix $\mathbf{B}$ will be $\mathbf{E^H D^{-1}}$. That is, the basis vectors of $\mathbf{B}$ and $\mathbf{E}$ are *dual orthogonal*. No real number multiplication is required.

B. Process to Derive the Integer Transform

When we try to find the integer transform that is analogous to some noninteger transform, we can use the following steps.

1) Forming the Prototype Matrix—To Satisfy the Requirements a) and c): We first form the *prototype* of integer transform matrix and assign the unknowns properly to obey the relations as follows.

1) The equality relations for each row of the original noninteger transform should be kept.
2) The sign of the entries of original noninteger transform should be kept.

To be convenient, all the unknowns are assumed to be positive real integers. If the prototype matrix has too many unknowns, we can try to make some unknowns be the same or make some unknowns be 1 to reduce the number of unknowns.

If we want to derive the complete integer transform, this prototype is common for the forward transform matrix $\mathbf{B}$ and the inverse transform matrix $\mathbf{E}$, but sometimes, we must modify the prototype a little so that the complete integer transform can be derived. For example, when we derive the complete integer Fourier transform of six points, we have modified the original prototype (illustrated in Section III-D).

2) Constraints for Orthogonality (Equality Constraints)—To Satisfy Requirement d): In this step, we search for the requirements to make all the rows of the integer transform matrix be orthogonal to one another. These will be the constraints for the unknowns we have assigned.

If we want to derive the complete integer transform, these types of constraints become the constraints for dual orthogonality for the forward and inverse transform matrix.

3) Constraints for Inequality (Inequality Constraints)—To Satisfy Requirement b): In this step, we find the inequality relations among the unknowns from the inequality relations for each row of the original noninteger transform matrix. These will also be the constraints for the unknowns.

If we want to derive the complete integer transform, this type of constraint is common for the forward and inverse transform matrix.

4) Assign the Values for All the Unknowns: At last, we assign the values of unknowns. They must all be expressed as (2) and satisfy the constraints obtained from steps 2) and 3).

If we want to derive the complete integer transform, the values of unknowns assigned for the forward and inverse transform matrix would be different.

In Sections III to V, we will use this method to derive the integer transform.

We note, however, for the complete integer transform that the inverse matrix $\mathbf{E^H}$ will not be the Hermitian of the forward transform matrix $\mathbf{B}$. However, since $\mathbf{B}$ and $\mathbf{E}$ are of the same prototype, the structures of their implementation are also basically the same, except that the direction is reversed, and the parameters are different.

III. Integer Fourier Transform (8-Points and 6-Points)

A. Near-Complete Integer Fourier Transform

In this subsection, we will derive the 8-point integer Fourier transform (ITFT) analogous to the 8-points DFT:

$$F(m,n)=\exp(-jmn\pi/4)\quad m,n\in[0,1,\ldots 7]. \tag{15}$$

We will follow the method introduced in Section II-B. Because this is the first integer transform we derive in this paper, we will discuss the derivation process in detail.

Step 1) *Forming the prototype matrix of the 8-point ITFT.*

We first list the equality relations for each row of the DFT matrix as in Table I. The values of G and C mean, for the mth row of the 8-points DFT matrix

$$F(m,((n+G))_8)=C\cdot F(m,n) \tag{16}$$

where $(( ))$ is the *modulus addition*

$$((n+G))_N=n+G+h\cdot N$$

$$\text{where } h \text{ is some integer and } 0\le n+G+hN<N. \tag{17}$$

Then, together with the sign of the real and imaginary parts of each entry of the 8-point DFT matrix, we obtain the prototype matrix of the 8-point integer Fourier transform (ITFT) as in (18), shown at the bottom of the next page, and there are a total of 12 unknowns.

Step 2) *Searching the constraints for orthogonality of the unknowns.*

Before discussing the constraints for orthogonality, we first introduce a special way to conclude that two discrete vectors are orthogonal

to each other. Suppose there are two vectors $f_a(n), f_b(n)$ that have the length of N where N is even. In addition, suppose we use some method that make every two points form a set: $\{h_1, k_1\}, \{h_2, k_2\}, \ldots, \{h_{N/2}, k_{N/2}\}$, where $h_m, k_n \in \{0, 1, 2, \ldots, N-1\}$, and $h_m \neq k_n$ for all $m, n, h_m \neq h_n, k_m \neq k_n$ for $m \neq n$. Then, if $f_a(n), f_b(n)$ have the symmetry relations

$$f_a(k_s) = C_s \cdot f_b(h_s), \quad f_a(k_s) = D_s \cdot f_b(h_s) \quad \text{for } s = 1, 2, \ldots, N/2 \tag{19}$$

and C_s, D_s satisfies

$$C_s \cdot \overline{D_s} = -1 \quad \text{for } s = 1, 2, \ldots, N/2 \tag{20}$$

We then find that

$$\langle f_a(n), f_b(n) \rangle = \sum_{s=1}^{N/2} [f_a(h_s) \cdot \overline{f_b(h_s)} + f_a(k_s) \cdot \overline{f_b(k_s)}] \tag{21}$$

$$\langle f_a(n), f_b(n) \rangle = \sum_{s=1}^{N/2} [f_a(h_s) \cdot \overline{f_b(h_s)} + C_s \cdot \overline{D_s} \cdot f_a(h_s) \cdot \overline{f_b(h_s)}] = 0. \tag{22}$$

That is, $f_a(n), f_b(n)$ are orthogonal to each other. Thus, the relation of (19) and (20) will be the sufficient condition for the two functions to be orthogonal.

Thus, from (19)–(22) and Table I, we can conclude, except for {row 1, row 5} and {row 3, row 7}, that all pairs of rows will be orthogonal to each other.

Then, calculating the inner product of {row 1, row 5} and {row 3, row 7} directly, we find that if we want these pairs of rows to be orthogonal, the following equation must be satisfied:

$$a_1 \cdot d_1 = 2 \cdot a_2 \cdot d_2, \quad c_1 \cdot f_1 = 2 \cdot c_2 \cdot f_2. \tag{23}$$

Step 3) *Searching the inequality constraints of the unknowns.*

From the inequality relations for each row of the original DFT matrix, we obtain the inequality constraints for the ITFT as

$$a_1 \geq a_2 \quad c_1 \geq c_2 \quad d_1 \geq d_2 \quad f_1 \geq f_2 \tag{24}$$

and we have obtained all the constraints for the unknowns.

From all the constraint equations, we find that the unknowns set $\{a_1, a_2, d_1, d_2\}$ is independent of the set $\{c_1, c_2, f_1, f_2\}$, and their constraints are all the same. There is also no constraint for b_1, b_2, e_1, e_2. Thus, we can use the following equality equations:

$$a_1 = f_1, \quad a_2 = f_2, \quad d_1 = c_1 \quad d_2 = c_2, \quad b_1 = b_2 = e_1 = e_2 = 1. \tag{25}$$

We note that after we use the above equality relations, then the ITFT we obtain will satisfy

$$FI_p(m, n) = \overline{FI_p(N-m, n)}. \tag{26}$$

Then, the conjugation system property, which is the important property of the original DFT, will be kept. Although we can also use $a_1 = c_1, a_2 = c_2, d_1 = f_1, d_2 = f_2$ instead of $a_1 = f_1, a_2 = f_2, d_1 = c_1, d_2 = c_2$, when we use these equality relations, then the conjugation system relation of (26) will not exist. From (25), the number of unknowns of the prototype matrix is reduced from 12 to 4, and the prototype matrix of ITFT in (18) is simplified as in (27), shown at the bottom of the next page. The constraints then become

$$a_1 \cdot c_1 = 2 \cdot a_2 \cdot c_2, \quad a_1 \geq a_2, \quad c_1 \geq c_2. \tag{28}$$

Step 4) Assign the values of unknowns.

Then, we follow step 4 in Section II-B and assign the values of unknowns to satisfy (28) and (29), shown at the bottom of the next page. We find that there are many possible choices for the values of unknowns $\{a_1, a_2, c_1, c_2\}$. We list some possible choices of the parameters as Table II.

We can implement 8-point ITFT by the method of 1) decimation-in-frequency (as shown in Fig. 2) and 2) decimation-in-time (as shown in Fig. 3). We find that both the decimation-in-time implementation and the decimation-in-frequency implementation require eight integer multiplications. For the original DFT, we require four real number multi-

$$\mathbf{FI_P} = \begin{bmatrix} b_1 & b_1 & b_1 & b_1 & b_1 & b_1 & b_1 & b_1 \\ a_1 & a_2 - ja_2 & -ja_1 & -a_2 - ja_2 & -a_1 & -a_2 + ja_2 & ja_1 & a_2 + ja_2 \\ b_2 & -jb_2 & -b_2 & jb_2 & b_2 & -jb_2 & -b_2 & jb_2 \\ c_1 & -c_2 - jc_2 & jc_1 & c_2 - jc_2 & -c_1 & c_2 + jc_2 & -jc_1 & -c_2 + jc_2 \\ e_1 & -e_1 & e_1 & -e_1 & e_1 & -e_1 & e_1 & -e_1 \\ d_1 & -d_2 + jd_2 & -jd_1 & d_2 + jd_2 & -d_1 & d_2 - jd_2 & jd_1 & -d_2 - jd_2 \\ e_2 & je_2 & -e_2 & -je_2 & e_2 & je_2 & -e_2 & -je_2 \\ f_1 & f_2 + jf_2 & jf_1 & -f_2 + jf_2 & -f_1 & -f_2 - jf_2 & -jf_1 & f_2 - jf_2 \end{bmatrix} \tag{18}$$

TABLE I
SYMMETRY RELATION OF THE 8-POINTS DFT MATRIX

row	m=0	m=1	m=2	m=3	m=4	m=5	m=6	m=7
G=1	C= 1		C=$-j$		C=-1		C=j	
G=2	C= 1	C=$-j$	C=-1	C=j	C= 1	C=$-j$	C=-1	C=j
G=4	C= 1	C=-1	C=1	C=-1	C= 1	C=-1	C= 1	C=-1

TABLE II
SOME POSSIBLE CHOICES OF THE PARAMETERS OF 8-POINTS ITFT

a_1	2	3	4	5	8	10	17	99	500
a_2	1	2	3	3	5	7	12	70	353
c_1	1	4	3	6	5	7	24	140	353
c_2	1	3	2	5	4	5	17	99	500

plications. Although the 8-point ITFT requires twice the multiplication operations and since all the multiplication operations are fixed-point, it will still be more efficient.

From (10), the inverse 8-point ITFT is as in (29), where

$$\begin{aligned} &C(m,n)=0 \quad \text{when } m\neq n\\ &C(0,0)=C(2,2)=C(4,4)=C(6,6)=8\\ &C(1,1)=C(7,7)=4a_1^2+8a_2^2\\ &C(3,3)=C(5,5)=4c_1^2+8c_2^2. \end{aligned} \tag{30}$$

The inverse 8-point ITFT can be implemented as in Fig. 4. We note that the implementation of the inverse ITFT has almost the same structure as the implementation of the forward ITFT in Fig. 2. The difference is that the direction is reverse, and there are some multiplication operations for the input of inverse ITFT. In fact, for all of the the integer transform, the implementations of the forward and inverse transforms will have *the same structure*.

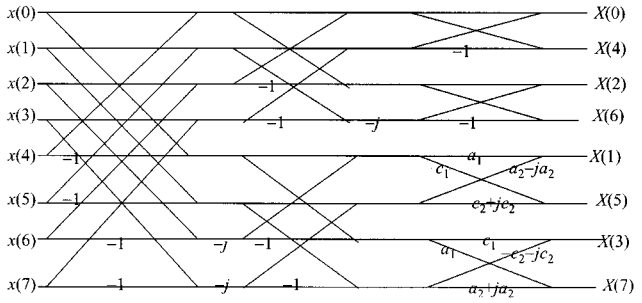

Fig. 2. Decimation-in-frequency 8-point integer Fourier transform.

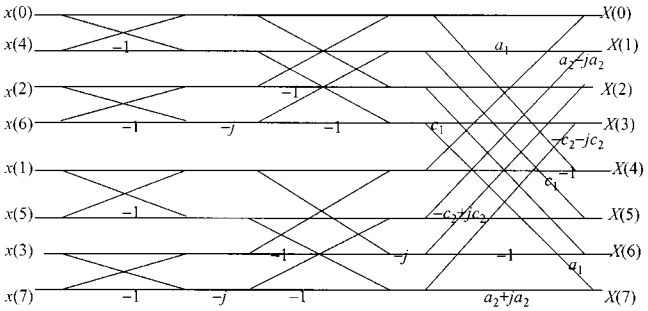

Fig. 3. Decimation-in-time 8-point integer Fourier transform.

B. Complete Integer Fourier Transform of 8-Points

After the near-complete integer Fourier transform has been derived, now we will also derive the complete integer Fourier transform. We first discuss the case of 8-points.

We will also use (27) as the prototype of the forward 8-point complete ITFT. In addition, for the inverse transform, we use a similar prototype but modify the parameters $\{a_1, a_2, c_1, c_2\}$ as

$$\mathbf{FI_P}=\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1\\ a_1 & a_2-ja_2 & -ja_1 & -a_2-ja_2 & -a_1 & -a_2+ja_2 & ja_1 & a_2+ja_2\\ 1 & -j & -1 & j & 1 & -j & -1 & j\\ c_1 & -c_2-jc_2 & jc_1 & c_2-jc_2 & -c_1 & c_2+jc_2 & -jc_1 & -c_2+jc_2\\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1\\ c_1 & -c_2+jc_2 & -jc_1 & c_2+jc_2 & -c_1 & c_2-jc_2 & jc_1 & -c_2-jc_2\\ 1 & j & -1 & -j & 1 & j & -1 & -j\\ a_1 & a_2+ja_2 & ja_1 & -a_2+ja_2 & -a_1 & -a_2-ja_2 & -ja_1 & a_2-ja_2 \end{bmatrix}. \tag{27}$$

$$\mathbf{FI_P^{-1}}=\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1\\ a_1 & a_2-ja_2 & -ja_1 & -a_2-ja_2 & -a_1 & -a_2+ja_2 & ja_1 & a_2+ja_2\\ 1 & -j & -1 & j & 1 & -j & -1 & j\\ c_1 & -c_2-jc_2 & jc_1 & c_2-jc_2 & -c_1 & c_2+jc_2 & -jc_1 & -c_2+jc_2\\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1\\ c_1 & -c_2+jc_2 & -jc_1 & c_2+jc_2 & -c_1 & c_2-jc_2 & jc_1 & -c_2-jc_2\\ 1 & j & -1 & -j & 1 & j & -1 & -j\\ a_1 & a_2+ja_2 & ja_1 & -a_2+ja_2 & -a_1 & -a_2-ja_2 & -ja_1 & a_2-ja_2 \end{bmatrix}^{\mathbf{T}} \cdot \mathbf{C}^{-1} \tag{29}$$

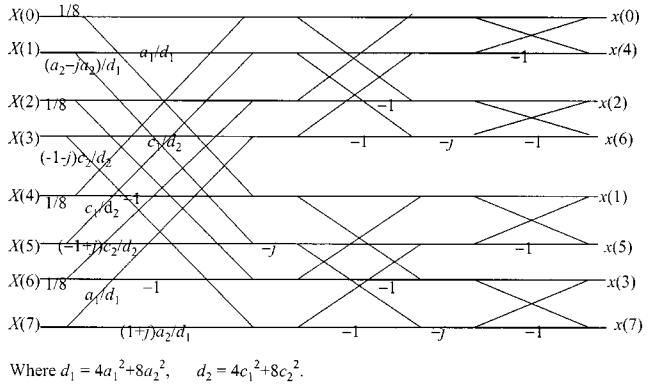

Fig. 4. Implementation of the 8-points inverse integer Fourier transform.

$\{a_3, a_4, c_3, c_4\}$, shown in (31) at the bottom of the page. Therefore, there are a total of eight parameters. The equality constraint in (28) now becomes

$$(1)\ a_1 \cdot c_3 = 2 \cdot a_2 \cdot c_4, \quad (2)\ a_3 \cdot c_1 = 2 \cdot a_4 \cdot c_2. \tag{32}$$

From the requirement of (12), we also obtain the equality constraint as

$$\begin{aligned} (3)\ & a_1 \cdot a_3 + 2 \cdot a_2 \cdot a_4 = 2^k \\ (4)\ & c_1 \cdot c_3 + 2 \cdot c_2 \cdot c_4 = 2^h \end{aligned} \tag{33}$$

where k, h are integer numbers. The inequality constraints in (28) become

$$(5)\ a_1 \geq a_2, \quad (6)\ c_1 \geq c_2, \quad (7)\ a_3 \geq a_4, \quad (8)\ c_3 \geq c_4. \tag{34}$$

Thus, there are a total of four equality constraints and four inequality constraints.

To find the values of the eight parameters $\{a_1, a_2, c_1, c_2, a_3, a_4, c_3, c_4\}$ that satisfy all the above eight constraints directly is time consuming. We introduce a convenient process to assign the values of the parameters as follows. From (1) and (2), we can set

$$c_3 = 2 \cdot a_2, \quad c_4 = a_1, \quad a_3 = 2 \cdot c_2, \quad a_4 = c_1. \tag{35}$$

Then, (3) and (4) become

$$\begin{aligned} (3')\ & 2 \cdot (a_1 \cdot c_2 + a_2 \cdot c_1) = 2^k \\ (4')\ & 2 \cdot (c_1 \cdot a_2 + c_2 \cdot a_1) = 2^h. \end{aligned} \tag{36}$$

We can then implement the following process to find the values of the parameters.

a) Choose the values of a_1, a_2 such that a_1, a_2 are integer numbers and

$$2 \cdot a_2 \geq a_1 \geq a_2. \tag{37}$$

b) Find the integer values of c_1, c_2 such that

$$2 \cdot c_2 \geq c_1 \geq c_2, \quad \text{and} \quad a_1 \cdot c_2 + a_2 \cdot c_1 = 2^n \quad \text{where } n \text{ is integer.} \tag{38}$$

c) Set the value of a_3, a_4, c_3, c_4 as

$$\begin{aligned} & c_3 = 2^{h+1} a_2, \quad c_4 = 2^h a_1, \quad a_3 = 2^{h+1} c_2 \\ & a_4 = 2^h c_1 \quad h \text{ is any integer.} \end{aligned} \tag{39}$$

Then, the parameters are all obtained.

We list some examples of the parameters as follows.

1) $\{a_1 = 2, a_2 = 1, c_1 = 2, c_2 = 1, a_3 = 1, a_4 = 1, c_3 = 1, c_4 = 1\}$
   This is the smallest, simplest integer solution for the parameters. If

$$\sum_{n=0}^{N-1} FI_P(m,n) \cdot \overline{IFI_P(m,n)} = D_m \tag{40}$$

   then $D_0 = D_2 = D_4 = D_6 = 2^3, D_1 = D_3 = D_5 = D_7 = 2^4$.
2) $\{a_1 = 7, a_2 = 5, c_1 = 13, c_2 = 9, a_3 = 18, a_4 = 13, c_3 = 10, c_4 = 7\}$.

We note that when we choose the parameters as the values list above, the ratios of $a_1 : a_2, c_1 : c_2, a_3 : a_4, c_3 : c_4$ are all near to 1.414:1, which is ratio of the original discrete Fourier transform. If D_m is defined as (82), then in this case, $D_0 = D_2 = D_4 = D_6 = 2^3, D_1 = D_3 = D_5 = D_7 = 2^9$.

We use Table III to list some possible choices of the values of the parameters of the 8-point ITFT.

We can also implement the forward transform as Fig. 2 or 3 and implement the inverse transform as Fig. 5.

We note that the numbers of multiplication operations required for the 8-point original and complete integer Fourier transform are

Original DFT:

four real numbers, 8 fixed-point multiplication operations

Complete ITFT:

20 fixed-points multiplication operations.

$$\mathbf{IFI_P} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ a_3 & a_4 - ja_4 & -ja_3 & -a_4 - ja_4 & -a_3 & -a_4 + ja_4 & ja_3 & a_4 + ja_4 \\ 1 & -j & -1 & j & 1 & -j & -1 & j \\ c_3 & -c_4 - jc_4 & jc_3 & c_4 - jc_4 & -c_3 & c_4 + jc_4 & -jc_3 & -c_4 + jc_4 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ c_3 & -c_4 + jc_4 & -jc_3 & c_4 + jc_4 & -c_3 & c_4 - jc_4 & jc_3 & -c_4 - jc_4 \\ 1 & j & -1 & -j & 1 & j & -1 & -j \\ a_3 & a_4 + ja_4 & ja_3 & -a_4 + ja_4 & -a_3 & -a_4 - ja_4 & -ja_3 & a_4 - ja_4 \end{bmatrix}. \tag{31}$$

TABLE III
SOME POSSIBLE CHOICES OF THE VALUES OF THE PARAMETERS OF 8-POINT COMPLETE ITFT

a_1	2	7	3	4	4	5	10
a_2	1	5	2	3	3	4	7
c_1	2	13	17	12	44	17	18
c_2	1	9	10	7	31	12	13
a_3	1	18	34	7	31	24	13
a_4	1	13	10	6	22	17	9
c_3	1	10	4	3	3	8	7
c_4	1	7	3	2	2	5	5

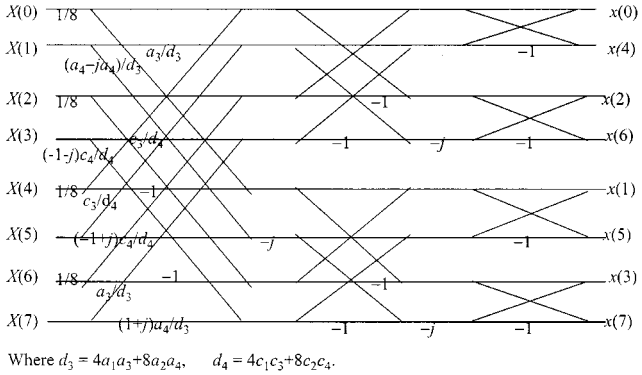

Fig. 5. Implementation of the inverse 8-point complete ITFT.

That is, we replace four real number multiplication operations with 12 fixed-point multiplication operations. The complete ITFT will be much faster than the original DFT.

C. The Properties and Performance of the 8-Point Complete ITFT

We will see some interesting properties of the 8-point complete integer Fourier transform (ITFT). Many properties of the original 8-point DFT will also exist for the 8-point ITFT.

a) Even input $\rightarrow$ even output, odd input $\rightarrow$ odd output:
If $x(n)$ is even, and $y(n)$ is odd

$$x(n) = x(N-n), \quad y(n) = y(N-m) \tag{41}$$

then the complete ITFT of $x(n)$ [denoted by $X(m)$] will also be even, and the complete ITFT of $y(n)$ [denoted by $Y(m)$] will also be odd

$$X(m) = X(N-m), \quad Y(m) = -Y(N-m). \tag{42}$$

b) Pure real input $\rightarrow$ conjugated symmetry output, pure imaginary input $\rightarrow$ conjugated asymmetry output:
If $x(n)$ is pure real, $y(n)$ is pure imaginary, i.e., $\mathrm{Im}(x(n)) = 0, \mathrm{Re}(y(n) = 0)$, and the complete ITFT of $x(n), y(n)$ are $X(m), Y(m)$, then

$$X(m) = \overline{X(N-m)}, \quad Y(m) = -\overline{Y(N-m)}. \tag{43}$$

c) Power preservation property:
If $X(m), Y(m)$ are the complete ITFT of $x(n), y(n)$, then

$$\sum_{n=0}^{N-1} x(n) \cdot \overline{y(n)} = \sum_{m=0}^{N-1} X(m) \cdot \overline{Y(m)} \cdot D_m^{-1}$$
where D_m is defined as (40). (44)

d) Time-reverse property and conjugation property:

$$\mathrm{ITFT}(x(((-n))_N)) = X(((-m))_N)$$
$$\mathrm{ITFT}(\overline{x(n)}) = \overline{X(((-m))_N)}. \tag{45}$$

e) Shift-invariant property:
For the 8-point complete ITFT, if

$$\mathrm{ITFT}(x(n)) = X(m), \quad \mathrm{ITFT}(x(((n-k))_N)) = X_k(m) \tag{46}$$

then

$$X_k(m) = e^{-j\frac{2\pi k}{N}m} X(m)$$
when k is even or when k is odd and m is even (47)
$$X_k(m) \approx e^{-j\frac{2\pi k}{N}m} X(m)$$
when k is odd and m is odd. (48)

Equation (47) is satisfied because the equality relations for each row of the original 8-point DFT matrix (listed in Table I) have been kept for the 8-point ITFT; therefore, for the 8-point ITFT matrix, the relation that $FI(m,((n+G))_N) = \exp(-j2\pi mG/N) \cdot FI(m,n)$ is still satisfied when G is even or when m is even. Thus, if the amount of time-shift k is even or m is even, then

$$\begin{aligned} X_k(m) &= \sum_{n=0}^{N-1} FI(m,n) x(((n-k))_N) \\ &= \sum_{n=0}^{N-1} FI(m,((n+k))_N) x(n) \\ &= e^{-j\frac{2\pi mk}{N}} \sum_{n=0}^{N-1} FI(m,n) x(n) \\ &= e^{-j\frac{2\pi mk}{N}} \cdot X(m) \end{aligned}$$

and the relation of (47) has been proved.

Then, we will see the performance of the 8-point ITFT. Here, the parameters we choose are $a_1 = 7, a_2 = 5, c_1 = 13, c_2 = 9, a_3 = 18, a_4 = 13, c_3 = 10, c_4 = 7$. Then, we just require 4 bits to implement the kernel of forward ITFT (since the largest entry is $c_1 = 13$) and require 5 bits to implement the kernel of inverse ITFT (since the largest entry is $a_3 = 18$).

We first see the transform result. We choose two inputs:

$$\mathbf{x_1} = [2\ 3\ 4\ 5\ 4\ 5\ 2\ 3] \tag{49}$$
$$\mathbf{x_2} = [2.8\ 4.3 - 0.6j\ 3.7 + 0.9j\ 3.1 - 0.6j\ 4.6\ 3.1 + 0.6j\ 3.7 - 0.9j\ 4.3 + 0.6j]. \tag{50}$$

We plot them in Fig. 6(a) and (b). (The real part and imaginary part are plotted in separation.). Then, we do the original 8-point DFT, and plot the transform results in Fig. 6(c) and (d). Then, we do the 8-point complete ITFT. To facilitate the comparison, we will normalize the transform results. That is, for the transform result of the complete ITFT

$$X(m) = \sum_{n=0}^{7} FI(m,n) \cdot x(n) \tag{51}$$

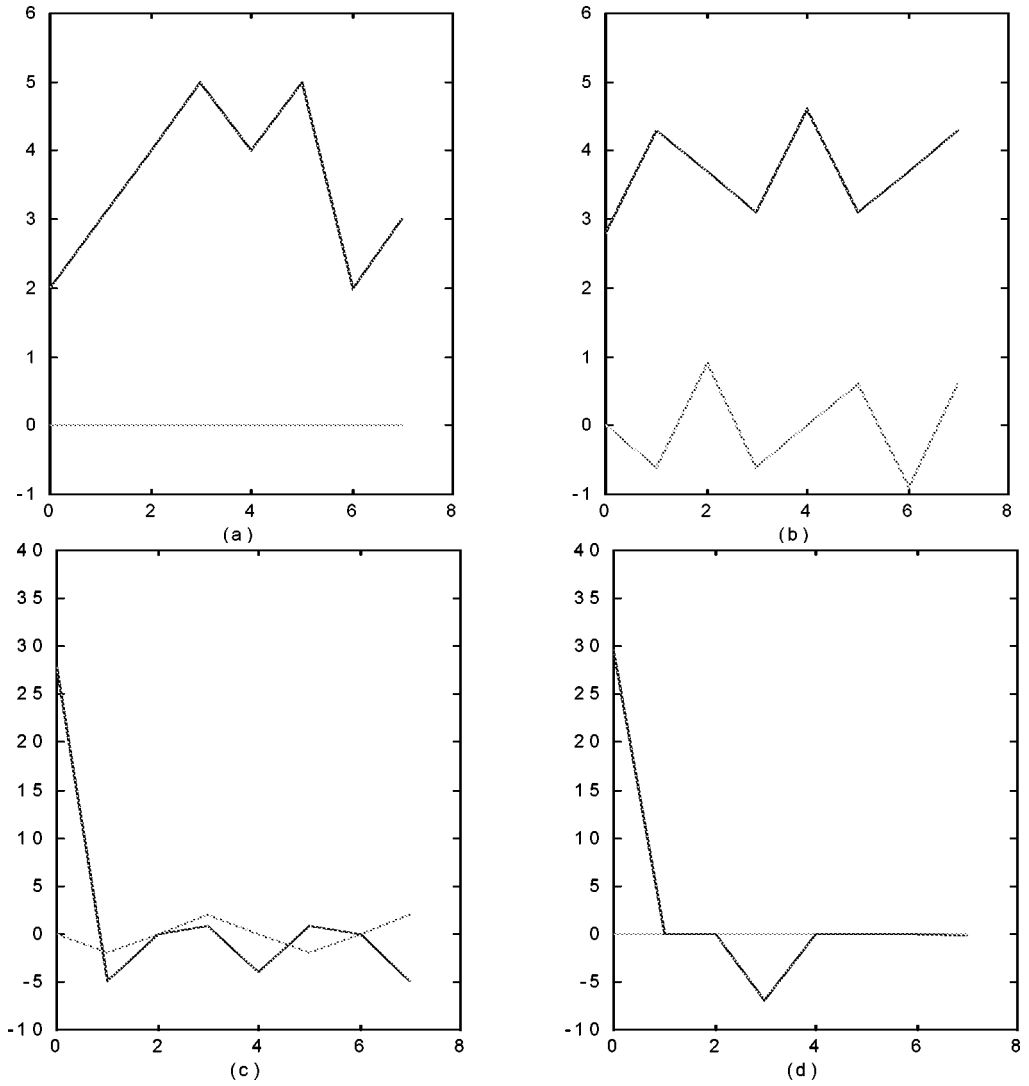

Fig. 6. Transform results of the original, approximated DFTs and complete ITFT. (a), (b) Original signals. (c), (d) Transform results of original DFT.

we will normalize $X(m)$ by the first column of the forward integer transform matrix

$$\tilde{X}(m) = X(m)/FI(m,0). \tag{52}$$

We plot the normalized transform results of the 8-point complete ITFT in Fig. 7(c) and (d).

We will compare the complete ITFT with the direct approximation of DFT. Since, in this example, we require 4 bits to implement the kernel of forward ITFT and require 5 bits to implement the kernel of inverse ITFT, we will also use 4 bits to approximate the forward DFT and use 5 bits to approximate the inverse DFT:

$$\hat{F}(m,n) \approx F(m,n) = \sum_{h=0}^{3} a_h \cdot 2^{H-h}$$
$$I\hat{F}(m,n) \approx IF(m,n) = \sum_{h=0}^{4} b_h \cdot 2^{H-h}. \tag{53}$$

Here, we use $F(m,n), IF(m,n), \hat{F}(m,n), I\hat{F}(m,n)$ to denote DFT, IDFT, approximated DFT, and approximated IDFT, respectively. Then, we use the approximated DFT to calculate the transform results of $\mathbf{x_1}, \mathbf{x_2}$ and plot the results in Fig. 7(a) and (b).

We then use the following equation to calculate the approximation error.

$$\text{err} = \sqrt{\sum_{m=0}^{N-1} |\tilde{X}(m) - X_F(m)|^2 \Big/ \sum_{m=0}^{N-1} |X_F(m)|^2} \tag{54}$$

where $X_F(m)$ is the transform result of original DFT. Then, the errors of the four results in Fig. 7 are

$$\begin{aligned} \text{for Fig. 7(a): err} &= 5.3363 \cdot 10^{-3} \\ \text{for Fig. 7(b): err} &= 4.3759 \cdot 10^{-3} \\ \text{Fig. 7(c): err} &= 3.1655 \cdot 10^{-3} \\ \text{for Fig. 7(d): err} &= 2.5958 \cdot 10^{-3}. \end{aligned} \tag{55}$$

We find that the errors are very small. The approximated results of the complete ITFT [Fig. 7(c) and (d)] are even better than the direct approximation (Fig. 7(a), (b)) and are very similar to the transform results for the original 8-point DFT.

Besides, we note that input $x_2[n]$ is the combination of a low-frequency component and a high-frequency component. These

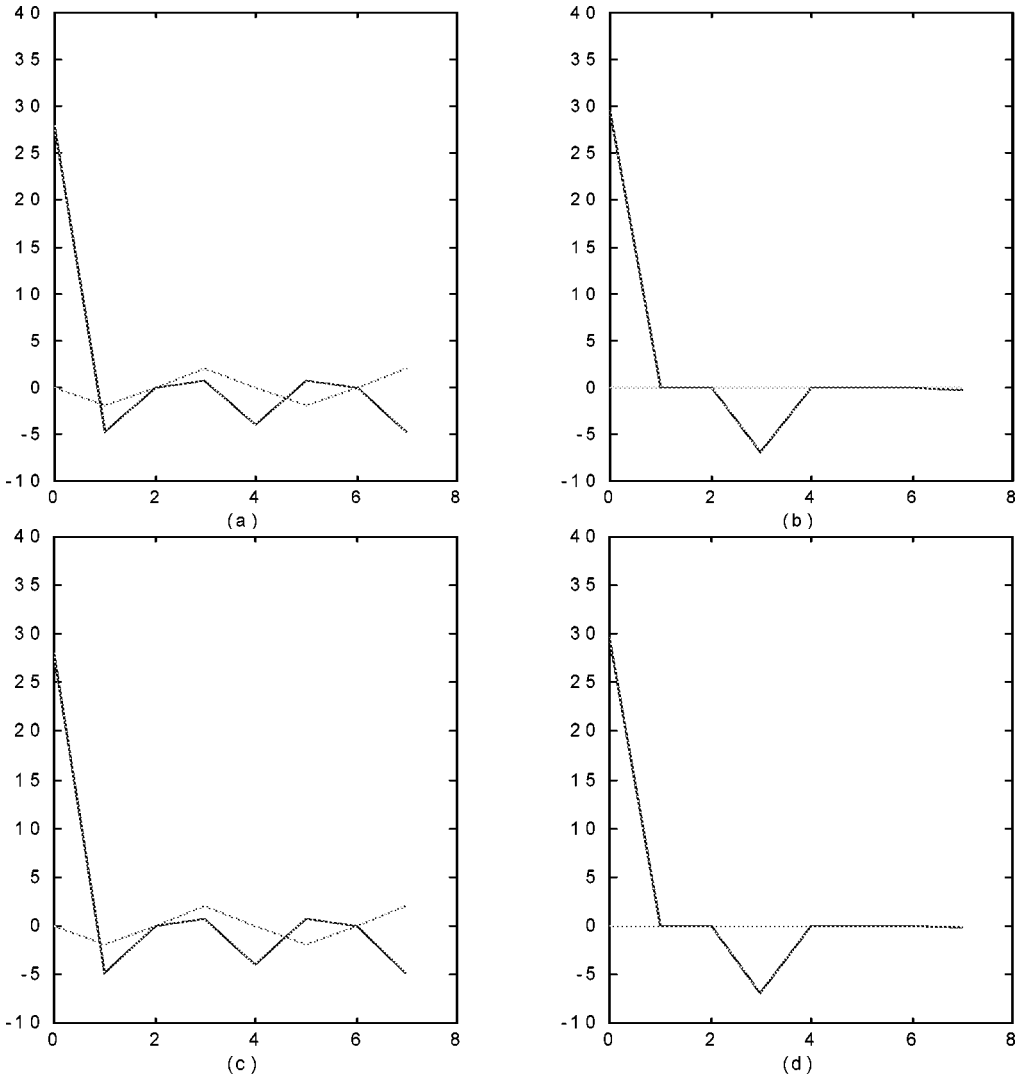

Fig. 7. Transform results of the original, approximated DFTs and complete ITFT. (a), (b) Transform results of approximated DFT. (c), (d) Transform results of complete ITFT.

two components can be separated by both original and complete ITFTs. Therefore, the complete ITFT can also be used for the filter design. We can set

$$X_f(m) = X(m) \quad (X(m) \text{ is the complete ITFT of } x(n)) \quad \text{when } m \neq 3,$$
$$X_f(m) = 0 \quad \text{when } m \neq 0$$

to remove the high-frequency component and then do the inverse complete ITFT for $X_f(m)$

$$x_o(n) = \sum_{n=0}^{7} \overline{IFI(n,m)} \cdot D_m^{-1} \cdot X_f(m)$$

to obtain the filter output. In Fig. 8, we show the filter results. We plot the original signal $\mathbf{x_2}$ in Fig. 8(a). Then, we use the original DFT to filter out the high-frequency component, and plot the result in Fig. 8(b). Then, we use the approximated DFT [as (53)] and plot the result in Fig. 8(c). Then, we use the complete ITFT and plot the result in Fig. 8(d). Then, we use (54) to calculate the error, but $X_F(m)$ is changed to the filter output by the original DFT, and $\tilde{X}(m)$ is changed to the filter outputs by the approximated DFT and by the complete ITFT. The errors are

$$\text{for Fig. 8(c): err} = 3.2639 \cdot 10^{-3}$$
$$\text{for Fig. 8(d): err} = 1.1952 \cdot 10^{-3}.$$

We find that the error of approximated DFT is about three times the error of the complete ITFT. This is because the complete ITFT is orthogonal and, hence, is reversible. However, the approximated DFT is not orthogonal and reversible, especially when we use fewer bits. Therefore, for the applications that we must do both the forward and inverse transforms, such as the filter design, the performance of complete ITFT will be much better than the approximated DFT.

Then, we see the displacement property of the complete ITFT. Here, we use the displacement of input $x_2[n]$ [which is defined in (50)] as the input:

$$x_3[n] = x_2[((n+1))_8], \quad x_4[n] = x_2[((n+2))_8]. \quad (56)$$

We plot $\mathbf{x_3}, \mathbf{x_4}$ in Fig. 9(a) and (b). Then, we do the complete ITFT for $\mathbf{x_3}, \mathbf{x_4}$ and plot the amplitudes of the normalized results in Fig. 10(c) and (d). We also do the original DFT and the

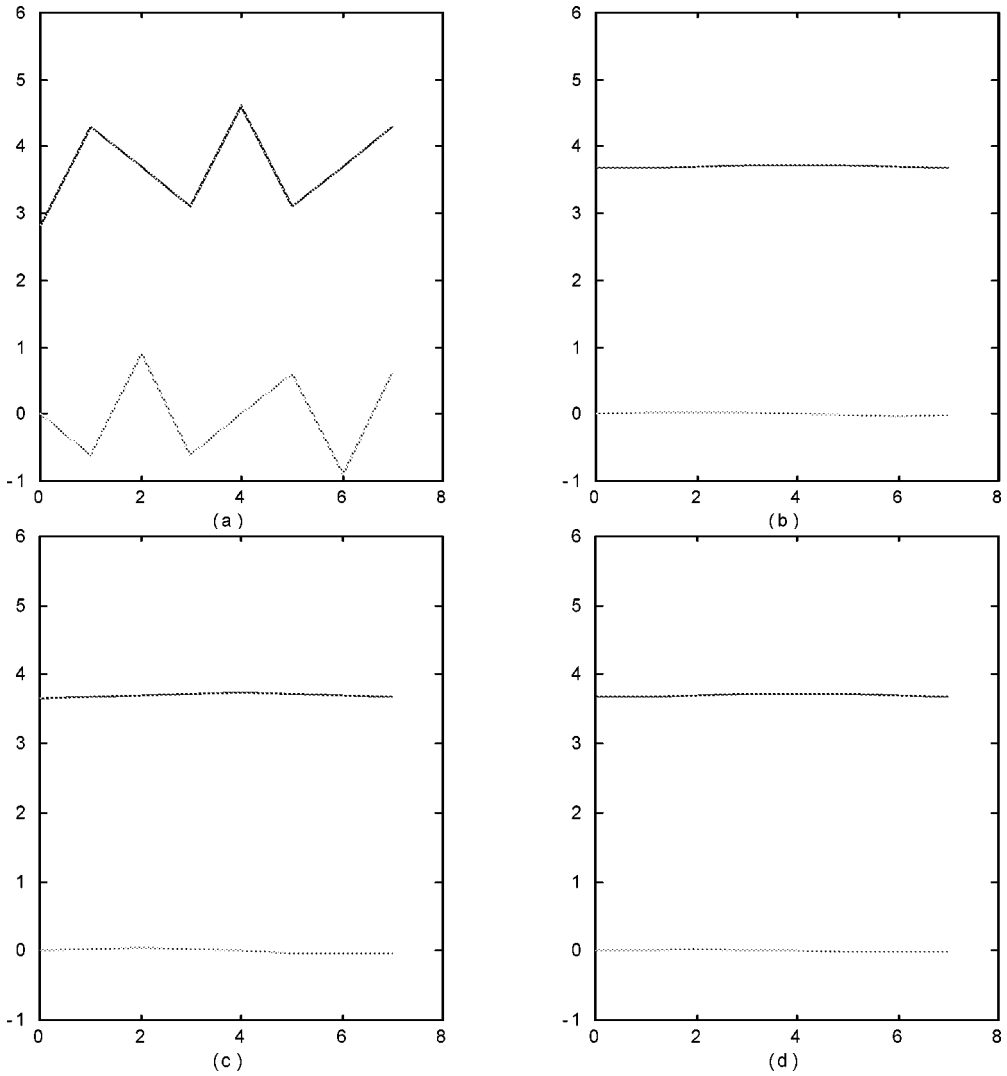

Fig. 8. Filter experiment. (a) Original signal. (b) Filter by original DFT. (c) Filter by approximated DFT. (d) Filter by complete ITFT.

approximated DFT [as in (53)] for $\mathbf{x_3}, \mathbf{x_4}$, plot the amplitudes of the results of the original DFT in Fig. 9(a) and (b), and plot the amplitudes of the results of the approximated DFT in Fig. 10(a) and (b).

We find that for the complete ITFT, the displacement property is almost kept. The signal after displacement will have the same transform amplitude as the original signal. When we compare the phase, there is an interesting phenomenon. We find that

$$\begin{aligned}&\text{angle}(\mathbf{X_3}) - \text{angle}(\mathbf{X_2})\\ &\quad = [0 \quad -45 \quad -90 \quad -135 \quad 0 \quad 135 \quad -270 \quad -315]\end{aligned} \tag{57}$$

$$\begin{aligned}&\text{angle}(\mathbf{X_4}) - \text{angle}(\mathbf{X_2})\\ &\quad = [0 \quad -90 \quad -180 \quad -270 \quad 0 \quad -90 \quad -180 \quad -270].\end{aligned} \tag{58}$$

This is exactly the same as the original DFT.

Then, we use the equation as below to calculate the difference between $X_2(m)$ (the ITFT or approximated DFT of $x_2(n)$) and $X_3(m), X_4(m)$ (the ITFT or approximated DFT of $x_3(n), x_4(n)$):

$$\text{dif} = \sqrt{\sum_{m=0}^{N-1} |X_{3\,\text{or}\,4}(m) - X_2(m)|^2 \Big/ \sum_{m=0}^{N-1} |X_2(m)|^2}. \tag{59}$$

Then

$$\begin{aligned}&\text{for Fig. 10(a): dif} = 6.3832 \cdot 10^{-3}\\ &\text{for Fig. 10(b): dif} = 0 \quad \text{(approximated DFT)}\\ &\text{Fig. 10(c): dif} = 2.3401 \cdot 10^{-3}\\ &\text{for Fig. 10(d): dif} = 0 \quad \text{(complete ITFT).}\end{aligned} \tag{60}$$

Thus, the complete ITFT will have more excellent displacement-invariant property than the approximated DFT, especially when the bits are fewer.

D. Complete Integer Fourier Transform of 6-Points

Because $6 \neq 2^k$, it seems impossible to avoid the real numbers multiplication for 6-point DFT, but in fact, we can also derive the 6-point complete ITFT.

The original 6-point DFT is

$$F(m,n) = \exp(-j \cdot m \cdot n \cdot \pi/3) \quad \text{where } m,n \in [0,1,\ldots 5]. \quad (61)$$

By a similar process as the derivation of the 8-point ITFT, we obtain the prototype of the 6-point ITFT as in (62), shown at the bottom of the page. For the prototype as in (92), it is impossible to derive the 6-point complete ITFT; therefore, we modify the prototype a little as in (63) at the bottom of the page. The prototype for the inverse transform can be set as in (64), shown at the bottom of the page. From the orthogonality, we obtain the constraints as

$$\begin{aligned} &(1)\ d_1 a_2 = 2 d_2 b_2, \quad (2)\ d_3 a_1 = 2 d_4 b_1 \\ &(3)\ a_1 a_2 + 2 b_1 b_2 - 2 c_1 c_2 = 0. \end{aligned} \quad (65)$$

From the requirement of (12), we obtain the constraints as

$$\begin{aligned} &(4)\ d_1 d_3 + 2 d_2 d_4 = 2^k \\ &(5)\ a_1 a_2 + 2 b_1 b_2 + 2 c_1 c_2 = 2^h. \end{aligned} \quad (66)$$

From the inequality relations of the entries of the original 6-point DFT matrix, we obtain the inequality relations as

$$(6)\ b_1 \le c_1 \le a_1, \quad (7)\ b_2 \le c_2 \le a_2. \quad (67)$$

Besides, to make (93) and (94) similar to (92), we also impose the following inequality constraints:

$$(8)\ d_2/d_1 \approx 1, \quad (9)\ d_4/d_3 \approx 1. \quad (68)$$

There are a total of five equality constraints and four inequality constraints for ten parameters.

From the constraints 3, 5, we can convert these constraints as

$$(3')\ c_1 c_2 = 2^{h-2}, \quad (5')\ a_1 a_2 + 2 b_1 b_2 = 2^{h-1} \quad (69)$$

and from the constraints 1, 2, $a_2 = 2(d_2/d_1) b_2, a_1 = 2(d_4/d_3) b_1$. We can substitute them into the new constraint 5', and we obtain

$$\left(2\frac{d_2 d_4}{d_1 d_3} + 1\right) b_1 b_2 = 2^{h-2}. \quad (70)$$

Together with constraint 4, we obtain

$$b_1 b_2 = 2^{h-2-k} \cdot d_1 d_3. \quad (71)$$

Thus, to find the values of the parameters, we can follow the process as follows.

1) Choose the value of d_1, d_2, and $d_1/d_2 \approx 1$ must be satisfied.
2) Find the value of d_3, d_4 such that $d_3/d_4 \approx 1$, and
$$d_1 d_3 + 2 d_2 d_4 = 2^k, \quad \text{where } k \text{ is integer.} \quad (72)$$
3) Choose b_1, b_2 as
$$b_1 = 2^m \cdot d_3, \quad b_2 = 2^n \cdot d_1 \quad \text{where } m,n \text{ are integer numbers.} \quad (73)$$
4) Then, values of a_1, a_2 can be calculated as
$$a_1 = 2^{m+1} \cdot d_4, \quad a_2 = 2^{n+1} \cdot d_2 \quad m,n \text{ the same as above.} \quad (74)$$

$$\mathbf{FI_p} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ a_1 & b_1 - jc_1 & -b_1 - jc_1 & -a_1 & -b_1 + jc_1 & b_1 + jc_1 \\ a_1 & -b_1 - jc_1 & -b_1 + jc_1 & a_1 & -b_1 - jc_1 & -b_1 + jc_1 \\ 1 & -1 & 1 & -1 & 1 & -1 \\ a_1 & -b_1 + jc_1 & -b_1 - jc_1 & a_1 & -b_1 + jc_1 & -b_1 - jc_1 \\ a_1 & b_1 + jc_1 & -b_1 + jc_1 & -a_1 & -b_1 - jc_1 & b_1 - jc_1 \end{bmatrix}. \quad (62)$$

$$\mathbf{FI_p} = \begin{bmatrix} d_1 & d_2 & d_2 & d_1 & d_2 & d_2 \\ a_1 & b_1 - jc_1 & -b_1 - jc_1 & -a_1 & -b_1 + jc_1 & b_1 + jc_1 \\ a_1 & -b_1 - jc_1 & -b_1 + jc_1 & a_1 & -b_1 - jc_1 & -b_1 + jc_1 \\ d_1 & -d_2 & d_2 & -d_1 & d_2 & -d_2 \\ a_1 & -b_1 + jc_1 & -b_1 - jc_1 & a_1 & -b_1 + jc_1 & -b_1 - jc_1 \\ a_1 & b_1 + jc_1 & -b_1 + jc_1 & -a_1 & -b_1 - jc_1 & b_1 - jc_1 \end{bmatrix}. \quad (63)$$

$$\mathbf{IFI_p} = \begin{bmatrix} d_3 & d_4 & d_4 & d_3 & d_4 & d_4 \\ a_2 & b_2 - jc_2 & -b_2 - jc_2 & -a_2 & -b_2 + jc_2 & b_2 + jc_2 \\ a_2 & -b_2 - jc_2 & -b_2 + jc_2 & a_2 & -b_2 - jc_2 & -b_2 + jc_2 \\ d_3 & -d_4 & d_4 & -d_3 & d_4 & -d_4 \\ a_2 & -b_2 + jc_2 & -b_2 - jc_2 & a_2 & -b_2 + jc_2 & -b_2 - jc_2 \\ a_2 & b_2 + jc_2 & -b_2 + jc_2 & -a_2 & -b_2 - jc_2 & b_2 - jc_2 \end{bmatrix}. \quad (64)$$

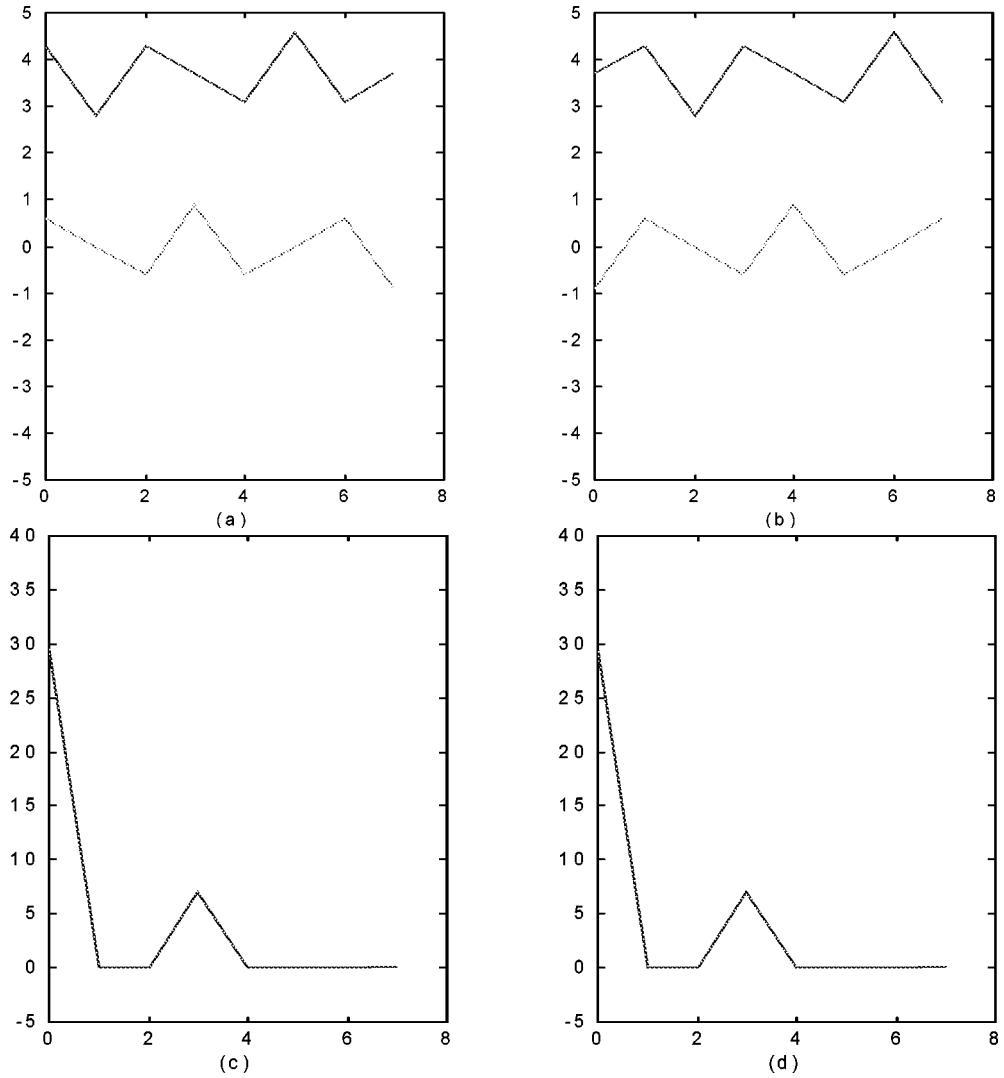

Fig. 9. Space-invariant property. (a), (b) Shifted input. (c), (d) Amplitude of the transform results of original ITFT.

5) Find the values of c_3, c_4 such that

$$c_1 c_2 = 2^{m+n+k}, \quad m, n, k \text{ are the same as above.} \tag{75}$$

In addition, the inequality constraints 6 and 7 must be satisfied.

We give some examples of the values of parameters as follows.

a) $d_1 = 1, d_2 = 1, d_3 = 6, d_4 = 5, a_1 = 5, b_1 = 3, c_1 = 4, a_2 = 2, b_2 = 1, c_2 = 2$.
This would be the smallest possible choice of the parameters. If

$$\sum_{n=0}^{5} FI_P(m,n) \cdot IFI_P(m,n) = D_m \qquad m = 0,1,2,3,4 \tag{76}$$

then $D_0 = D_3 = 2^5, D_1 = D_2 = D_4 = D_5 = 2^6$.

b) $d_1 = 9, d_2 = 10, d_3 = 36, d_4 = 35, a_1 = 35, b_1 = 18, c_1 = 32, a_2 = 20, b_2 = 9, c_2 = 16$.
In this case, the 6-point complete ITFT will be more similar to the original 6-point DFT. If D_m is defined as (107), then $D_0 = D_3 = 2^{11}, D_1 = D_2 = D_4 = D_5 = 2^{12}$.

We can implement the forward 6–point complete ITFT as Fig. 11. There are ten integer multiplication operations required. In contrast, to implement the original 6-point DFT, we need eight real multiplication operations. If we consider the overall system, that is, include the forward and inverse transform, then the numbers of multiplication operations required are as

original 6-point DFT:

22 real number multiplication operations.

(eight for forward, eight for inverse, six for normalization)

complete 6-point ITFT:

26 fixed-point multiplication operations.

(ten for forward, ten for inverse, six for normalization).

It is obvious that the complete integer 6-points DFT is much more efficient.

IV. Integer Hartley Transform

The discrete Hartley transform (DHT) [5] is defined as

$$H(m,n) = \text{cas}(mn\pi/N) \quad m,n \in [0,1,\ldots N-1] \tag{77}$$

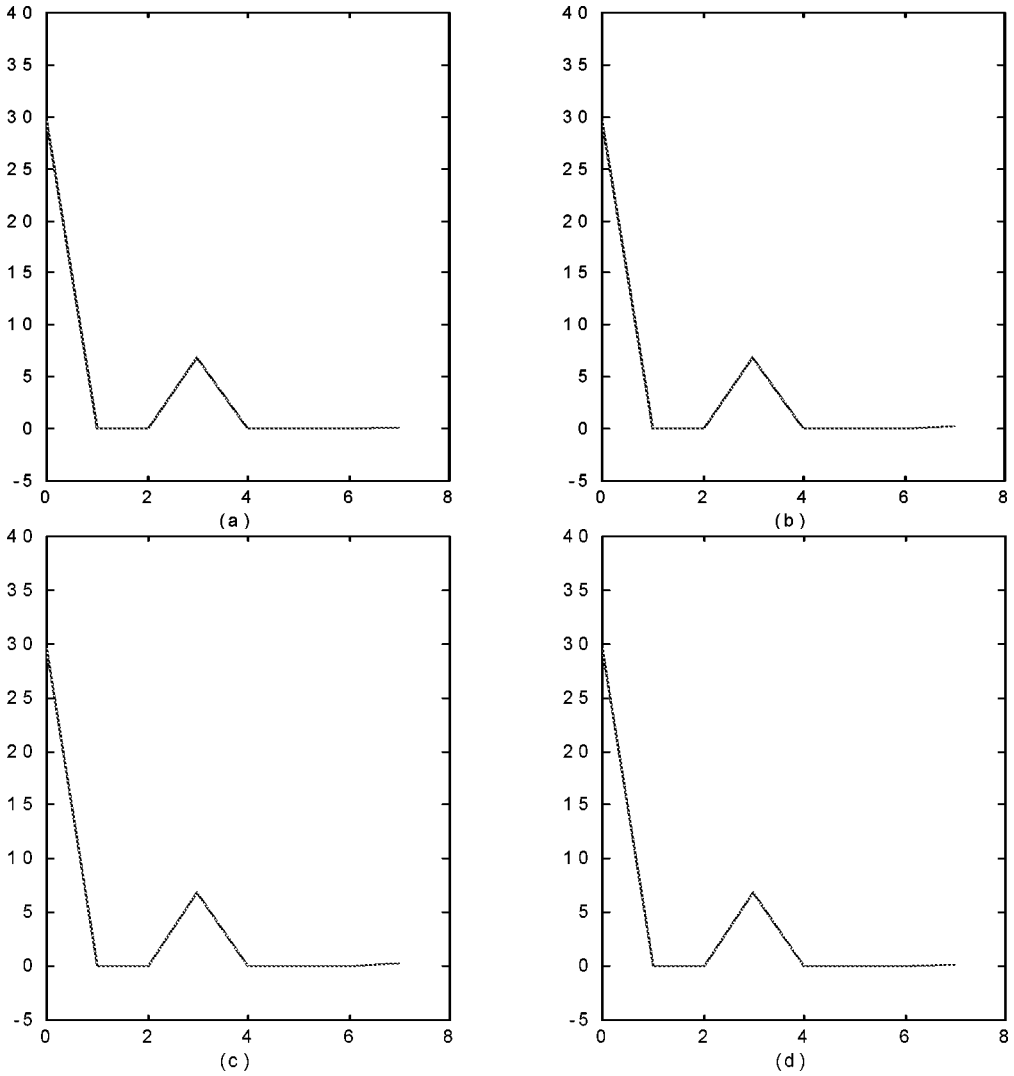

Fig. 10. Space-invariant property. (a), (b) Amplitude of the transform results of approximated DFT. (c), (d) Amplitude of the transform results of complete ITFT.

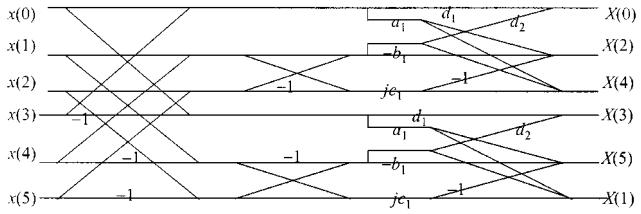

Fig. 11. Implementation of 6-point integer Fourier transform.

where $\text{cas}(x) = \cos(x) + \sin(x)$. The transform matrix of the DHT is just the summation of the real part and imaginary part of DFT

$$H(m,n) = \text{Re}(F(m,n)) - \text{Im}(F(m,n)). \tag{78}$$

The basis vectors of the DHT are orthogonal

$$\sum_{n=0}^{N-1} H(m,n) \cdot H(k,n) = N \cdot \delta_{m,k}.$$

The applications of the DHT are similar to the applications of the DFT, and the most important advantage of the DHT is when the input is pure real; then, the output is also pure real.

In this subsection, we will derive the integer Hartley transform (ITHT). We will not follow the process introduced in Section II. Instead, we will just derive the near-complete ITHT as

$$HI(m,n) = \text{Re}(FI(m,n)) - \text{Im}(FI(m,n)) \tag{79}$$

where $FI(m,n)$ is the near complete ITFT. We can just derive the complete ITHT as

$$\begin{aligned} &\text{forward ITHT: } HI(m,n) \\ &\quad = \text{Re}(FI(m,n)) - \text{Im}(FI(m,n)) \qquad (80) \\ &\text{inverse ITHT: } IHI(m,n) \\ &\quad = \text{Re}(IFI(m,n)) - \text{Im}(IFI(m,n)) \qquad (81) \end{aligned}$$

where $FI(m,n)$ is the forward complete ITFT, and $IFI(m,n)$ is the inverse complete ITFT. Here, we have constrained that the ITFT we derive should keep the conjugation system property

$$\begin{aligned} FI(m,n) &= \overline{FI(N-m,n)} \\ IFI(m,n) &= \overline{IFI(N-m,n)}. \end{aligned} \tag{82}$$

The goal of the integer transform is to keep the abilities of the original transform, but no real number multiplication operation is required. If we define the ITHT as (79) or (80) and (81), then the abilities of the DHT will be kept. That is, as the original

DHT, the ITHT defined as (79) or (80) and (81) will be another form of the ITFT, but for the real input, the output will also be real. Therefore, the ITHT may replace the ITFT for processing the real signals.

We then prove that the ITHT is reversible. We prove that the complete ITHT is reversible. From (80) and (81)

$$\begin{aligned}&\sum_{m=0}^{N-1} HI(m,n)IHI(m,k)\\&\quad=\sum_{m=0}^{N-1}\mathrm{Re}(FI(m,n))\mathrm{Re}(IFI(m,k))\\&\qquad+\sum_{m=0}^{N-1}\mathrm{Im}(FI(m,n))\mathrm{Im}(IFI(m,k))\\&\qquad-\sum_{m=0}^{N-1}\mathrm{Im}(FI(m,n))\mathrm{Re}(IFI(m,k))\\&\qquad-\sum_{m=0}^{N-1}\mathrm{Re}(FI(m,n))\mathrm{Im}(IFI(m,k)).\end{aligned}$$

Since the basis vectors of ITFT are orthogonal, we have

$$\begin{aligned}&\sum_{m=0}^{N-1} FI(m,n)\overline{IFI(m,k)}\\&\quad=\sum_{m=0}^{N-1}\mathrm{Re}(FI(m,n))\mathrm{Re}(IFI(m,k))\\&\qquad+\sum_{m=0}^{N-1}\mathrm{Im}(FI(m,n))\mathrm{Im}(IFI(m,k))\\&\qquad+j\sum_{m=0}^{N-1}\mathrm{Im}(FI(m,n))\mathrm{Re}(IFI(m,k))\\&\qquad-j\sum_{m=0}^{N-1}\mathrm{Re}(FI(m,n))\mathrm{Im}(IFI(m,k))\\&\quad=N\cdot\delta_{n,k}.\end{aligned}\tag{83}$$

Thus

$$\begin{aligned}&\sum_{m=0}^{N-1}\mathrm{Re}(FI(m,n))\mathrm{Re}(IFI(m,k))\\&\qquad+\sum_{m=0}^{N-1}\mathrm{Im}(FI(m,n))\mathrm{Im}(IFI(m,k))\\&\quad=N\cdot\delta_{n,k}\end{aligned}\tag{84}$$

$$\begin{aligned}&\sum_{m=0}^{N-1}\mathrm{Im}(FI(m,n))\mathrm{Re}(IFI(m,k))\\&\qquad-\sum_{m=0}^{N-1}\mathrm{Re}(FI(m,n))\mathrm{Im}(IFI(m,k))\\&\quad=0\end{aligned}\tag{85}$$

and (83) becomes

$$\begin{aligned}&\sum_{m=0}^{N-1} HI(m,n)IHI(m,k)\\&\quad=N\cdot\delta_{n,k}-2\sum_{m=0}^{N-1}\mathrm{Im}(FI(m,n))\mathrm{Re}(IFI(m,k)).\end{aligned}\tag{86}$$

Then since, for the ITFT, we have kept the conjugation symmetry property of (82) so that

$$\begin{aligned}\mathrm{Re}(FI(m,n))&=\mathrm{Re}(FI(N-m,n))\\\mathrm{Im}(FI(m,n))&=-\mathrm{Im}(FI(N-m,n))\end{aligned}\tag{87}$$

$$\begin{aligned}\mathrm{Re}(IFI(m,n))&=\mathrm{Re}(IFI(N-m,n))\\\mathrm{Im}(IFI(m,n))&=-\mathrm{Im}(IFI(N-m,n))\end{aligned}\tag{88}$$

we have

$$\begin{aligned}&\sum_{m=0}^{N-1}\mathrm{Im}(FI(m,n))\mathrm{Re}(IFI(m,k))\\&\quad=\sum_{m=0}^{[N/2]-1}\mathrm{Im}(FI(m,n))\mathrm{Re}(IFI(m,k))\\&\qquad+\sum_{m=0}^{[N/2]-1}\mathrm{Im}(FI(N-m,n))\mathrm{Re}(IFI(N-m,k))\\&\quad=\sum_{m=0}^{[N/2]-1}\mathrm{Im}(FI(m,n))\mathrm{Re}(IFI(m,k))\\&\qquad-\sum_{m=0}^{[N/2]-1}\mathrm{Im}(FI(m,n))\mathrm{Re}(IFI(m,k))=0\end{aligned}\tag{89}$$

and (86) becomes

$$\sum_{m=0}^{N-1} HI(m,n)IHI(m,k)=N\cdot\delta_{n,k}.\tag{90}$$

From (27) and (79), we can derive the prototype of the transform matrix of 8-point near-complete ITHT as (91), shown at the bottom of the page. The constraints are the same as (28).

$$\mathbf{HI_P}=\begin{bmatrix}1&1&1&1&1&1&1&1\\a_1&2a_2&a_1&0&-a_1&-2a_2&-a_1&0\\1&1&-1&-1&1&1&-1&-1\\c_1&0&-c_1&2c_2&-c_1&0&c_1&-2c_2\\1&-1&1&-1&1&-1&1&-1\\c_1&-2c_2&c_1&0&-c_1&2c_2&-c_1&0\\1&-1&-1&1&1&-1&-1&1\\a_1&0&-a_1&-2a_2&-a_1&0&a_1&2a_2\end{bmatrix}.\tag{91}$$

Similarly to the near-complete ITFT, we can choose the values of $\{a_1, a_2, c_1, c_2\}$ as the values listed in Table II.

For the 8-point complete ITHT, the prototype of the forward transform matrix is the same as (91). From (31) and (81), we can derive the prototype of the inverse transform matrix of the 8-point complete ITHT as (92), shown at the bottom of the page. The values of $\{a_1, a_2, c_1, c_2, a_3, a_4, c_3, c_4\}$ must also satisfy the constraints of (32)–(34). We can also choose the values of the $\{a_1, a_2, c_1, c_2, a_3, a_4, c_3, c_4\}$ as in Table III.

We draw the implementation of the forward, inverse ITHT as Figs. 12 and 13.

For the original DHT and complete ITHT, the multiplication operations we required are

original DHT:

four real number, eight fixed-point multiplication operations

complete ITHT:

20 fixed-point multiplication operations.

Although there are fewer multiplication operations for the original DHT, four of the multiplication operations are real numbers. For the complete ITHT, although there are more fixed-point multiplication operations, there are no real number multiplication operations. Thus, the complete integer Hartley transform will be simpler and faster.

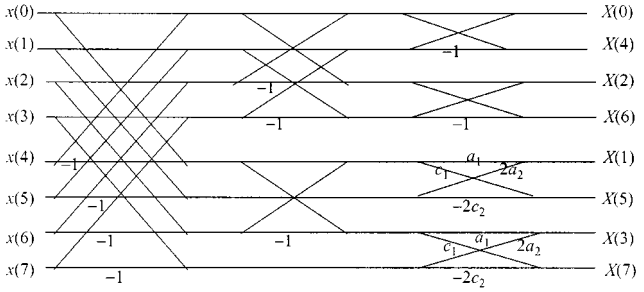

Fig. 12. Implementation of the forward 8-point complete integer Hartley transform.

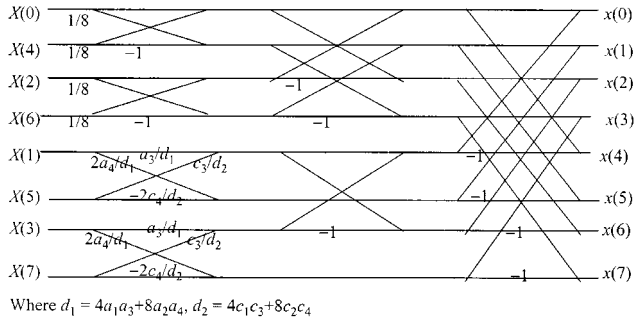

Fig. 13. Implementation of the inverse 8-point complete integer Hartley transform.

V. Integer Sine and Cosine Transforms

A. Integer Sine Transform of 8-Points

There are many types of DST [6]. In [2], they have derived the 8-point integer: even sine-1, even sine-2, even sine-3, odd sine-1, and odd sine-2 transforms. We describe the 8-point integer even sine-1 transform. Others can be seen in [2].

The original 8-point discrete even sine-1 transform is defined as

$$S(m,n) = \sin((m+1)\cdot(n+1)\pi/9) \quad \text{where } m,n \in [0,1,\ldots 7]. \tag{93}$$

In [2], they have derived the prototype of the 8-point integer even sine-1 transform (ITST-1) as

$$\mathbf{SI_P} = \begin{bmatrix} a_1 & a_2 & a_3 & a_4 & a_4 & a_3 & a_2 & a_1 \\ a_2 & a_4 & a_3 & a_1 & -a_1 & -a_3 & -a_4 & -a_2 \\ 1 & 1 & 0 & -1 & -1 & 0 & 1 & 1 \\ a_4 & a_1 & -a_3 & -a_2 & a_2 & a_3 & -a_1 & -a_4 \\ a_4 & -a_1 & -a_3 & a_2 & a_2 & -a_3 & -a_1 & a_4 \\ 1 & -1 & 0 & 1 & -1 & 0 & 1 & -1 \\ a_2 & -a_4 & a_3 & -a_1 & -a_1 & a_3 & -a_4 & a_2 \\ a_1 & -a_2 & a_3 & -a_4 & a_4 & -a_3 & a_2 & -a_1 \end{bmatrix}. \tag{94}$$

There are four unknowns a_1, a_2, a_3, a_4. The constraints of the parameters are

$$\begin{aligned} &(1)\ a_1 + a_2 = a_4, \quad (2)\ a_1a_4 + a_2a_4 = a_1a_2 + a_3^2 \\ &(3)\ a_1 \le a_2 \le a_3 \le a_4. \end{aligned} \tag{95}$$

There are many possible choices of the values of these unknowns (see [2, Tab. II]). One example is $\{a_1 = 3, a_2 = 5, a_3 = 7, a_4 = 8\}$ (This is the example where the parameters are smallest).

$$\mathbf{IHI_P} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ a_3 & 2a_4 & a_3 & 0 & -a_3 & -2a_4 & -a_3 & 0 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ c_3 & 0 & -c_3 & 2c_4 & -c_3 & 0 & c_3 & -2c_4 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ c_3 & -2c_4 & c_3 & 0 & -c_3 & 2c_4 & -c_3 & 0 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ a_3 & 0 & -a_3 & -2a_4 & -a_3 & 0 & a_3 & 2a_4 \end{bmatrix}. \tag{92}$$

The implementation of the 8-point ITST-1 has not been discussed in the previous works. We draw the implementation of the 8-point ITST-1 as Fig. 14.

The implementation of the 8-point ITST-1 requires a total of eight fixed-point multiplication operations. Since the 8-point ITST-1 introduced above is a near-complete integer transform, there are still eight real number multiplication operations required for the inverse transform. It is very hard to derive the complete integer sine transform.

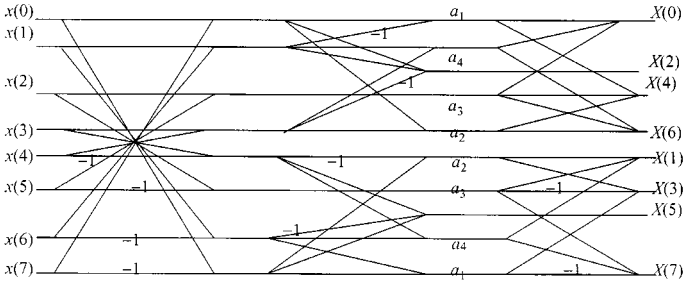

Fig. 14. Implementation of the 8-point ITST-1.

B. Integer Cosine Transform of 8-Points

In [1], they have derived the integer cosine transform (ITCT), which approximates the 8-point discrete cosine transform

$$C(m,n) = \cos(m \cdot (n+0.5)\pi/8) \quad \text{where } m,n \in [0,1,\ldots 7]. \quad (96)$$

The prototype matrix of ITCT is

$$\mathbf{CI_P} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ a_1 & a_2 & a_3 & a_4 & -a_4 & -a_3 & -a_2 & -a_1 \\ b_1 & b_2 & -b_2 & -b_1 & -b_1 & -b_2 & b_2 & b_1 \\ a_2 & -a_4 & -a_1 & -a_3 & a_3 & a_1 & a_4 & -a_2 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ a_3 & -a_1 & a_4 & a_2 & -a_2 & -a_4 & a_1 & -a_3 \\ b_2 & -b_1 & b_1 & -b_2 & -b_2 & b_1 & -b_1 & b_2 \\ a_4 & -a_3 & a_2 & -a_1 & a_1 & -a_2 & a_3 & -a_4 \end{bmatrix}. \quad (97)$$

The constraints are

$$(1)\ a_1a_2 - a_2a_4 - a_1a_3 - a_3a_4 = 0$$
$$(2)\ a_1 \geq a_2 \geq a_3 \geq a_4, \quad (3)\ b_1 \geq b_2. \quad (98)$$

Other details about ITCT can be seen in [1] and [2]. The 8-point ITCT can be implemented as Fig. 15.

In [1], they have also discussed the implementation of the 8-point integer cosine transform. In the above, we use an alternative method to implement it, and only the fixed-point multiplication operations are required. As in [1], we set $b_2 = 1$. Here, we try to make the number of multiplication operations as small as possible. If we implement the 8-point ITCT as in Fig. 15, there are 12 integer multiplication operations required for both the forward and inverse transform, but there are still six real number multiplication operations required for the inverse transform.

After the 8-point integer cosine transform (ITCT) defined in (97) has been derived in [1], in [3] and [4], the 16-point ITCT has been derived. In [2], the 8-point integer even cosine-2, odd cosine-1 transforms have been derived, and the ITCT defined as (97) is treated as the 8-point integer even cosine-1 transform.

We can also derive the complete integer cosine transform and do not require any real number multiplication operations, no matter what the forward or inverse transform. We also use (97) as the prototype for the forward ITCT. For the inverse ITCT,

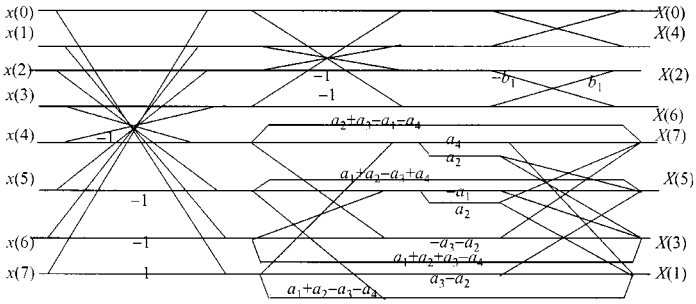

Fig. 15. Implementation of 8-point integer cosine transform.

we also use the same form of prototype, but the parameters are changed as in

$$\mathbf{ICI_P} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ c_1 & c_2 & c_3 & c_4 & -c_4 & -c_3 & -c_2 & -c_1 \\ d_1 & d_2 & -d_2 & -d_1 & -d_1 & -d_2 & d_2 & d_1 \\ c_2 & -c_4 & -c_1 & -c_3 & c_3 & c_1 & c_4 & -c_2 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ c_3 & -c_1 & c_4 & c_2 & -c_2 & -c_4 & c_1 & -c_3 \\ d_2 & -d_1 & d_1 & -d_2 & -d_2 & d_1 & -d_1 & d_2 \\ c_4 & -c_3 & c_2 & -c_1 & c_1 & -c_2 & c_3 & -c_4 \end{bmatrix}. \quad (99)$$

Thus, there are a total of 12 parameters $\{a_1, a_2, a_3, a_4, b_1, b_2\}$ $\{c_1, c_2, c_3, c_4, b_3, b_4\}$. Then, from the requirement of dual orthogonality, we obtain the constraints as

$$(1)\ a_1c_2 - a_2c_4 - a_3c_1 - a_4c_3 = 0$$
$$(2)\ a_2c_1 - a_4c_2 - a_1c_3 - a_3c_4 = 0. \quad (100)$$

From the requirement that the inner product must be the value of 2^k, we obtain

$$(3)\ a_1c_1 + a_2c_2 + a_3c_3 + a_4c_4 = 2^k$$
$$(4)\ b_1b_3 + b_2b_4 = 2^h \quad (101)$$

where k, h are integer numbers. We keep the two inequality constraints in (98) as

$$(5)\ a_1 \geq a_2 \geq a_3 \geq a_4, \quad (6)\ b_1 \geq b_2. \quad (102)$$

For flexibility, we omit the inequality constraints required for $\{c_1, c_2, c_3, c_4, b_3, b_4\}$. Thus, we obtain four equality constraints and two inequality constraints.

Although there are 12 parameters and four constraints, it seems that there would be many possible solutions for the

values of the parameters. However, these solutions are hard to find. We introduce a special method to find the solutions. We set $c_4 = 0$, and constraints 1, 2, and 3 become

$$\begin{bmatrix} -a_3 & a_1 & -a_4 \\ a_2 & -a_4 & -a_1 \\ a_1 & a_2 & a_3 \end{bmatrix} \cdot \begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 2^k \end{bmatrix} \quad \text{where } k \text{ is an integer.} \tag{103}$$

Therefore, we want

$$\det \begin{bmatrix} -a_3 & a_1 & -a_4 \\ a_2 & -a_4 & -a_1 \\ a_1 & a_2 & a_3 \end{bmatrix} = \pm 2^n \quad \text{where } n \text{ is an integer.} \tag{104}$$

Then, we can assure the solution of $\{c_1, c_2, c_3\}$ will all be the values of the form as $D \cdot 2^{-h}$, where D, h are integer numbers. We then want

$$-a_1^3 - a_2^2 a_4 + a_3^2 a_4 - 2a_1 a_2 a_3 - a_1 a_4^2 = \pm 2^n \quad \text{where } n \text{ is an integer.} \tag{105}$$

The values of $\{a_1, a_2, a_3, a_4\}$ that satisfy (105) are hard to find. Therefore, we suppose

$$a_1 : a_3 = a_3 : a_4, \quad \text{i.e., } a_1 = r^2 a_4, \quad a_3 = r \cdot a_4. \tag{106}$$

Then, (38) becomes

$$-r^6 a_4^3 - a_2^2 a_4 - 2r^3 a_2 a_4 = \pm 2^n \quad \text{where } n \text{ is an integer.} \tag{107}$$

We further suppose that $a_4 = 2^s$; therefore, the requirement becomes

$$r^3 a_4 + a_2 = 2^m \quad \text{where } m = (n - s)/2 \text{ and must be an integer.} \tag{108}$$

Thus, we can do the following process to choose the parameters.

a) First, find the value of $\{b_1, b_2, b_3, b_4\}$ to satisfy constraint 4. Since, for the original cosine transform, $b_1 = 0.9238, b_2 = 0.3827$, and $b_1 : b_2 = 2.4142 : 1$, we can choose $b_1 = 5$ and $b_2 = 2$ and substitute them into constraint 4 to find the values of b_3, b_4.
b) Set $c_4 = 0$ and $a_4 = 2^s$, where s is an integer number.
c) Choose $r (r > 1)$, and set a_1, a_3 as

$$a_1 = r^2 a_4, \quad a_3 = r \cdot a_4, \quad a_1, a_3 \text{ must be an integer.} \tag{109}$$

d) Calculate a_2 from

$$a_2 = 2^m - r^3 a_4, \quad m \text{ is an integer.} \tag{110}$$

e) Check whether $a_1 \geq a_2 \geq a_3$. If not, go back to step b).
f) Solve c_1, c_2, c_3 from (103).

Then, we can obtain the values of all the parameters. However, we must be careful to satisfy the inequality constraints.

We list some other possible choices of the values of parameters as Table IV.

We note that the forward ITCT can be implemented as Fig. 15. The inverse ITCT can also be implemented in Fig. 15, but the direction is reversed with some multiplication operations at the input. There are 12 fixed-point multiplication operations both for the forward and inverse transform. There are eight multiplication operations for D_m^{-1}, where D_m is defined as

$$\sum_{n=0}^{N-1} CI(m,n) \cdot ICI(m,n) = D_m. \tag{111}$$

Four of the multiplications of D_m can be absorbed in the multiplication operations of the inverse transform; therefore, we need a total of 28 fixed-point multiplication operations for the overall system.

TABLE IV
SOME POSSIBLE CHOICES OF THE VALUES OF THE PARAMETERS OF THE 8-POINT COMPLETE ITCT

a_1	a_2	a_3	a_4	b_1	b_2	b_3	b_4	c_1	c_2	c_3	c_4
81/16	295/64	9/4	1	5	2	6	1	6817/16	256	21591/64	0
49/16	169/64	7/4	1	5	2	22	9	2657/16	128	6489/64	0
121/64	717/512	11/8	1	12	5	9	4	149896	131072	41701	0
361/64	1333/512	19/8	1	17	7	13	5	1075336	524288	403389	0

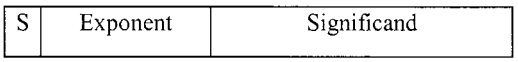

Fig. 16. Data structure including the sign, mantissa, and exponent terms.

VI. DISCUSSION ABOUT THE INTEGER TRANSFORMS

A. Advantages of the Integer Transform over Direct Approximation

In this paper, we use the integer transforms to approximate the noninteger transforms. In fact, we can also approximate the noninteger transform directly. For example, we can use the following data structure, including the mantissa and exponent terms to simulate real number operations by the fixed-point operations numerically [10] as in Fig. 16. In Fig. 16, "S" is the sign of the number, "Exponent" is the exponent term, and "Significand" is the mantissa term. For example, when we use the above structure to express the real value of $-C$, we can first approximate $-C$ as the form of

$$-C \approx -B \cdot 2^h \quad B \text{ is a positive integer number, } h \text{ is an integer number.} \tag{112}$$

Then, we can use "$S = 1$" to express the minus sign, use "Exponent $= h$" to express the exponent of 2, and use "Significand $= B$" to express the mantissa term. After using the above data structure, we can use the fixed-point operations to simulate the real number operations.

We may ask what are the advantages of the integer transform over the direct approximation method. We list the advantages below. We will just compare the direct approximation method with the near-complete integer transform.

1) The basis vectors of the integer transform are orthogonal.
We note that in Section II-A, we have discussed the four requirements for the integer transform. We note that requirements a)–c) are also necessary for the direct approximation method. The key difference between the integer transform and the direct approximation method is requirement d). That is, for the complete integer transform, the basis vectors of the forward transform is dual orthogonal to the basis vectors of the inverse transform. However, for the direct approximation method, the basis

vectors of the transform matrix may not be orthogonal. This difference is very important, and many of the advantages of the integer transform come from it.

2) The integer transform is reversible, but the transform obtained by direct approximation will not be reversible.

Since the complete integer transform is dual orthogonal, i.e., the basis vectors of the forward transform matrix $\mathbf{B}$ are dual orthogonal to the basis vectors of some matrix $\mathbf{E}$, then $\mathbf{E^H D^{-1}}$, where $\mathbf{D}$, which is defined as (12) and (14), is the inverse of the forward transform.

However, for the direct approximation method, since the basis vectors are not orthogonal, if the forward transform matrix is $\mathbf{K}$, then the inverse of $\mathbf{K}$ (i.e., $\mathbf{K^{-1}}$) will not be as simple as the inverse of the integer transform. Furthermore, the entries of $\mathbf{K^{-1}}$ can usually just be approximated and not explicitly expressed as the data structure as Fig. 16. Thus, although the basis vectors of the original transform (which are denoted by $\mathbf{F}$) are orthogonal, when we use the data structure as Fig. 16, we will use a transform matrix $\mathbf{K}$ to approximate $\mathbf{F}$ and use another transform matrix $\mathbf{R}$ to approximate $\mathbf{F^{-1}}$. Since

$$(\mathbf{F^{-1}}) \cdot \mathbf{F} = \mathbf{I}, \quad \mathbf{K} \approx \mathbf{F}, \quad \mathbf{R} \approx \mathbf{F^{-1}} \quad \mathbf{R} \cdot \mathbf{K} \neq \mathbf{I} \tag{113}$$

we use the data structure as Fig. 16 to approximate a reversible transform; then, the approximated transform would not be reversible.

The reversible property is very important, especially for the applications such as the filter design, data compression, and other applications that must do both the forward and inverse transforms. Although, for the directly approximation method, if the number of the bits are sufficient, then the approximated transform will be almost reversible. However, for the integer transform, we can use far fewer bits to obtain the reversible transform.

3) The integer transform can use fewer bits to obtain the accurate approximation results.

This is because when we use the data structure as in Fig. 16, some bits must be used for the exponent. Besides, the length of the data structure in Fig. 16 is usually fixed to 16 or 32 bits [10], but for the integer transform, the approximation results are sufficient, even when we use fewer bits. For example, for the two experiments about the ITFT in Section III-C, the largest parameter we choose is $a_3 = 18$, and it only requires 5 bits. Other parameters require 4 bits or less. From the experiment results, we still get accurate approximation results.

4) For the integer transform, when the input is expressed as the fixed-point representation, we do not need to convert it as the real number representation in Fig. 16. We can process it without any real number processor during the overall system.

5) More properties will be kept for the integer transform.

For example, the power preservation property will be kept for the complete ITFT (see Section III-C), but this property will not be kept when we approximate the DFT directly.

In conclusion, even when we use fewer bits, the integer transforms can still approximate the original transforms well and retain the properties and performance of the original transform. If we use enough bits, then using the direct approximation method will also have the advantages described above. In this case, the advantages of the integer transform may not be so obvious as the case where we use fewer bits.

B. Which Set of Parameters Is the Best

From the discussion about the integer sine, cosine, Hartley, and Fourier transforms, we find that the number of the equality constraints is always less than the number of the parameters. Thus, there are always infinite possible choices for the parameters. Thus, we have a question: Among these choices of parameters, which one is the best? Here, we will discuss this question.

To discuss which set of the parameters is the best, we mainly consider two aspects.

1) *For the implementation, we want the parameters to be as small as possible*. As in Fig. 1, for binary implementation, the entire number is decomposed as

$$C = a_0 + a_1 \cdot 2 + a_2 \cdot 2^2 + a_3 \cdot 2^3 \cdots \quad \text{where } a_r = 0, \pm 1. \tag{114}$$

Thus, if the parameters we choose are small, then the number of a_n such that $a_n \neq 0$ will usually be less. Then, for the binary implementation, the number of binary-addition operations and binary-shifting operations will be less.

2) *For the performance, we want the ratios of the entries for each row of the integer transform matrix to be approximated to those of the original transform matrix.*

For example, for the 8-point ITFT that has the prototype (27), from (15), the original 8-point DFT will have the parameters a_1, a_2 as

$$a_1 = 1 \quad a_2 = 1.414. \tag{115}$$

Therefore, if we want the performance of the 8-point ITFT to be similar to the original, then we want

$$a_1 : a_2 \approx 1 : 1.414. \tag{116}$$

For the complete integer transform, it is difficult for the parameters of the forward and inverse transforms to have the ratios approximated to the original at the same time. In this case, the parameters of the forward transform will have higher priority. If the parameters of the forward transform have ratios similar to the original transform, then the transform result will be similar to the original transform.

In [1], the tradeoff between how large the values of unknowns we choose and the performance for the integer cosine transform is discussed. In fact, this tradeoff also exists for other integer transforms. If the unknowns we choose are very large, than it is possible for the integer transforms to better approximate the original transforms and make the performance better; however, the implementation will be tough. On the other hand, if the values of unknown are too small, then the implementation is easier, but the performance is usually worse.

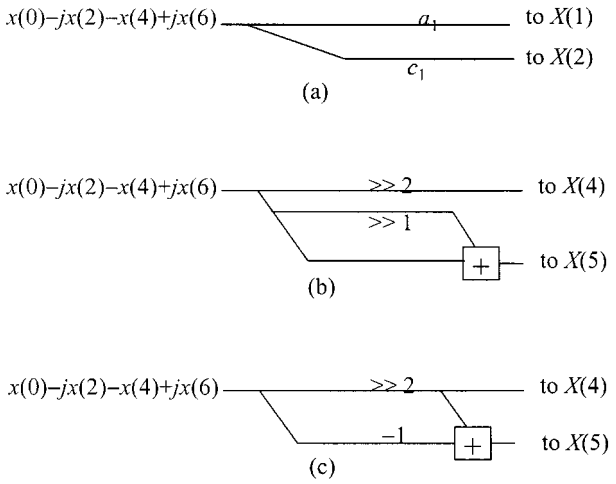

Fig. 17. (a) Implementation of 8-point ITFT (decimation-in-frequency) between the fifth position of the second stage and the fifth and sixth position of the third stage. (b) Binary implementation of Fig. 17(a) when $a_1 = 4$ and $c_1 = 3$. (c) Simplification of (b).

To find the optimal choices for the parameters is hard work. We suggest a way to find the better choices for the parameters.

1) First, find the ratios between the parameters for each row of the original transform matrix.
2) Then, assign the smaller integer values for the parameters such that the ratios are approximated well, and all the constraints for the parameters are satisfied.

   Then, although we may not obtain the optimal choices for the parameters, the integer transform we obtain will perform better with simpler implementation.

C. Some Special Methods to Implement Further More Efficient

Suppose $C = D \cdot 2^h$, where D, h are integer numbers; then if we want to multiply a number C, we can first decompose C as the linear combination of the powers of 2:

$$C = \sum_r a_r \cdot 2^r \quad \text{where } a_r = 0, \pm 1. \tag{117}$$

For the integer transform, all the multiplication operations can be implemented by the method of (117), and all use the binary shifting and the addition operations. Sometimes, the amount of multiplication operations is huge, or the numbers we want to multiply are very large. In such a case, there will be a lot of binary shifting and addition operations required. We introduce some ways to decrease the number of addition and binary shifting operations as follows.

1) Decompose C properly so that in (117), the amount of a_r that $a_r \neq 0$ will be minimum.

   For example, we can decompose 15 as $15 = 1 \cdot 2^0 + 1 \cdot 2^1 + 1 \cdot 2^2 + 1 \cdot 2^3$ or $15 = -1 \cdot 2^0 + 1 \cdot 2^4$. However, the former has 4 a_rs not equal to 0, and the latter only has 2 a_rs not equal to 0. When multiplying 15, decomposing 15 as $-1 \cdot 2^0 + 1 \cdot 2^4$ will require less binary addition operations.
2) Some multiplication operations with the same multiplicand can be merged together.

   For example, for the 8-point ITFT implemented by decimation-in-frequency (Fig. 2), between the fifth position of the second stage and the fifth and sixth position of the third stage, there is the branch as Fig. 17(a). For the case that $a_1 = 4$ and $c_1 = 3$, when we implement the multiplication of 3 and 4 individually, then we must require two binary shifting operations and one addition operation, as in Fig. 17(b). In fact, we can implement these two multiplication operations together. We can decompose 3 as $3 = -1 + 2^2$, and then, there is one binary shifting operation that is common for the multiplication of 3 and 4. Therefore, when we implement the multiplication of 3 and 4 together, then one binary shifting is saved, as in Fig. 17(c).

VII. Conclusion

In Sections III–V, we only derive the integer cosine, sine, Hartley, and Fourier transform of eight points and the integer Fourier transform of six points. In fact, we can also derive the integer transforms analogous to other noninteger transforms in a similar way.

For the integer transform, we can replace the real number multiplication operations with the fixed-points multiplication operations. Especially for the complete integer transform, there are no real number multiplication operations required. Thus, the integer transform is very efficient. If the datum we want to process are all integer numbers, using the complete integer transform is especially efficient, such as in image and video processing. This is because during the whole process, there are no real number processors required.

The concept of integer transform analogous to the noninteger transform is very new, and there has been little about it until now. Because they are convenient for implementation and almost retain the quality of original noninteger transform; therefore, they can compete with the original discrete transform in many applications. We believe the integer transform will be very popular in the future.

References

[1] W. K. Cham, "Development of integer cosine transform by the principles of dyadic symmetry," *Proc. Inst. Elect. Eng.*, pt. 1, vol. 136, no. 4, pp. 276–282, Aug. 1989.
[2] W. K. Cham and P. P. C. Yip, "Integer sinusoid transforms for image processing," *Int. J. Electron.*, vol. 70, no. 6, pp. 1015–1030, 1991.
[3] W. K. Cham and Y. T. Chan, "An order 16-integer cosine transform," *IEEE Trans. Signal Processing*, vol. 39, pp. 1205–1208, May 1991.
[4] S. N. Koh, S. J. Huang, and H. K. Tang, "Development of order 16-integer transforms," *Signal Process.*, vol. 24, pp. 283–289, Sept. 1991.
[5] R. N. Bracewell, *The Hartley Transform*. New York: Oxford Univ. Press, 1986.
[6] A. K. Jain, "A sinusoid family of unitary transforms," *IEEE Trans. Pattern Anal. Machine Intel.*, vol. PAMI-4, pp. 356–365, Oct. 1979.
[7] W. K. Cham, C. S. Choy, W. K. Lam, and S. M. Chiang, "2-D integer cosine transform chip set and its application," *IEEE Trans. Consumer Electron.*, vol. 38, pp. 43–47, May 1992.
[8] T. C. J. Pang, C. S. O. Choy, C. F. Chan, and W. K. Cham, "A self-timed ICT chip for image coding," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 9, no. 6, pp. 856–860, 1999.
[9] W. K. Cham, "Integer sinusoid transform," *Adv. Electron. Phys.*, vol. 88, pp. 1–61, 1994.
[10] W. Stallings, *Computer Organization Architecture, Principle of Structure and Function*. New York: Macmillan, 1987.

Soo-Chang Pei (F'00) was born in Soo-Auo, Taiwan, R.O.C., in 1949. He received B.S.S.E. degree from National Taiwan University (NTU), Taipei, in 1970 and the M.S.S.E. and Ph.D. degrees from the University of California, Santa Barbara, in 1972 and 1975, respectively.

He was an engineering officer with the Chinese Navy Shipyard from 1970 to 1971. From 1971 to 1975, he was a Research Assistant with the University of California, Santa Barbara. He was the Professor and Chairman with the Electrical Engineering Department, Tatung Institute of Technology from 1981 to 1983 and with NTU from 1995 to 1998. Presently, he is a Professor with the Electrical Engineering Department, NTU. His research interests include digital signal processing, image processing, optical information processing, and laser holography.

Dr. Pei received the National Sun Yat Sen Academic Achievement Award in Engineering in 1984, the Distinguished Research Award from the National Science Council from 1990 to 1998, the Outstanding Electrical Engineering Professor Award from the Chinese Institute of Electrical Engineering in 1998, and the Academic Achievement Award in Engineering from the Ministry of Education in 1998. He has been President of the Chinese Image Processing and Pattern Recognition Society in Taiwan from 1996 to 1998 and is a member Eta Kappa Nu and the Optical Society of America.

Jian-Jiun Ding was born in 1973 in Pingdong, Taiwan, R.O.C. He received both the B.S. and M.S. degree in electrical engineering from National Taiwan University (NTU), Taipei, Taiwan, in 1995 and 1997, respectively. He is currently pursuing the Ph.D. degree under the supervision of Prof. S.-C. Pei with the Department of Electrical Engineering, NTU.

He is currently a Teaching Assistant with NTU. His current research areas include fractional and affine Fourier transforms, other fractional transforms, orthogonal polynomials, integer transforms, quaternion Fourier transforms, pattern recognition, fractals, and filter design.