IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received August 18, 2018, accepted September 18, 2018, date of publication September 24, 2018, date of current version October 25, 2018.

Digital Object Identifier 10.1109/ACCESS.2018.2871885

Fast Gray Code Kernel Algorithm for the Sliding Conjugate Symmetric Sequency-Ordered Complex Hadamard Transform

JIASONG WU[1,2,3], (Member, IEEE), FUZHI WU[1,2,3], ZHIFANG DONG[4], KAIWEN SONG[4], YOUYONG KONG[1,2,3], LOTFI SENHADJI[5,6,7], (Senior Member, IEEE), AND HUAZHONG SHU[1,2,3], (Senior Member, IEEE)

[1]Laboratory of Image Science and Technology, Key Laboratory of Computer Network and Information Integration, Southeast University, Ministry of Education, Nanjing 210096, China
[2]International Joint Research Laboratory of Information Display and Visualization, Southeast University, Ministry of Education, Nanjing 210096, China
[3]Centre de Recherche en Information Biomédicale Sino-Français, Nanjing 210096, China
[4]School of Electronic Science and Engineering, Southeast University, Nanjing 210096, China
[5]INSERM, U1099, 35000 Rennes, France
[6]Laboratoire Traitement du Signal et de l'Image, Université de Rennes 1, 35000 Rennes, France
[7]Centre de Recherche en Information Biomédicale Sino–Français, 35000 Rennes, France

Corresponding author: Jiasong Wu (jswu@seu.edu.cn)

This work was supported in part by the National Natural Science Foundation of China under Grant 61876037, Grant 31800825, Grant 61871117, Grant 61871124, Grant 61773117, Grant 31571001, and Grant 61572258, in part by the National Key Research and Development Program of China under Grant 2017YFC0107900, in part by the Qing Lan Project and the "333" Project under Grant BRA2015288, and in part by the Short-Term Recruitment Program of Foreign Experts under Grant WQ20163200398.

ABSTRACT A fast algorithm based on the gray code kernel (GCK) for computing the conjugate symmetric sequency-ordered complex Hadamard transform (CS-SCHT) in a sliding window is presented. The proposed algorithm computes the current projection value from the previously computed ones. In order to obtain the peculiar computation order of the projection values, we construct the CS-SCHT matrix tree and also introduce the α-related concept. The properties of the elements of the CS-SCHT matrix are also given for deriving the GCK sliding CS-SCHT algorithm. The proposed algorithm only needs $N/2+\log_2 N-2$ (or $\log_2 N-1$) multiplications with j and $4N-2$ (or $2N-1$) real additions for complex (or real) input data, which is more efficient than the block-based CS-SCHT and other existing sliding complex transform algorithms, such as the radix-4 sliding CS-SCHT algorithm, sliding FFT algorithm, and sliding DFT algorithm. A comparison of the proposed algorithm with other sliding transforms in terms of computation time is also presented to validate the theoretical results.

INDEX TERMS Fast algorithm, conjugate symmetric sequency-ordered complex Hadamard transform, gray code kernel, sliding algorithm.

I. INTRODUCTION

The discrete orthogonal transforms (DOTs), including discrete Fourier transform (DFT), discrete cosine transform (DCT), discrete Hartley transform (DHT), and Walsh-Hadamard transform (WHT), play an important role in the fields of digital signal processing, filtering and communications [1], [2]. Many fast algorithms have been developed for the computation of DFT and WHT (e.g., [3]–[8]). In the past two decades, special attention has been paid to the definition of the complex Hadamard transforms and the associated fast algorithms and applications [9]–[19]. For example, Rahardja and Falkowski proposed a family of unified complex Hadamard transforms (UCHTs) [9], which find their applications in multiple-valued logic design [10] and communications [11]. Aung *et al.* introduced the sequency-ordered complex Hadamard transform (SCHT) [12], which has been used for spectrum analysis [12], image watermarking [13], asynchronous CDMA system [14] and image retrieval [15]. Two block-based algorithms, namely, the radix-2 decimation-in-time (DIT) [12] and decimation-in-sequency (DIS) [16] algorithms, have been developed for fast computation of SCHT. More recently, based on the natural-ordered complex Hadamard transform (NCHT) [17], Aung *et al.* introduced a new transform named the conjugate symmetric

2169-3536 © 2018 IEEE. Translations and content mining are permitted for academic research only.
Personal use is also permitted, but republication/redistribution requires IEEE permission.
See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

SCHT (CS-SCHT) [18], which can be used as an alternative to DFT in some applications that require lower computational complexity, such as spectrum estimation, since the spectrum of CS-SCHT is more similar to that of DFT than to those of other real transforms, such as WHT [6]–[8]. A fast block-based DIS algorithm was also reported in [18] and [19]. Kyochi and Tanaka then proposed a general factorization method for CS-SCHT [20]. Pei *et al.* [21] proposed a conjugate symmetric discrete orthogonal transform, which is a generalized version of CS-SCHT. The applications of CS-SCHT to signal processing and image denoising were presented by Jabeen *et al.* [22], [23].

When dealing with the spectrum of a nonstationary process, such as a speech, radar, biomedical, or communication signal, we need to use the so-called sliding discrete orthogonal transform (sliding DOT) [24]–[62]. Many fast algorithms have been reported in the literature for computing sliding DOTs. These algorithms can generally be categorized into two types as follows: recursive methods [24], [25] and nonrecursive methods [26]–[62]. In the recursive methods, the sliding DOTs are implemented by frequency sampling structures. However, since the poles of their corresponding recursive filters lie exactly on the unit circle (or, in practice, close to the unit circle), these methods are very sensitive to round-off errors, which may result in filter instability [63]. The nonrecursive methods are more frequently utilized. They include 1) the structures of the radix-2 and radix-4 fast algorithms, namely, sliding FFT [26]–[31], sliding WHT [32]–[34], sliding SCHT [35], and sliding CS-SCHT [36], [37]; and 2) the first-, second-, and $N/4$-order shift properties of DOTs: sliding DFT [38]–[49], sliding DCT [50]–[55], sliding DHT [56], [57], sliding discrete fractional transforms [58], [59], and sliding WHT [60]–[62]. Recently, special attention has been paid to the fast computation of the sliding WHT [32]–[34], [60]–[62] due to the requirement of real-time pattern matching in many applications, such as video block motion estimation [62]. A fast algorithm for the sliding WHT was proposed in [32], which decomposes a length-N WHT into two length-$N/2$ WHTs plus $2N-2$ additions with a memory size of $N(\log_2 N - 1)$,. This algorithm was further improved by Ben-Artzi *et al.* [61] who proposed the gray code kernel (GCK) algorithm, which requires $2N$ additions and a memory size of $2N$. Ouyang and Cham [33] presented a more efficient algorithm for computing the sliding WHT, which computes the length-N WHT from one length-$N/4$ WHT and $3N/2 + 1$ additions with a memory size of $3N/2$ for real input data. More recently, by using the structures of the radix-2 and radix-4 DIS fast SCHT [12], [16] and CS-SCHT algorithms [18], [19], Wu *et al.* proposed some fast algorithms for computing the sliding SCHT [35] and sliding CS-SCHT [36], [37].

The computational complexities of various sliding transforms are shown in Table 5 of [37], from which we can see that the computational complexity of sliding DFT [39], [40] is higher than sliding FFT [26], [27] and the computational complexity of GCK sliding WHT [61] is higher than radix-2 and radix-4 sliding WHTs [32]–[34]. Two questions have then arisen: Does a GCK sliding CS-SCHT algorithm exist in parallel with both sliding DFT algorithm and GCK sliding WHT algorithm? If the answer is "yes", then is the computational complexity of GCK sliding CS-SCHT still higher than that of the radix-4 sliding CS-SCHT [37]? The paper answers the two questions.

The contributions of the paper include: (1) A fast GCK sliding CS-SCHT algorithm that is surprisingly more efficient than the radix-4 sliding CS-SCHT [37] in terms of computational complexity. This phenomenon is opposite to the sliding DFT case and the sliding WHT case; (2) As the GCK sliding CS-SCHT algorithm calculates the current projection value based on the previously computed ones, the computation order of the projection value is very important. In this paper, we find the computation order of the projection value and it is very different from those of the sliding DFT and the sliding WHT.

The paper is organized as follows. In Section II, preliminaries regarding the sliding CS-SCHTs are given. The construction of the CS-SCHT matrix tree is presented in Section III. We demonstrate some properties of the CS-SCHT matrix in Section IV. The proposed sliding CS-SCHT algorithm is described and a comparison of the results with those of other algorithms are provided in Section V and Section VI, respectively. Section VII concludes the paper.

II. PRELIMINARIES

In this section, we first give the generalized definition of sliding DOT, and then give the definition of sliding CS-SCHT.

Consider M input signal elements x_i, where $i = 0, 1, \ldots, M-1$, which are divided into overlapping windows of size $N (M > N)$, then, sliding DOT is defined as

$$y_N(k, i) = \sum_{l=0}^{N-1} x_{i+l} w_l \psi_N(k, l), \tag{1}$$

where w_l is a window function, and $\{\psi_N(k, l)\}$ is an orthogonal basis set. $y_N(k, i)$ represents the kth orthogonal transform projection value for the ith window. Eq. (1) can also be expressed as the following matrix-vector form

$$\mathbf{y}_N(i) = \mathbf{\Psi}_N \mathbf{W}_N \mathbf{x}_N(i), \tag{2}$$

where

$$\mathbf{\Psi}_N = \begin{bmatrix} \psi(0,0) & \psi(0,1) & \cdots & \psi(0,N-1) \\ \psi(1,0) & \psi(1,1) & \cdots & \psi(1,N-1) \\ \vdots & \vdots & \ddots & \vdots \\ \psi(N-1,0) & \psi(N-1,1) & \cdots & \psi(N-1,N-1) \end{bmatrix}, \tag{3}$$

$$\mathbf{W}_N = diag\begin{bmatrix} w_0 & w_1 & \cdots & w_{N-1} \end{bmatrix}, \tag{4}$$

$$\mathbf{x}_N(i) = \begin{bmatrix} x_i & x_{i+1} & \cdots & x_{i+N-1} \end{bmatrix}^T, \tag{5}$$

$$\mathbf{y}_N(i) = \begin{bmatrix} y_N(0,i) & y_N(1,i) & \cdots & y_N(N-1,i) \end{bmatrix}^T, \tag{6}$$

where $diag(.)$ denotes a diagonal matrix formed from its vector argument, T denotes the transpose operation, $\mathbf{\Psi}_N$ and $\mathbf{W}_N$ are orthogonal transform matrix and window matrix, respectively,

Let w_m be a rectangle windowing function, from (2), the forward and backward sliding CS-SCHTs for length-$N = 2^p, p \geq 1$, can be respectively defined as

$$\mathbf{y}_N(i) = \mathbf{H}_N \mathbf{x}_N(i), \tag{7}$$

$$\mathbf{x}_N(i) = \frac{1}{N}\mathbf{H}_N^H \mathbf{y}_N(i), \tag{8}$$

where $\mathbf{H}_N$ is the order-N CS-SCHT matrix and the superscript H denotes the Hermitian transpose operation. Similar to (1), we can also express (7) as follows

$$y_N(k,i) = \sum_{l=0}^{N-1} x_{i+l} h_N(k,l), \tag{9}$$

where $y_N(k,i)$ represents the kth CS-SCHT projection value for the ith window and $h_N(k,l)$ are the elements of $\mathbf{H}_N$ [18]

$$h_N(k,l) = (-1)^{\sum_{r=0}^{p-1} g_r l_r}(-j)^{\sum_{r=0}^{p-1} f_r l_r}, \quad 0 \leq k, l \leq 2^p - 1 \quad \text{and } p = \log_2 N, \tag{10}$$

TABLE 1. Binary representations of g_r and f_r ($N = 16$).

k	Binary $(k_3k_2k_1k_0)_2$	Bit Reversal $(\tilde{k}_3\tilde{k}_2\tilde{k}_1\tilde{k}_0)_2$	$(g_3g_2g_1g_0)_2$	$c(k)$	$c(k)/2$	Highest power of 2 $d(k)$	$(f_3f_2f_1f_0)_2$
0	0000	0000	0000	0	0	0	0000
1	0001	1000	1100	8	4	4	0100
2	0010	0100	0110	4	2	2	0010
3	0011	1100	1010	12	6	4	0100
4	0100	0010	0011	2	1	1	0001
5	0101	1010	1111	10	5	4	0100
6	0110	0110	0101	6	3	2	0010
7	0111	1110	1001	14	7	4	0100
8	1000	0001	0001	1	0.5	0	0000
9	1001	1001	1101	9	4.5	4	0100
10	1010	0101	0111	5	2.5	2	0010
11	1011	1101	1011	13	6.5	4	0100
12	1100	0011	0010	3	1.5	1	0001
13	1101	1011	1110	11	5.5	4	0100
14	1110	0111	0100	7	3.5	2	0010
15	1111	1111	1000	15	7.5	4	0100

where l_r is the rth bit of the binary representation of the decimal integer l, i.e., $(l)_{10} = (l_{p-1}, l_{p-2} \ldots, l_r \ldots, l_1, l_0)_2$. Since g_r and f_r are complicated, we explain them together with an example for $N = 16$, which is shown in Table 1. Note that g_r and f_r are shown in column 4 and column 8, respectively. Column 2 expresses decimal k in binary form, that is, $(k)_{10} = (k_3k_2k_1k_0)_2$; column 3 obtains $\left(\tilde{k}_3\tilde{k}_2\tilde{k}_1\tilde{k}_0\right)_2$ through bit reversal of $(k_3k_2k_1k_0)_2$, that is, $\left(\tilde{k}_3\tilde{k}_2\tilde{k}_1\tilde{k}_0\right)_2 = (k_0k_1k_2k_3)_2$; and column 4 obtains $(g_3g_2g_1g_0)_2$, whose rth bit value is g_r, by finding the gray code of $\left(\tilde{k}_3\tilde{k}_2\tilde{k}_1\tilde{k}_0\right)_2$. Note that gray code is a binary numeral system in which two successive values differ in only one bit. $c(k)$ in column 5 is the decimal expression of $\left(\tilde{k}_3\tilde{k}_2\tilde{k}_1\tilde{k}_0\right)_2$. Then, $c(k)$ is divided by 2 to obtain column 6. Column 7 shows that $d(k)$ is the highest power of 2 that is not larger than $c(k)/2$; for example, if $c(k)/2 = 6$, then 4 is the highest power of 2 that is not larger than c(k)/2. Column 8 obtains $(f_3f_2f_1f_0)_2$, whose rth bit value is f_r, through binary expression of decimal $d(k)$.

From (10), we have

$$\mathbf{H}_1 = [1], \quad \mathbf{H}_2 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix},$$

$$\mathbf{H}_4 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & j & -1 & -j \\ 1 & -1 & 1 & -1 \\ 1 & -j & -1 & j \end{bmatrix},$$

$$\mathbf{H}_8 = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & j & j & -1 & -1 & -j & -j \\ 1 & j & -1 & -j & 1 & j & -1 & -j \\ 1 & -1 & -j & j & -1 & 1 & j & -j \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & -1 & j & -j & -1 & 1 & -j & j \\ 1 & -j & -1 & j & 1 & -j & -1 & j \\ 1 & 1 & -j & -j & -1 & -1 & j & j \end{bmatrix}. \tag{11}$$

Let

$$\mathbf{H}_N = \left[\left(\mathbf{h}_N^r(0)\right)^T, \left(\mathbf{h}_N^r(1)\right)^T, \ldots, \left(\mathbf{h}_N^r(N-1)\right)^T\right]^T, \tag{12}$$

$$\mathbf{H}_N^{1/m} = \left[\mathbf{h}_N^c(0), \mathbf{h}_N^c(1), \ldots, \mathbf{h}_N^c(N/m-1)\right] = \left[\left(\mathbf{h}_N^{1/m}(0)\right)^T, \left(\mathbf{h}_N^{1/m}(1)\right)^T, \ldots, \left(\mathbf{h}_N^{1/m}(N-1)\right)^T\right]^T, \quad m = 2, 4, 8, \tag{13}$$

where $\mathbf{h}_N^r(k)$ and $\mathbf{h}_N^c(k), k = 0, 1, \ldots, N-1$, are the kth row and kth column of the CS-SCHT matrix, respectively, and $\mathbf{h}_N^{1/m}(k), k = 0, 1, \ldots, N-1$, is the kth row of $\mathbf{H}_N^{1/m}$.

From (7), (9), and (12), we have

$$y_N(k,i) = \mathbf{h}_N^r(k)\mathbf{x}_N(i) \quad \text{for } k = 0, 1, \ldots N-1; \quad i = 0, 1, \ldots, M-N, \quad N = 2^p, \quad p \geq 2, \tag{14}$$

where M is the length of the input data sequence.

For the real input data, $y_N(k,i)$ satisfies the following conjugate symmetry property:

$$y_N(N-k,i) = y_N^*(k,i), \quad k = 1, 2, \ldots, N/2-1, \tag{15}$$

where the subscript $*$ denotes complex conjugation.

Specifically, the idea of GCK sliding CS-SCHT algorithm is to compute $y_N(k', i+n)$ by using $y_N(k', i)$, $y_N(k, i+n)$ and $y_N(k,i)$, where $i+n$ and i denote the current window and the previous window, respectively. k' and k are sequency values obeying $k' \neq k$ and $0 \leq k', k \leq N-1$.

From (14), we have

$$\begin{cases} y_N(k',i) = \mathbf{h}_N^r(k')\mathbf{x}_N(i) \\ y_N(k,i) = \mathbf{h}_N^r(k)\mathbf{x}_N(i) \\ y_N(k',i+n) = \mathbf{h}_N^r(k')\mathbf{x}_N(i+n) \\ y_N(k,i+n) = \mathbf{h}_N^r(k)\mathbf{x}_N(i+n) \end{cases} \tag{16}$$

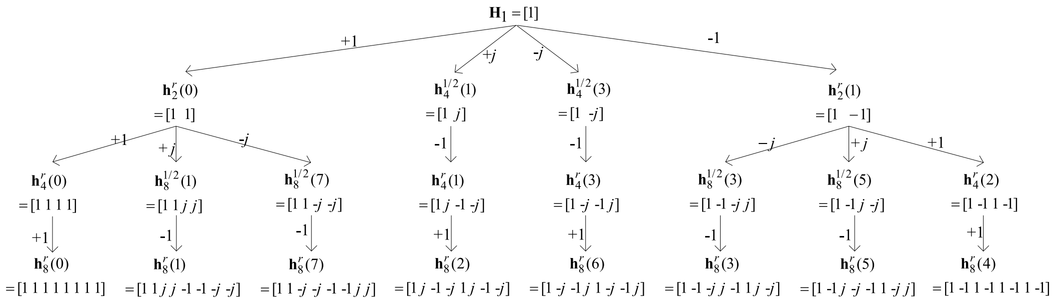

FIGURE 1. **Tree structure for the CS-SCHT matrix construction**

for $k', k = 0, 1, \ldots N-1$; $i = 0, 1, \ldots, M–N$, $N = 2^p$, $p \geq 2$.

From (16), we can see that if we want to express $y_N(k', i+n)$ in terms of $y_N(k, i+n)$, the most import point is to find the relation between $\mathbf{h}^r_N(k')$ and $\mathbf{h}^r_N(k)$. In order to obtain this relation, we should first find how to construct the CS-SCHT matrix in row format, which is shown in the following Section III.

III. THE CONSTRUCTION OF CS-SCHT MATRIX TREE

In [18], Aung *et al.* proposed a decomposition method for the CS-SCHT matrix. Wu *et al.* [36], [37] derived a new relationship between the CS-SCHT and WHT matrices. In this section, we present a novel method for constructing the CS-SCHT matrix, which is described in the following theorem.

Theorem 1: The CS-SCHT matrix can be constructed by

$$\mathbf{h}^r_N(k) = \begin{cases} \begin{bmatrix} 1 & 1 \end{bmatrix} \otimes \mathbf{h}^r_{N/2}(k/2), & k = 0, 2, 4, 6, \ldots, N-2; \\ \begin{bmatrix} 1 & -1 \end{bmatrix} \otimes \mathbf{h}^{1/2}_N(k), & k = 1, 3, 5, 7, \ldots, N-1; \end{cases} \tag{17}$$

$$\mathbf{h}^{1/2}_N(k) = \begin{cases} \begin{bmatrix} 1 & j \end{bmatrix} \otimes \mathbf{h}^{1/4}_N(k), \\ \quad k = 1, 5, 9, 13, \ldots, N-7, N-3; \\ \begin{bmatrix} 1 & -j \end{bmatrix} \otimes \mathbf{h}^{1/4}_N(k), \\ \quad k = 3, 7, 11, 15, \ldots, N-5, N-1; \end{cases} \tag{18}$$

$$\mathbf{h}^{1/4}_N(k) = \begin{cases} \begin{bmatrix} 1 & 1 \end{bmatrix} \otimes \begin{cases} \mathbf{h}^{1/4}_{N/2}((k+1)/2), \\ \quad k = 1, 9, 17, \ldots, N-7; \\ \mathbf{h}^{1/4}_{N/2}((k-1)/2), \\ \quad k = 7, 15, 23, \ldots, N-1; \end{cases} \\ \begin{bmatrix} 1 & -1 \end{bmatrix} \otimes \begin{cases} \mathbf{h}^{1/4}_{N/2}((k+1)/2), \\ \quad k = 5, 13, 21, \ldots, N-3; \\ \mathbf{h}^{1/4}_{N/2}((k-1)/2), \\ \quad k = 3, 11, 19, \ldots, N-5; \end{cases} \end{cases} \tag{19}$$

where $\otimes$ is the Kronecker product.

The proof of Theorem 1 is given in Appendix A. The tree structure of the order-8 CS-SCHT matrix construction is shown as an example in Fig. 1. In the CS-SCHT matrix tree, for the length-$N = 2^p$ CS-SCHT, we have $p+1$ layers nodes, which we call the 0th, 1th, …, mth, …, pth nodes, respectively. The values $\alpha_m \in \{1, -1, j, -j\}$ are shown on the left- or right-hand side of the arrows, which are between the mth and (m+1)th nodes.

From Theorem 1 and Figure 1, we can see that $\mathbf{h}^r_N(k')$ is related to $\mathbf{h}^r_N(k)$ only for some specific k' and k. How to find these k' and k?

Similarly to [61], with some modifications, we first introduce some definitions that are used in Section IV.

Definition 1: The sequence $\alpha = \alpha_1, \alpha_2, \ldots, \alpha_p$, with $\alpha_i \in \{+1, +j, -1, -j\}$, that uniquely defines a kernel $\mathbf{h}^r_N(k)$ is called the α**-index** of $\mathbf{h}^r_N(k)$, $k = 0, 1, \ldots, N-1$.

Definition 2: Two kernels $\mathbf{h}^r_N(k')$ and $\mathbf{h}^r_N(k)$, for $k', k = 0, 1, \ldots, N-1$ and $k' \neq k$, are α**-related** if their α-indices differ in only one value.

Definition 3: An ordered set of kernels $\mathbf{h}^r_N(k)$, for $k = 0, 1, \ldots, N-1$, that are consecutively α-related form a sequence of **Gray Code Kernels (GCKs)**. The sequence is called a **Gray Code Sequence (GCS)**.

In order to express more explicitly the α**-related** relation between $\mathbf{h}^r_N(k')$ and $\mathbf{h}^r_N(k)$, which are the row vectors of CS-SCHT matrix $\mathbf{H}_N$, we use $h_N(k, l)$ in (10), which are the elements of CS-SCHT matrix $\mathbf{H}_N$. Some of their useful properties are reported in the following section.

IV. PROPERTIES OF $h_N(k, l)$

In this section, we describe all the properties of $h_N(k, l)$, for $k, l = 0, 1, \ldots, N-1$. The relation between $h_N(k', l+n)$ and $h_N(k', l)$, $h_N(k, l)$, $h_N(k, l+n)$ exists only for some specific k' and k to be determined. Let us take $N = 16$ for example, from Eq. (10) and Table 1, we can see that k, especially its binary expression $(k_3k_2k_1k_0)_2$, is one-to-one correspondent to the binary expression $(g_3g_2g_1g_0)_2$ and $(f_3f_2f_1f_0)_2$. Therefore, the problem of finding some special k' and k is convert to finding some special cases of g_r and f_r shown in (10).

First, we find the relationships of the corresponding g_r (or f_r) for some special values of k. We mainly consider two relationships:

1) The relationships of g_r (or f_r) between $k = 0$ and $k = 1, 2, 4, \ldots, N/4, N/2, N - N/4, \ldots, N - 4, N - 2, N - 1$;

2) For other values of k, we focus on those that have the same value as f_r and for which the values of g_r satisfy the consecutive GCS.

a) When k is odd, we choose k to have the same value as f_r, while the corresponding values of g_r constitute a consecutive GCS in the following order in the computation of $h_N(k, l)$:

$$1 \to N-1 \to 3 \to N-3 \to 5 \to N-5 \to \cdots \to N/2-1 \to N/2+1; \quad (20)$$

Let us consider $N = 16$, for example. The corresponding values of g_r constitute a consecutive GCS:

$$1100 \to 1000 \to 1010 \to 1110 \to 1111 \to 1011 \to 1001 \to 1101$$

The α-indices of the corresponding $\mathbf{h}_N^r(k)$ also constitute a consecutive GCS:

$$\{1, 1, j, -1\} \to \{1, 1, -j, -1\} \to \{1, -1, -j, -1\} \to \{1, -1, j, -1\} \to \{-1, -1, j, -1\} \to \{-1, -1, -j, -1\} \to \{-1, 1, -j, -1\} \to \{-1, 1, j, -1\}.$$

b) When k is even, we choose k to have the same value as f_r, while the corresponding values of g_r constitute a consecutive GCS, in the following order, in the computation of $h_N(k, l)$:

$$\begin{cases} 2 \to N-2 \to 6 \to N-6 \\ \quad \to \cdots \to N/2-2 \to N/2+2 \\ 4 \to N-4 \to 12 \to N-12 \\ \quad \to \cdots \to N/2-4 \to N/2+4 \\ \ldots \\ N/4 \to 3N/4 \\ N/2. \end{cases} \quad (21)$$

Note that each row is computed separately. Let us again consider the case where $N = 16$. The corresponding values of g_r constitute three consecutive GCSs:

$$\begin{cases} 0110 \to 0100 \to 0101 \to 0111 \\ 0011 \to 0010 \\ 0001. \end{cases}$$

The corresponding α-indices of $\mathbf{h}_N^r(k)$ also constitute three consecutive GCSs:

$$\begin{cases} \{1, j, -1, 1\} \to \{1, -j, -1, 1\} \\ \quad \to \{-1, -j, -1, 1\} \to \{-1, j, -1, 1\} \\ \{j, -1, 1, 1\} \to \{-j, -1, 1, 1\} \\ \{-1, 1, 1, 1\}. \end{cases}$$

By combining (20) and (21), we can continuously compute $h_N(k, l)$ using the following order of k:

$$\begin{cases} 0 \to 1 \to N-1 \to 3 \to N-3 \to 5 \to N-5 \\ \quad \to \cdots \to N/2-1 \to N/2+1 \\ 0 \to 2 \to N-2 \to 6 \to N-6 \\ \quad \to \cdots \to N/2-2 \to N/2+2 \\ 0 \to 4 \to N-4 \to 12 \to N-12 \\ \quad \to \cdots \to N/2-4 \to N/2+4 \\ \ldots \\ 0 \to N/4 \to 3N/4 \\ 0 \to N/2. \end{cases} \quad (22)$$

Note that the order of k is determined by the order of the consecutive GCS of g_r.

From (22), we find the order of k' that may allow computing $h_N(k', l)$ from $h_N(k, l)$ for all the $k' = 0, 1, \ldots, N-1$. Let us take the first row of (22) for example, we should compute $h_N(1, l)$ from $h_N(0, l)$, and then compute $h_N(N-1, l)$ from $h_N(1, l)$, and then compute $h_N(3\, l)$ from $h_N(N-1, l)$, and so on. However, there are still two questions left:

The first one is: are there any relations between $h_N(1, l)$ and $h_N(0, l)$, between $h_N(N-1, l)$ and $h_N(1, l)$, between $h_N(3, l)$ and $h_N(N-1, l)$? If the answer is yes, then, we can compute $h_N(k', l)$ from $h_N(k, l)$ by using the order of k shown in (22) and thus compute $h_N(k, l)$ for all indexes $k = 0, 1, \ldots, N-1$ in sequency domain.

The second question is that from (22), besides the relation between $h_N(k', l)$ and $h_N(k, l)$, what are the relations between $h_N(k', l+n)$ and $h_N(k', l)$? Once they are known, we can implement the sliding process in time domain.

In order to deal with the above two questions, we derive the following three properties. The first question can be solved by **Property 1** and **Property 2**, and the second one can be solved by **Property 3**.

Property 1:

(a) $h_N(0, l) = 1;$ (23-1)

(b) $h_N(N/2, l) = (-1)^l;$ (23-2)

(c) $h_N(k, l) = j^{\lfloor l/n \rfloor}, \quad k = N/(4 \times n), \quad n = 1, 2, 4, 8, \ldots, N/4;$ (23-3)

Property 2:

$$h_N\left(k + (N - mN/(2n)), l\right) = (-1)^{\lfloor l/n \rfloor} h_N(k, l), \quad k = (m-1)N/(4n) + 1, (m-1)N/(4n) + 2, \ldots, (m+1)N/(4n) - 1; \quad m = 1, 3, 5, 7, \ldots, 2n-1; \quad n = 1, 2, 4, \ldots, N/8, N/4. \quad (24)$$

It can be deduced from (24) that

$$h_N(N-k, l) = (-1)^{\lfloor l/n \rfloor} h_N(k, l), k = mN/(4n); \quad m = 1, 3, 5, 7, \ldots, 2n-3, 2n-1; \quad n = 1, 2, 4, \ldots, N/8, N/4. \quad (25)$$

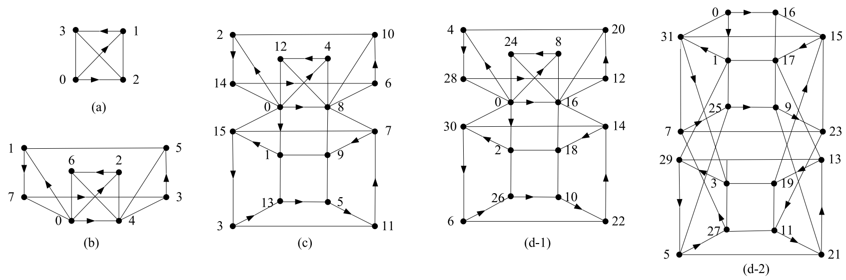

FIGURE 2. Relationship figure of $N = 4, 8, 16, 32$.

Property 3: The relationship between $h_N(k, l{+}n)$ and $h_N(k, l)$ for $l \in [0, n-1] \cup [2n, 3n-1] \cup \ldots \cup [N-2n, N-n-1]$, and the relationship between $h_N(k, l+N-n)$ and $h_N(k, l)$ are given by

$$\begin{aligned} &h_N(k, l+n) \\ &= (-1)^{(m-1)/2} h_N(k, l), \\ &\quad k = (m-1)N/(4n)+1, \ldots, mN/(4n)-1; \\ &\quad m = 1, 3, 5, \ldots, 2n-1; \quad n = 1, 2, 4, \ldots, N/8 \end{aligned} \tag{26-1}$$

$$\begin{aligned} &h_N(k, l+n) \\ &= -(-1)^{(m-1)/2} h_N(k, l), \\ &\quad k = mN/(4n)+1, \ldots, (m+1)N/(4n)-1; \\ &\quad m = 1, 3, 5, \ldots, 2n-1; \quad n = 1, 2, 4, \ldots, N/8 \end{aligned} \tag{26-2}$$

$$\begin{aligned} &\begin{cases} h_N(k, l+n) = j^m h_N(k, l), \\ h_N(k, l+N-n) = (-j)^m h_N(k, l), \end{cases} \\ &\quad k = mN/(4n); \\ &\quad m = 1, 3, 5, \ldots, 2n-1; \quad n = 1, 2, 4, \ldots, N/8, N/4 \\ &\qquad \text{or } m = 2, 4, 6, \ldots, 2n-2; \quad n = N/4 \end{aligned} \tag{26-3}$$

The proofs of Properties 1-3 are given in Appendix B.

V. GRAY CODE KERNEL ALGORITHM FOR SLIDING CS-SCHT

In this section, we present a GCK algorithm for computing the sliding CS-SCHT, that is, computing $y_N(k', l+n)$ from $y_N(k', l)$, $y_N(k, l+n)$, and $y_N(k, l)$. From (16), we can see that the relations between $y_N(k', l+n)$ and $y_N(k', l)$ can be derived from $h_N(k', l+n)$ and $h_N(k', l)$; the relations between $y_N(k', l)$ and $y_N(k, l)$ can be derived from $h_N(k', l)$ and $h_N(k, l)$. Note that the properties of $h_N(k, l)$ are shown in Section IV.

In this section, we show that $y_N(k', l+n)$ can be computed by the following three Theorems. Theorem 2 formulates the relationship between the 0th row and kth row ($k = 1, 2 \ldots, N/4$). Theorem 3 formulates the relationships between conjugate rows; that is, the kth row and (N-k)th row. Theorem 4 formulates other relationships that cannot be expressed by Theorem 2 or 3.

Theorem 2 (The 0th Row):

$$\text{(a) } y_N(0, i) - y_N(0, i+1) = y_N(N/2, i) + y_N(N/2, i+1); \tag{27-1}$$

$$\begin{aligned} \text{(b) } &y_N(0, i) - y_N(0, i+N/(4k)) \\ &= y_N(k, i) - j y_N(k, i+N/(4k)) \\ &\qquad \text{for } k = 1, 2, 4, \ldots, N/8, N/4. \end{aligned} \tag{27-2}$$

Theorem 3 (Conjugate Row):

$$\begin{aligned} &y_N(k, i) - j^m y_N(k, i+n) \\ &= y_N(N-k, i) + j^m y_N(N-k, i+n), \\ &\quad \text{for } k = mN/(4n), \quad m = 1, 3, 5, \ldots, 2n-1; \\ &\quad n = 1, 2, 4, \ldots, N/8, N/4 \end{aligned} \tag{28}$$

Theorem 4 (Other Relationships):

$$\begin{aligned} \text{(a) } &y_N(k, i) - (-1)^{(m-1)/2} y_N(k, i+n) \\ &= y_N(k + (N - mN/(2n)), i) \\ &\quad + (-1)^{(m-1)/2} y_N(k + (N - mN/(2n)), i+n), \\ &\quad \text{for } k = (m-1)N/(4n)+1, \ldots, mN/(4n)-1, \\ &\quad m = 1, 3, 5, \ldots, 2n-1; \quad n = 1, 2, 4, \ldots, N/8 \end{aligned} \tag{29-1}$$

$$\begin{aligned} \text{(b) } &y_N(k, i) + (-1)^{(m-1)/2} y_N(k, i+n) \\ &= y_N(k + (N - mN/(2n)), i) \\ &\quad - (-1)^{(m-1)/2} y_N(k + (N - mN/(2n)), i+n), \\ &\quad \text{for } k = mN/(4n)+1, \ldots, (m+1)N/(4n)-1, \\ &\quad m = 1, 3, 5, \ldots, 2n-1; \quad n = 1, 2, 4, \ldots, N/8 \end{aligned} \tag{29-2}$$

The proofs of Theorems 2-4 are given in Appendix C. Furthermore, the proof of Theorems 2 utilizes Property 1; the proof of Theorems 3 utilizes Properties 1-3; the proof of Theorems 4 utilizes Properties 2 and 3.

To visualize clearly the relationships in (27)-(29) between the rows, we provide in Figure 2 some examples for $N = 4$, 8, 16, 32. In Figure 2, i denotes the ith row, and a line between two numbers indicates that there is a relationship between the two corresponding rows. The arrow indicates the direction of computation that we have chosen. The paths indicate that all the rows have relationships between each other. However, not all of the relationships should be exploited in the following computation. For example, (29-1) is not used in our computation. The paths with arrows indicate that we exploit the relationship. The paths without arrows indicate that we do not use the relationship.

To simplify the expression of our sliding CS-SCHT algorithm, we should change the expression forms of (28) and (29-2).

Eq. (28) is equivalent to

$$\begin{aligned} &y_N(N-k,i)+j^{k/k_0}y_N(N-k,i+N/(4k_0))\\ &\quad=y_N(k,i)-j^{k/k_0}y_N(k,i+N/(4k_0)),\\ &k=k_0,3k_0,5k_0,\ldots,N/2-k_0;\\ &\qquad k_0=1,2,4,8,\ldots,N/8,N/4 \end{aligned} \tag{30}$$

The following equation can be derived from Eq. (29-2),

$$\begin{aligned} &y_N(k+2k_0,i)+(-1)^{(m-1)/2}y_N(k+2k_0,i+N/(8n))\\ &\quad=y_N(N-k,i)-(-1)^{(m-1)/2}y_N(N-k,i+N/(8n)),\\ &k=k_0,3k_0,5k_0,7k_0,\ldots,N/2-3k_0;\\ &n=1,2,4,8,\ldots,N/8;\ k_0=1,2,4,8,\ldots,N/8;\\ &m=(k+k_0)/(2n),\text{ and } m \text{ is positive odd integer} \end{aligned} \tag{31}$$

The proofs of (30) and (31) are given in Appendixes D and E, respectively.

In the process of computation, we choose the 0th projection as the starting point, and then compute as many other projections as possible. From the order of k in (22), we can calculate $y_N(k,i)$, for $k=1,2,3,\ldots,N-1$, of the length-N CS-SCHT from $y_N(0,i)$ as follows:

$$\begin{aligned} y_N(0,i)&\xrightarrow[T2]{+k_0}y_N(k_0,i)\xrightarrow[T3]{+(N-2k_0)}y_N(N-k_0,i)\\ &\xrightarrow[T4]{-(N-4k_0)}y_N(3k_0,i)\xrightarrow[T3]{+(N-6k_0)}y_N(N-3k_0,i)\\ &\xrightarrow[T4]{-(N-8k_0)}y_N(5k_0,i)\xrightarrow[T3]{+(N-10k_0)}y_N(N-5k_0,i)\\ &\cdots\cdots\\ &\xrightarrow[T3]{+10k_0}y_N(N/2+5k_0,i)\xrightarrow[T4]{-8k_0}y_N(N/2-3k_0,i)\\ &\xrightarrow[T3]{+6k_0}y_N(N/2+3k_0,i)\xrightarrow[T4]{-4k_0}y_N(N/2-k_0,i)\\ &\xrightarrow[T3]{+2k_0}y_N(N/2+k_0,i);\quad k_0=1,2,4,\ldots,N/8;\\ y_N(0,i)&\xrightarrow[T2]{}y_N(N/4,i)\xrightarrow[T3]{}y_N(3N/4,i);\\ y_N(0,i)&\xrightarrow[T2]{}y_N(N/2,i). \end{aligned} \tag{32}$$

Let us take $N=8$ for example. The computation order is as follows:

$$\begin{cases} y_8(0,i)\xrightarrow[T2]{+1}y_8(1,i)\xrightarrow[T3]{+6}y_8(7,i)\\ \quad\xrightarrow[T4]{-4}y_8(3,i)\xrightarrow[T3]{+2}y_8(5,i);\\ y_8(0,i)\xrightarrow[T2]{}y_8(2,i)\xrightarrow[T3]{}y_8(6,i);\\ y_N(0,i)\xrightarrow[T2]{}y_N(4,i). \end{cases} \tag{33}$$

The pseudocode of the algorithm is shown in Table 2, and the computation details of Length-4 and Length-8 CS-SCHT are shown in Table 3. The whole process of deriving the GCK sliding CS-SCHT algorithm for $N = 8$ is also given in supplement document for understanding the algorithm more intuitively.

TABLE 2. Pseudocode of the proposed GCK sliding CS-SCHT algorithm.

```
if k=1,2,4,8,…,N/4,
   y_N(k, i+N/(4k))=-j(y_N(k, i)-(y_N(0, i)-y_N(0, i+N/(4k))));  # Eq. (27-2)
if k=N/2,
   y_N(N/2, i+1)=-y_N(N/2, i)+y_N(0, i)-y_N(0, i+1);  # Eq.(27-1)
if k=3N/4,
   y_N(3N/4, i+1)=-(y_N(N/4, i+1)+ j(y_N(N/4, i)-y_N(3N/4, i)));  # Eq. (30)
for k_0=1,2,4,8,…,N/8,
   for k=k_0, 3k_0, 5k_0, …, N/2-3k_0, N/2-k_0,
      y_N(N-k, i+N/(4k_0))=-(y_N(k, i+N/(4k_0))+j^(-k/k0)(y_N(N-k, i)-y_N(k, i)));  # Eq. (30)
        if  k=N/2-k_0,
          continue;   # Exit the last loop of k
         else
          for n=1,2,4,8,…,N/8,
              if m=(k+k_0)/(2n) is a positive odd integer,
                 y_N(k+2k_0, i+N/(8n))=-(y_N(N-k, i+N/(8n))+(-1)^((m+1)/2) (y_N(N-k, i)-y_N(k+2k_0, i))); # Eq. (31)
          end
      end
end
```

VI. COMPLEXITY ANALYSIS AND COMPARISON RESULTS

The computational complexity of the proposed algorithm is analyzed as follows:

1. Additions: For each window in the computation of (32), we need 4 additions for a complex signal and 2 additions for a real signal. Thus, we need a total of $4N$ additions for a complex signal and $2N$ additions for a real signal for computing all k values of $y_N(k,i+N/4)$ from $y_N(k,i)$, for $k=0,1\ldots,N-1$.
2. Multiplication with j: For a complex signal, (27-1) is used only once in the computation of (32), (27-2) is used $\log_2 N-1$ times, and Theorem 3 is used $N/4+N/8+\ldots+2+1=N/2-1$ times. Therefore, $N/2+\log_2 N-2$ multiplications with j are required. Note that Theorem 4 does not require multiplication with j. For a real signal, because $y_N(k,i)$ satisfies the conjugate symmetry property (15), there is no need to apply Theorem 3 in (32). Therefore, only $\log_2 N-1$ multiplications with j are required. The GCK method requires only two signal vectors to be stored in memory; therefore, it needs $4N$ words of memory for a complex signal and $2N$ words of memory for a real signal.

A. COMPARISON OF THE PROPOSED ALGORITHM WITH SOME EXISTING SLIDING ALGORITHMS IN TERMS OF COMPUTATIONAL AND MEMORY COMPLEXITIES

In [35]–[37], we provided a detailed comparison of the radix-2 and radix-4 sliding SCHTs and radix-4 sliding CS-SCHT with sliding FFT, sliding DFT, and sliding WHT in terms of computational complexity. Therefore, in this paper, we only compare with the sliding complex Hadamard transform algorithms. The comparison results are shown in Table 4. The proposed GCK algorithm reduces significantly the number of real additions compared to the block CS-SCHT algorithm [18], [19], but its memory requirement is twice as large. The proposed algorithm is of lower computational complexity than the radix-4 sliding

TABLE 3. Fast GCK algorithm for length-4 and length-8 CS-SCHTs. T2, T3, and T4 indicate the use of Theorem 2, Theorem 3 and Theorem 4, respectively.

Proposed algorithm (N=4)	Proposed algorithm (N=8)	
$y_4(0,i+1)=y_4(0,i)-(x_i-x_{i+4})$ $y_4(1,i+1)=-j[y_4(1,i)-(y_4(0,i)-y_4(0,i+1))]$ for k=1, using T2. $y_4(2,i+1)=-[y_4(2,i)-(y_4(0,i)-y_4(0,i+1))]$ for k=2, using T2. $y_4(3,i+1)=-[y_4(1,i+1)+j(y_4(1,i)-y_4(3,i))]$ for k=3, using T3.	$y_8(0,i+2)=y_8(0,i+1)-(x_{i+1}-x_{i+9})$	$y_8(4,i+2)=-[y_8(4,i+1)-(y_8(0,i+1)-y_8(0,i+2))]$ for k=4, using T2.
	$y_8(1,i+2)=-j[y_8(1,i)-(y_8(0,i)-y_8(0,i+2))]$ for k=1, using T2.	$y_8(5,i+2)=-[y_8(3,i+2)+j(y_8(5,i)-y_8(3,i))]$ for k=5, using T3.
	$y_8(2,i+2)=-j[y_8(2,i+1)-(y_8(0,i+1)-y_8(0,i+2))]$ for k=2, using T2.	$y_8(6,i+2)=-[y_8(2,i+2)+j(y_8(2,i+1)-$ $y_8(6,i+1))]$ for k=6, using T3.
	$y_8(3,i+2)=-[y_8(3,i+1)-(y_8(7,i+1)-y_8(7,i+2))]$ for k=3, using T4.	$y_8(7,i+2)=-[y_8(1,i+2)+j(y_8(1,i)-y_8(7,i))]$ for k=7, using T3.

TABLE 4. The computational and memory complexities of the proposed GCK algorithm, radix-4 sliding CS-SCHT algorithm [36], [37], block-based CS-SCHT algorithm [18], [19], sliding FFT algorithm [26], [27], and sliding DFT algorithm [39], [40]. ***Muls(j)*** denotes the number of multiplications by ***j***, ***Adds*** denotes the number of real additions, and ***Me*** denotes the size of the memory (in words). Superscripts CS and RCS denote CS-SCHT and Real CS-SCHT, respectively.

Algorithm	Input	*Muls*	*Muls(j)*	*Adds*	*Me*
Proposed GCK sliding CS-SCHT	Complex	0	$N/2+\log_2N-2$	$4N-2$	$4N$
	Real	0	$\log_2N-1$	$2N-1$	$2N$
Radix-4 sliding CS-SCHT (Scheme 1) [36], [37]	Complex	0	$2N/3+2\log_4N-8/3$, $N=2^p$, p=4,6,… $2N/3+2\log_4(N/2)-7/3$, $N=2^p$, p=5,7,…	$44N/9+(\log_4N-14/3)(\log_4N-1)-86/9$, $N=2^p$, p=4,6,… $44N/9+(\log_4(N/2)-10/3)(\log_4(N/2)-1)-118/9$, $N=2^p$, p=5,7,…	$N/2+\max\{7N/2$, $Me_{N/4}^{CS}+N(\log_2N-1)/2\}$
	Real	0	$N/3+2\log_4N-7/3$, $N=2^p$, p=4,6,… $N/3+2\log_4(N/2)-5/3$, $N=2^p$, p=5,7,…	$25N/9+(\log_4N-1)(\log_4N-50/3)/2-64/9$, $N=2^p$, p=4,6,… $25N/9+(\log_4(N/2)-1)(\log_4(N/2)-46/3)/2-110/9$, $N=2^p$, p=5,7,…	$N/4+\max\{3N/2$, $Me_{N/4}^{RCS}+(N/2)\log_2N-11N/8\}$
Block-based CS-SCHT [18], [19]	Complex	0	$N/2-1$	$2N\log_2N$	$2N$
	Real	0	$N/2-1$	$N\log_2N-N/2+1$	N
Sliding FFT [26], [27]	Complex	$4N-8\log_2 N$	$\log_2N-1$	$4N-4\log_2N-2$	$2N\log_2N-4$
	Real	$2N-4\log_2N$	$\log_2N-1$	$3N-4\log_2N$	$N\log_2N-N/4-2$
Sliding DFT [39], [40]	Complex	$4N-16$	2	$4N-6$	$4N-6$
	Real	$2N-8$	1	$3N/2-2$	$2N-3$

CS-SCHT [36], [37], sliding FFT [26], [27], and the sliding DFT [39], [40]. The proposed algorithm is of lower memory complexity than the radix-4 sliding CS-SCHT [36], [37] and the sliding FFT [26], [27]. For comparison purposes, Table 4 lists the numbers of multiplications with j and real additions.

B. COMPARISON OF THE PROPOSED ALGORITHM WITH SOME EXISTING SLIDING ALGORITHMS IN TERMS OF COMPUTER RUN TIME

In this section, we compare the computer run time of the proposed algorithm with those of other similar sliding complex transform algorithms, including the radix-4 sliding CS-SCHT algorithm [36], [37], the sliding FFT algorithm [26], [27] and the sliding DFT algorithm [39], [40]. These algorithms have been implemented in the C++ programming language and executed on a Thinkpad T440 machine with an Intel Core I5-6360U 2.00 GHz CPU and 8 GB RAM. The run time of these algorithms have been calculated using GCC complier version 4.2.1. The operating system is macOS 10.12.3.

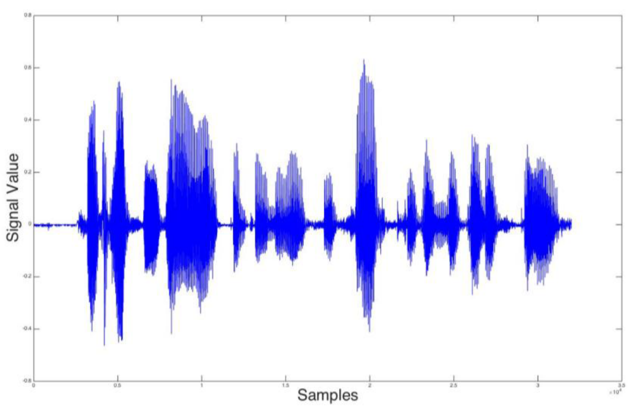

FIGURE 3. The audio data used in the experiment.

Figure 3 shows the experiment data, that is, a piece of audio signal consisting of $M = 32000$ samples. The audio signal is then divided into overlapping windows of size $N = 16$, and we obtain $M - N + 1$ length-N signals, which are then subject to various sliding complex transform algorithms. The time shown in the right-hand side of Fig. 4 represents

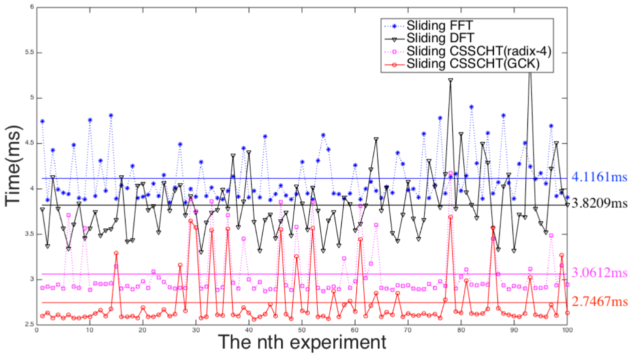

FIGURE 4. Comparison of the computer run time (ms) of the proposed GCK CS-SCHT algorithm with those of the radix-4 sliding CS-SCHT algorithm [36], [37], sliding FFT algorithm [26], [27], and sliding DFT algorithm [39], [40] on an Intel core I5 CPU using the GCC compiler.

the average over 100 repeated executions of the algorithm. According to this figure, the proposed GCK CS-SCHT algorithm requires 10.3% less time compared to the radix-4 sliding CS-SCHT algorithm [36], [37], 33.3% compared to the sliding FFT algorithm [26], [27], and 28.1% compared to the sliding DFT algorithm [39], [40].

Note that the results show that there are some fluctuations in the computation time. Since we use single-threaded compilation, therefore, we think the reason is that the computation time is sensitive to the CPU cache. The 100 experiments are executed automatically by batch shell script and the time of each experiment is less than 270 ms, therefore, the process scheduling and switching are frequent and CPU cache pages are frequently paged out.

VII. CONCLUSION

In this paper, we have presented a fast gray code kernel algorithm for computing the sliding CS-SCHT. The proposed algorithm computes the current projection value from the previously computed ones, where the peculiar computation order is obtained by using the α-related concept in the constructed CS-SCHT matrix tree. Then, the GCK sliding CS-SCHT algorithm is derived by using the properties of the elements of the CS-SCHT matrix. The arithmetic complexity order of the proposed algorithms is N, and improvement by a factor of $\log_2 N$ is achieved over the block-based algorithm for the length-N CS-SCHT. The proposed algorithm is also more efficient than that of other sliding complex transforms such as the radix-4 sliding CS-SCHT algorithm, the sliding FFT algorithm and the sliding DFT algorithm.

APPENDIX A
PROOF OF THEOREM 1

In this appendix, we provide the proof of Theorem 1. First, we present some basic concepts. Let k be an integer that is less than $N = 2^p$. Its binary representation is

$$k = \sum_{r=0}^{p-1} k_r 2^r. \tag{A1}$$

Let $\mathbf{G} = (g_{p-1}, g_{p-2}, \ldots, g_0)$ and $\mathbf{F} = (f_{p-1}, f_{p-2}, \ldots, f_0)$, where g_r is a binary gray code of the bit reversal of k_r and f_r is the rth binary bit of the highest power of 2 in $c(k)/2$, where $c(k)$ is the decimal number that is obtained through a bit-reversed conversion of the decimal k. Then, we have

$$f_s = \begin{cases} 0, & \text{if } s = p-1 \\ 1, & \text{if } s = p-2-t \\ 0, & \text{otherwise,} \end{cases} \tag{A2}$$

where t is the minimal value between 0 and $p-2$ such that $k_t = 1$.

$$g_{p-1} = k_0, \;\; g_{p-r} = (k_{r-1} + k_{r-2}) \mod 2, \text{ for } 2 \le r \le p. \tag{A3}$$

Proof of (17): The elements of $\mathbf{h}_N(k)$ are given by

$$h_N(k, l) = (-1)^{\mathbf{G \bullet L}} (-j)^{\mathbf{F \bullet L}} = (-1)^{\sum_{r=0}^{p-1} g_r l_r} (-j)^{\sum_{r=0}^{p-1} f_r l_r}. \tag{A4}$$

If k is even, using (A1), we have

$$k = \sum_{r=1}^{p-1} k_r 2^r. \tag{A5}$$

It can be deduced from (A2) and (A3) that

$$g_{p-1} = f_{p-1} = 0. \tag{A6}$$

For $0 \le l \le N/2 - 1$, we have

$$\begin{aligned} h_N(k, l+N/2) &= (-1)^{\sum_{r=0}^{p-1} g_r (l+N/2)_r} (-j)^{\sum_{r=0}^{p-1} f_r (l+N/2)_r} \\ &= (-1)^{g_{p-1}} (-j)^{f_{p-1}} (-1)^{\sum_{r=0}^{p-1} g_r l_r} (-j)^{\sum_{r=0}^{p-1} f_r l_r} \\ &= h_N(k, l). \end{aligned} \tag{A7}$$

Letting $k' = k/2$, the binary representation of k' is

$$k' = \sum_{r=0}^{p-2} k_{r+1} 2^r \quad \text{and } k'_{p-1} = 0. \tag{A8}$$

The elements of $\mathbf{h}_{N/2}(k')$ are given by

$$h_{N/2}(k', l) = (-1)^{\sum_{r=0}^{p-2} g'_r l_r} (-j)^{\sum_{r=0}^{p-2} f'_r l_r}, \tag{A9}$$

where

$$g'_{p-2} = k_1, \;\; g'_{p-r} = (k_{r-1} + k_{r-2}) \mod 2, \text{ for } 3 \le r \le p. \tag{A10}$$

Because

$$\begin{aligned} g_{p-1} &= 0; \\ g_{p-2} &= (k_1 + k_0) \mod 2 = k_1; \\ g_{p-r} &= (k_{r-1} + k_{r-2}) \mod 2, \text{ for } 3 \le r \le p. \end{aligned} \tag{A11}$$

Comparing (A10) with (A11), we obtain

$$g'_r = g_r, \quad \text{for } 0 \le r \le p-2. \tag{A12}$$

In addition, we can easily verify the following relationship:

$$f_r' = f_r, \quad \text{for } 0 \leq r \leq p-2. \tag{A13}$$

Therefore,

$$h_{N/2}(k', l) = h_N(k, l), \quad \text{for } 0 \leq l \leq N/2-1. \tag{A14}$$

Combining (A7) and (A14) yields the first equation in (17).

If k is odd (i.e., $k_0 = 1$), we have

$$g_{p-1} = 1 \quad \text{and } f_{p-1} = 0. \tag{A15}$$

It can be easily verified that

$$h_N(k, l + N/2) = -h_N(k, l). \tag{A16}$$

Moreover, using the definition of $\mathbf{H}_N^{1/2}$, we obtain

$$h_N^{1/2}(k, l) = h_N(k, l), \quad \text{for } 0 \leq l \leq N/2-1. \tag{A17}$$

Combining (A16) and (A17) yields the second equation in (17). □

Proof of (18): For $k = 4n+1$, $0 \leq n \leq N/4-1$, we have

$$k_0 = 1 \quad \text{and } k_1 = 0. \tag{A18}$$

Thus,

$$\begin{aligned} g_{p-2} &= 1 \\ f_{p-2} &= 1 \quad \text{and } f_r = 0 \text{ if } r \neq p-2. \end{aligned} \tag{A19}$$

Using (A17), for $0 \leq l \leq N/4-1$, we have

$$\begin{aligned} &h_N^{1/2}(k, l + N/4) \\ &\quad = h_N(k, l + N/4) \\ &\quad = (-1)^{g_{p-2}}(-j)^{f_{p-2}}(-1)^{\sum_{r=0}^{p-1} g_r l_r}(-j)^{\sum_{r=0}^{p-1} f_r l_r} \\ &\quad = j h_N(k, l) = j h_N^{1/2}(k, l). \end{aligned} \tag{A20}$$

Moreover,

$$h_N^{1/4}(k, l) = h_N^{1/2}(k, l) \quad \text{for } 0 \leq l \leq N/4-1. \tag{A21}$$

Combining (A20) and (A21) yields the first equation of (18).

For $k = 4n+3$, $0 \leq n \leq N/4-1$, we have

$$k_0 = k_1 = 1. \tag{A22}$$

Thus,

$$\begin{aligned} g_{p-2} &= 0 \\ f_{p-2} &= 1 \quad \text{and } f_r = 0 \text{ if } r \neq p-2. \end{aligned} \tag{A23}$$

We can deduce that

$$h_N^{1/2}(k, l + N/4) = -j h_N^{1/2}(k, l). \tag{A24}$$

Using (A21) and (A24), we obtain the second equation of (18). □

Proof of (19): The proof of (19) is similar to those of (17) and (18); the detailed derivation is omitted here.

APPENDIX B
PROOFS OF PROPERTIES 1-3

Proof of Property 1: (23-1) is easily to obtain from (10). The proofs of (23-2) and (23-3) are as follows.

Let $k = N/2^m = 2^{p-m}$ for $2 \leq m \leq p$. From (A2), we have

$$f_{m-2} = 1 \text{ and } f_r = 0 \text{ if } r \neq m-2, \tag{B1}$$

and

$$g_{m-1} = g_{m-2} = 1 \quad \text{and } g_r = 0, \text{ otherwise.} \tag{B2}$$

Using (B1) and (B2), we obtain

$$h_N(N/2^m, l) = (-1)^{l_{m-1}+l_{m-2}}(-j)^{l_{m-2}} = j^{\lfloor l/2^{m-2} \rfloor}, \tag{B3}$$

where $\lfloor x \rfloor$ denotes the lower integer part of x. The proof is complete. □

Proof of Property 2: Let $n = 2^s$ for $s = 0, 1, \ldots, p-2$, $m = 2^t - 1$ for $t = 1, 2, \ldots, s$, and $k' = k + (N - mN/(2n))$. We have

$$\begin{aligned} N - mN/(2n) &= 2^p - (2^t - 1)2^{p-s-1} \\ &= 2^{p-s-1} + \sum_{r=p+t-s-1}^{p-1} 2^r. \end{aligned} \tag{B4}$$

For $(m-1)\frac{N}{4n} + 1 \leq k \leq (m+1)\frac{N}{4n} - 1$, we obtain

$$(2^t - 2)2^{p-s-2} + 1 \leq k \leq 2^t 2^{p-s-2} - 1, \tag{B5}$$

or equivalently,

$$\sum_{r=p-s-1}^{p+t-s-3} 2^r + 1 \leq k \leq \sum_{r=0}^{p+t-s-3} 2^r = \sum_{r=0}^{p-s-2} 2^r + \sum_{r=p-s-1}^{p+t-s-3} 2^r. \tag{B6}$$

Letting $k = (k_{p-1}, k_{p-2}, \ldots, k_0)$, from (B8), we obtain

$$k_r = \begin{cases} k_r, & \text{if } 0 \leq r \leq p-s-2 \\ 1, & \text{if } p-s-1 \leq r \leq p+t-s-3 \\ 0, & \text{if } p+t-s-2 \leq r \leq p-1, \end{cases} \tag{B7}$$

that is,

$$k = \sum_{r=0}^{p-s-2} k_r 2^r + \sum_{r=p-s-1}^{p+t-s-3} 2^r, \tag{B8}$$

and there exists at least one r_0 between 0 and $p-s-2$ such that $k_{r_0} = 1$.

On the other hand,

$$\begin{aligned} k' &= \sum_{r=0}^{p-s-2} k_r 2^r + \sum_{r=p-s-1}^{p+t-s-3} 2^r + 2^{p-s-1} + \sum_{r=p+t-s-1}^{p-1} 2^r \\ &= \sum_{r=0}^{p-s-2} k_r 2^r + \sum_{r=p+t-s-2}^{p-1} 2^r = \sum_{r=0}^{p-1} k_r' 2^r, \end{aligned} \tag{B9}$$

where

$$k_r' = \begin{cases} k_r, & \text{if } 0 \leq r \leq p-s-2 \\ 0, & \text{if } p-s-1 \leq r \leq p+t-s-1 \\ 1, & \text{if } p+t-s-2 \leq r \leq p-1. \end{cases} \tag{B10}$$

By the definitions of g_r and f_r, we have

$$f_r = f_r' \quad \text{for } r = 0, 1, \ldots, p-1. \tag{B11}$$

We can deduce from (A3) that

$$g_{p-r} = g_{p-r}' \quad \text{for } 1 \le r \le p-s-1, \tag{B12}$$
$$g_{p-r} = g_{p-r}' \quad \text{for } p-s+1 \le r \le p-1. \tag{B13}$$

For $r = p - s$, we have

$$\begin{aligned} g_{p-(p-s)} &= g_s = (k_{p-s-1} + k_{p-s-2}) \mod 2 \\ &= (1 + k_{p-s-2}) \mod 2 \end{aligned}$$
$$g_{p-(p-s)}' = g_s' = (k_{p-s-1}' + k_{p-s-2}') \mod 2 \tag{B14}$$
$$= k_{p-s-2} \mod 2. \tag{B15}$$

It can be deduced from (B14) and (B15) that

$$g_{p-(p-s)} + g_{p-(p-s)}' = 1. \tag{B16}$$

Therefore,

$$\begin{aligned} &h_N\left(k + (N - mN/(2n)), l\right) \\ &= (-1)^{\sum_{r=0}^{p-1} g_r' l_r} (-j)^{\sum_{r=0}^{p-1} f_r' l_r} \\ &= (-1)^{\sum_{r=0}^{p-1} g_r l_r + l_{p-s} - 2g_{p-s} l_{p-s}} (-j)^{\sum_{r=0}^{p-1} f_r l_r} \\ &= (-1)^{l_{p-s}} h_N(k, l) = (-1)^{\lfloor l/2^s \rfloor} h_N(k, l) \\ &= (-1)^{\lfloor l/n \rfloor} h_N(k, l) \end{aligned} \tag{B17}$$

The proof of Property 2 has been completed. □

Proof of Property 3: We have

$$\begin{aligned} h_N(k, l+n) &= (-1)^{\sum_{r=0}^{p-1} g_r (l+n)_r} (-j)^{\sum_{r=0}^{p-1} f_r (l+n)_r} \\ &= (-1)^{\sum_{r=0}^{p-1} g_r n_r} (-j)^{\sum_{r=0}^{p-1} f_r n_r} h_N(k, l). \end{aligned} \tag{B18}$$

For $n = 2^s$, $0 \le s \le p-2$, the above equation becomes

$$h_N(k, l+n) = (-1)^{g_s} (-j)^{f_s} h_N(k, l). \tag{B19}$$

To obtain the relationship $h_N(k, l+n) = h_N(k, l)$, we must have

$$g_s = f_s = 0. \tag{B20}$$

According to (A2), there are two cases that lead to $f_s = 0$:

(1) $k_{p-s-2} = 0$;

(2) $k_{p-s-2} = 1$ and there exists at least one $s_0 > s$ such that $k_{p-s_0-2} = 1$.

Since

$$g_s = \left(k_{p-s-1} + k_{p-s-2}\right) \mod 2, \tag{B21}$$

we need one of the following two relationships to hold to obtain (B20):

(a) $k_{p-s-2} = k_{p-s-1} = 0$,

(b) $k_{p-s-2} = k_{p-s-1} = 1$ and there exists at least one $i \ge 3$ such that $k_{p-s-i} = 1$.

If $n = 1$ (i.e., $s = 0$), then all the values between 1 and $N/4 - 1$ satisfy (B20).

If $n = N/4$ (i.e., $s = p-2$), only $k_0 = k_1 = 0$ is possible. Then, we have $k = 4, 8, \ldots, N/2 - 4$.

If $n = 2^s$, for $1 \le s \le p-3$, the values of $k (1 < k < N/2)$ that satisfy (B21) can be expressed as

$$k = \sum_{r=0}^{p-s-3} k_r 2^r + k_{p-s-2} 2^{p-s-2} + k_{p-s-1} 2^{p-s-1} + \sum_{r=p-s}^{p-2} k_r 2^r. \tag{B22}$$

Corresponding to case (a), we have

$$k = \sum_{r=0}^{p-s-3} k_r 2^r + \sum_{r=p-s}^{p-2} k_r 2^r. \tag{B23}$$

The range of k in (B23) can be determined as follows: The first interval is

$$1 \le k \le 2^{p-s-2} - 1 \Leftrightarrow 1 \le k \le N/(4n) - 1.$$

Other intervals can be obtained by adding any integer r between 2^{p-s} and $2^{p-1} - 2^{p-s}$ to the first interval, that is,

$$r + 1 \le k \le N/(4n) - 1 + r, 2^{p-s} \le r \le 2^{p-1} - 2^{p-s}.$$

Thus, we can express (B23) as

$$\begin{aligned} k &= (m-1)N/(4n) + 1, \ldots, mN/(4n) - 1; \\ m &= n = 1 \text{ or } m = 1, 5, 9, \ldots, 2n-7, 2n-3; \\ &\qquad n = 2, 4, 8, \ldots, N/8. \end{aligned} \tag{B24}$$

Corresponding to case (b), we have

$$k = \sum_{r=0}^{p-s-3} k_r 2^r + 2^{p-s-2} + 2^{p-s-1} + \sum_{r=p-s}^{p-2} k_r 2^r. \tag{B25}$$

The first interval is

$$\begin{aligned} &2^{p-s-2} + 2^{p-s-1} + 1 \le k \\ &\quad \le 2^{p-s-2} - 1 + 2^{p-s-1} + 2^{p-s-2} \\ &\quad \Leftrightarrow 3N/(4n) + 1 \le k \le N/n - 1. \end{aligned}$$

Other intervals are given by

$$r + 3N/(4n) + 1 \le k \le N/n - 1 + r, \quad 2^{p-s} \le r \le 2^{p-1} - 2^{p-s}.$$

Thus, we can express (B25) as

$$\begin{aligned} k &= mN/(4n) + 1, \ldots, (m+1)N/(4n) - 1; \\ m &= 3, 7, 11, \ldots, 2n-5, 2n-1; \quad n = 2, 4, 8, \ldots, N/8. \end{aligned} \tag{B26}$$

The ranges of k for other cases can be determined in a similar manner; we give only a brief description below. To obtain the relationship $h_N(k, l+n) = -h_N(k, l)$, we must have

$$g_s = 1 \quad \text{and } f_s = 0. \tag{B27}$$

To obtain (B27), we need one of the following two relationships to hold:

(a) $k_{p-s-2} = 0$ and $k_{p-s-1} = 1$;

(b) $k_{p-s-2} = 1$ and $k_{p-s-1} = 0$, and there exists at least one $i \geq 3$ such that $k_{p-s-i} = 1$.

The ranges of k that correspond to (a) and (b) are respectively given by

$$\begin{aligned} k &= (m-1)N/(4n)+1, \ldots, mN/(4n)-1; \\ m &= 3, 7, 11, \ldots, 2n-5, 2n-1; \quad n = 2, 4, 8, \ldots, N/8; \end{aligned} \tag{B28}$$

$$\begin{aligned} k &= mN/(4n)+1, \ldots, (m+1)N/(4n)-1; \\ m &= n = 1 \text{ or } m = 1, 5, 9, \ldots, 2n-7, 2n-3; \\ & \qquad n = 2, 4, 8, \ldots, N/8. \end{aligned} \tag{B29}$$

By combining (B24) with (B28), we obtain (27-1), and by combining (B26) with (B29), we obtain (27-2).

To obtain the relationship $h_N(k, l+n) = jh_N(k, l)$, we must have

$$g_s = f_s = 1. \tag{B30}$$

To obtain (B30), we need

$$k_{p-s-2} = 1 \quad \text{and } k_{p-s-1} = 0,$$

where $p - s - 2$ is the minimal value between 0 and $p - 2$ such that $k_{p-s-2} = 1$.

To obtain the relationship $h_N(k, l+n) = -jh_N(k, l)$, we must have

$$g_s = 0 \quad \text{and } f_s = 1. \tag{B31}$$

To obtain (B31), we need

$$k_{p-s-2} = 1 \text{ and } k_{p-s-1} = 1,$$

where $p - s - 2$ is the minimal value between 0 and $p - 2$ such that $k_{p-s-2} = 1$. □

APPENDIX C
PROOFS OF THEOREMS 2-4

By the definition of $y_N(k, i)$, we have

$$\begin{aligned} y_N(k,i) &= \mathbf{h}_N^r(k)\mathbf{x}_N(i) = \sum_{l=0}^{N-1} h_N(k,l)x_{i+l} \\ &= \sum_{l=0}^{n-1} h_N(k,l)x_{i+l} + \sum_{l=0}^{N-n-1} h_N(k,l+n)x_{i+n+l}. \end{aligned} \tag{C1}$$

Similarly,

$$\begin{aligned} y_N(k,i+n) &= \mathbf{h}_N^r(k)\mathbf{x}_N(i+n) \\ &= \sum_{l=0}^{N-n-1} h_N(k,l)x_{i+n+l} \\ &\quad + \sum_{l=0}^{n-1} h_N(k,l+N-n)x_{i+N+l}. \end{aligned} \tag{C2}$$

From (C1) and (C2), we have

$$\begin{aligned} & y_N(k,i) - y_N(k,i+n) \\ &= \sum_{l=0}^{n-1} h_N(k,l)x_{i+l} - \sum_{l=0}^{n-1} h_N(k,l+N-n)x_{i+N+l} \\ &\quad - \sum_{l=0}^{N-n-1} \left(h_N(k,l) - h_N(k,l+n)\right) x_{i+n+l}, \end{aligned} \tag{C3}$$

$$\begin{aligned} & y_N(k,i) + y_N(k,i+n) \\ &= \sum_{l=0}^{n-1} h_N(k,l)x_{i+l} + \sum_{l=0}^{n-1} h_N(k,l+N-n)x_{i+N+l} \\ &\quad + \sum_{l=0}^{N-n-1} \left(h_N(k,l) + h_N(k,l+n)\right) x_{i+n+l}, \end{aligned} \tag{C4}$$

$$\begin{aligned} & y_N(k,i) - jy_N(k,i+n) \\ &= \sum_{l=0}^{n-1} h_N(k,l)x_{i+l} - j\sum_{l=0}^{n-1} h_N(k,l+N-n)x_{i+N+l} \\ &\quad - \sum_{l=0}^{N-n-1} \left(jh_N(k,l) - h_N(k,l+n)\right) x_{i+n+l}, \end{aligned} \tag{C5}$$

$$\begin{aligned} & y_N(k,i) + jy_N(k,i+n) \\ &= \sum_{l=0}^{n-1} h_N(k,l)x_{i+l} + j\sum_{l=0}^{n-1} h_N(k,l+N-n)x_{i+N+l} \\ &\quad + \sum_{l=0}^{N-n-1} \left(jh_N(k,l) + h_N(k,l+n)\right) x_{i+n+l}. \end{aligned} \tag{C6}$$

Proof of Theorem 2:

(a) For $k = 0$, we deduce from (C3) that

$$\begin{aligned} & y_N(0,i) - y_N(0,i+1) \\ &\quad = h_N(0,0)x_i - h_N(0,N-1)x_{i+N} \\ &\quad\quad - \sum_{l=0}^{N-2} \left(h_N(0,l) - h_N(0,l+1)\right) x_{i+1+l} \end{aligned} \tag{C7}$$

By applying (23-1) to (C7), we obtain

$$y_N(0,i) - y_N(0,i+1) = x_i - x_{i+N}. \tag{C8}$$

Setting $k = N/2$ in (C4) and applying (23-2) yields

$$\begin{aligned} & y_N(N/2,i) + y_N(N/2,i+1) \\ &= h_N(N/2,0)x_i + h_N(N/2,N-1)x_{i+N} \\ &\quad + \sum_{l=0}^{N-2} \left(h_N(N/2,l) + h_N(N/2,l+1)\right) x_{i+1+l} \\ &= x_i + (-1)^{N-1}x_{i+N} + \sum_{l=0}^{N-2} \left((-1)^l + (-1)^{l+1}\right) x_{i+1+l} \\ &= x_i - x_{i+N} \end{aligned} \tag{C9}$$

By comparing (C8) with (C9), we obtain the first formula of Theorem 2.

(b) Letting $n = N/(4k), k = 1, 2, 4, \ldots, N/8, N/4$, and using (23-1) yields

$$\begin{aligned} y_N(0, i) - y_N(0, i+n) &= \sum_{l=0}^{n-1} h_N(0, l)x_{i+l} - \sum_{l=0}^{n-1} h_N(0, l+N-n)x_{i+N+l} \\ &\quad - \sum_{l=0}^{N-n-1} \left(h_N(0, l) - h_N(0, l+n)\right) x_{i+n+l} \\ &= \sum_{l=0}^{n-1} (x_{i+l} - x_{i+N+l}) \end{aligned} \tag{C10}$$

Using (23-3), we have

$$h_N(k, l+n) = j^{\lfloor (l+n)/n \rfloor} = j \times j^{\lfloor l/n \rfloor} = jh_N(k, l). \tag{C11}$$

By substituting (C11) into (C5) and using (23-3), we obtain

$$\begin{aligned} y_N(k, i) - jy_N(k, i+n) &= \sum_{l=0}^{n-1} h_N(k, l)x_{i+l} + j^2 \sum_{l=0}^{n-1} h_N(k, l)x_{i+N+l} \\ &= \sum_{l=0}^{n-1} j^{\lfloor l/n \rfloor}(x_{i+l} - x_{i+N+l}) = \sum_{l=0}^{n-1} (x_{i+l} - x_{i+N+l}). \end{aligned} \tag{C12}$$

Similarly,

$$\begin{aligned} &y_N(N-k, i) + jy_N(N-k, i+n) \\ &= \sum_{l=0}^{n-1} h_N(N-k, l)x_{i+l} + j \sum_{l=0}^{n-1} h_N(N-k, l+N-n)x_{i+N+l} \\ &\quad + \sum_{l=0}^{N-n-1} \left(jh_N(N-k, l) + h_N(N-k, l+n)\right) x_{i+n+l} \\ &= \sum_{l=0}^{n-1} (-j)^{\lfloor l/n \rfloor} (x_{i+l} - x_{i+N+l}) = \sum_{l=0}^{n-1} (x_{i+l} - x_{i+N+l}) \end{aligned} \tag{C13}$$

From (C10), (C12) and (C13), we can obtain (27-2). □

Proof of Theorem 3:

(a) From (C5), we have

$$\begin{aligned} &y_N(k, i) + (-j)^m y_N(k, i+n) \\ &= \sum_{l=0}^{n-1} h_N(k, l)x_{i+l} + (-j)^m \sum_{l=0}^{n-1} h_N(k, l+N-n)x_{i+N+l} \\ &\quad + \sum_{r=0}^{N/n-2} \sum_{l=rn}^{(r+1)n-1} \left((-j)^m h_N(k, l) - h_N(k, l+n)\right) x_{i+n+l} \\ &= \sum_{l=0}^{n-1} h_N(k, l)x_{i+l} + (-j)^m \sum_{l=0}^{n-1} h_N(k, l+N-n)x_{i+N+l} \\ &\quad + \sum_{r=0}^{N/n-2} \sum_{l=0}^{n-1} ((-j)^m h_N(k, l+rn) \\ &\qquad + h_N(k, l+rn+n))x_{i+n+l+rn} \end{aligned} \tag{C14}$$

Applying (26-3), (C14) becomes

$$\begin{aligned} &y_N(k, i) + (-j)^m y_N(k, i+n) \\ &= \sum_{l=0}^{n-1} h_N(k, l)x_{i+l} + (-j)^m \sum_{l=0}^{n-1} h_N(k, l+N-n)x_{i+N+l}. \end{aligned} \tag{C15}$$

Using (23-3), it can be deduced from (C6) that

$$\begin{aligned} &y_N(N-k, i) + j^m y_N(N-k, i+n) \\ &= \sum_{l=0}^{n-1} (-1)^{\lfloor l/n \rfloor} h_N\,(k, l)\, x_{i+l} \\ &\quad + j^m \sum_{l=0}^{n-1} (-1)^{\lfloor (l+N-n)/n \rfloor} h_N\,(k, l+N-n)\, x_{i+N+l} \\ &\quad + \sum_{l=0}^{N-n-1} (j^m (-1)^{\lfloor l/n \rfloor} h_N(k, l) \\ &\quad + (-1)^{\lfloor (l+n)/n \rfloor} h_N(k, l+n))x_{i+n+l} \\ &= \sum_{l=0}^{n-1} h_N\,(k, l)\, x_{i+l} - j^m \sum_{l=0}^{n-1} h_N\,(k, l+N-n)\, x_{i+N+l} \\ &\quad + \sum_{r=0}^{N/n-2} \sum_{l=0}^{n-1} (-1)^{\lfloor r+l/n \rfloor} \\ &\quad \times \left(j^m h_N\,(k, l+rn) - h_N\,(k, l+rn+n)\right) x_{i+n+l+rn} \end{aligned} \tag{C16}$$

Using (26-3), (C16) becomes

$$\begin{aligned} &y_N(N-k, i) + j^m y_N(N-k, i+n) \\ &= \sum_{l=0}^{n-1} h_N\,(k, l)\, x_{i+l} - j^m \sum_{l=0}^{n-1} h_N\,(k, l+N-n)\, x_{i+N+l}. \end{aligned} \tag{C17}$$

Comparing (C15) with (C17), we obtain (28). □

Proof of Theorem 4:

(a) From (C3), we have

$$\begin{aligned} &y_N(k, i) - y_N(k, i+n) \\ &= \sum_{l=0}^{n-1} h_N(k, l)x_{i+l} - \sum_{l=0}^{n-1} h_N(k, l+N-n)x_{i+N+l} \\ &\quad - \sum_{r=0}^{N/n-2} \sum_{l=0}^{n-1} \left(h_N(k, l+rn) - h_N(k, l+rn+n)\right) x_{i+n+l+rn} \\ &= \sum_{l=0}^{n-1} h_N(k, l)x_{i+l} - \sum_{l=0}^{n-1} h_N(k, l+N-n)x_{i+N+l}, \end{aligned} \tag{C18}$$

where (26-1) has been utilized in the last step of the above equation.

From (C4), we have

$$\begin{aligned}
&y_N(k+(N-mN/(2n)),i) \\
&\quad +(-1)^{(m-1)/2}y_N(k+(N-mN/(2n)),i+n) \\
&=\sum_{l=0}^{n-1}h_N(k+(N-mN/(2n))),l)x_{i+l}+(-1)^{(m-1)/2} \\
&\quad\times\sum_{l=0}^{n-1}h_N(k+(N-mN/(2n)),l+N-n)x_{i+N+l} \\
&\quad+\sum_{r=0}^{N/n-2}\sum_{l=0}^{n-1} \\
&\quad\times\begin{pmatrix}(-1)^{(m-1)/2}h_N(k+(N-mN/(2n)),\\ l+rn)+h_N(k+(N-mN/(2n)),\\ l+rn+n)\end{pmatrix}x_{i+n+l+rn}
\end{aligned} \tag{C19}$$

Using Property 2 for

$$\begin{aligned}
&k=(m-1)N/(4n)+1,(m-1)N/(4n) \\
&\quad +2,\ldots,(m+1)N/(4n)-1; \\
&m=1,3,5,7,\ldots,2n-1;\ n=1,2,4,\ldots,N/8,N/4,
\end{aligned}$$

(C19) becomes

$$\begin{aligned}
&y_N(k+(N-mN/(2n)),i) \\
&\quad +(-1)^{(m-1)/2}y_N(k+(N-mN/(2n)),i+n) \\
&=\sum_{l=0}^{n-1}h_N(k,l)x_{i+l}-(-1)^{(m-1)/2} \\
&\quad\times\sum_{l=0}^{n-1}h_N(k,l+N-n)x_{i+N+l} \\
&\quad+\sum_{r=0}^{N/n-2}(-1)^r\sum_{l=0}^{n-1}((-1)^{(m-1)/2}h_N(k,l+rn) \\
&\quad-h_N(k,l+rn+n))x_{i+n+l+rn} \\
&=\sum_{l=0}^{n-1}h_N(k,l)x_{i+l}-(-1)^{(m-1)/2} \\
&\quad\times\sum_{l=0}^{n-1}h_N(k,l+N-n)x_{i+N+l},
\end{aligned} \tag{C20}$$

where (26-1) has been utilized in the last step of the above equation.

Combining (C18) and (C20) leads to (29-1).

(b) From (C4), we have

$$\begin{aligned}
&y_N(k,i)+(-1)^{(m-1)/2}y_N(k,i+n) \\
&=\sum_{l=0}^{n-1}h_N(k,l)x_{i+l}+(-1)^{(m-1)/2} \\
&\quad\times\sum_{l=0}^{n-1}h_N(k,l+N-n)x_{i+N+l}+\sum_{r=0}^{N/n-2}\sum_{l=0}^{n-1} \\
&\quad\times((-1)^{(m-1)/2}h_N(k,l+rn)+h_N(k,l+rn+n))x_{i+n+l+rn}
\end{aligned} \tag{C21}$$

Using (26-2), (C21) becomes

$$\begin{aligned}
y_N(k,i)+y_N(k,i+n)&=\sum_{l=0}^{n-1}h_N(k,l)x_{i+l}+(-1)^{(m-1)/2} \\
&\quad\times\sum_{l=0}^{n-1}h_N(k,l+N-n)x_{i+N+l}
\end{aligned} \tag{C22}$$

From (C3), we have

$$\begin{aligned}
&y_N(k+(N-mN/(2n)),i) \\
&\quad -(-1)^{(m-1)/2}y_N(k+(N-mN/(2n)),i+n) \\
&=\sum_{l=0}^{n-1}h_N(k+(N-mN/(2n)),l)x_{i+l}-(-1)^{(m-1)/2} \\
&\quad\times\sum_{l=0}^{n-1}h_N(k+(N-mN/(2n)),l+N-n)x_{i+N+l} \\
&\quad+\sum_{r=0}^{N/n-2}\sum_{l=0}^{n-1}\begin{pmatrix}(-1)^{(m-1)/2}h_N(k+(N-mN\\ /(2n)),l+rn)-h_N(k+(N\\ -mN/(2n)),l+rn+n)\end{pmatrix}x_{i+n+l+rn}
\end{aligned} \tag{C23}$$

Using Property 2, we have

$$\begin{aligned}
&y_N(k+(N-mN/(2n)),i) \\
&\quad -(-1)^{(m-1)/2}y_N(k+(N-mN/(2n)),i+n) \\
&=\sum_{l=0}^{n-1}h_N(k,l)x_{i+l}+(-1)^{(m-1)/2} \\
&\quad\times\sum_{l=0}^{n-1}h_N(k,l+N-n)x_{i+N+l} \\
&\quad+\sum_{r=0}^{N/n-2}(-1)^r\sum_{l=0}^{n-1}((-1)^{(m-1)/2}h_N(k,l+rn) \\
&\quad+h_N(k,l+rn+n))x_{i+n+l+rn} \\
&=\sum_{l=0}^{n-1}h_N(k,l)x_{i+l} \\
&\quad+(-1)^{(m-1)/2}\sum_{l=0}^{n-1}h_N(k,l+N-n)x_{i+N+l},
\end{aligned} \tag{C24}$$

where (26-2) has been utilized in the last step of the above equation.

Combining (C22) and (C24) leads to (29-2). □

APPENDIX D
PROOF OF (30)

We replace n by $N/(4n)$ in (28), which yields

$$\begin{aligned}
&y_N(mn,i)-j^m y_N(mn,i+N/(4n)) \\
&\quad =y_N(N-mn,i)+j^m y_N(N-mn,i+N/(4n)), \\
&m=1,3,5,\ldots,N/(2n)-1;\ n=1,2,4,\ldots,N/8,N/4
\end{aligned} \tag{D1}$$

Set $k = mn$, we have

$$\begin{aligned} &y_N(N-k,i)+j^m y_N(N-k,i+N/(4n)) \\ &\quad = y_N(k,i)-j^m y_N(k,i+N/(4n)), \\ &\qquad \text{for } k=1,2,3,4,\ldots,N/2-1; \\ &\qquad n=1,2,4,8,\ldots,N/8,N/4; \\ &\qquad m=k/n, \quad \text{and } m \text{ is a positive odd integer.} \end{aligned} \tag{D2}$$

Eq. (D2) is equal to

$$\begin{aligned} &y_N(N-k,i)+j^{k/n} y_N(N-k,i+N/(4n)) \\ &\quad = y_N(k,i)-j^{k/n} y_N(k,i+N/(4n)), \\ &\qquad \text{for } k=n,3n,5n,\ldots,N/2-n; \\ &n=1,2,4,8,\ldots,N/8,N/4; \end{aligned} \tag{D3}$$

Set $n = k_0$ in (D3), we obtain (30). □

APPENDIX E
PROOF OF (31)

We replace n by $N/(8n)$ and k by $k-(N-4mn)$ in (29-2), and then choose part of the k (from $N-2mn+1, \ldots, N-2mn+n$), which yields

$$\begin{aligned} &y_N(k-(N-4mn),i) \\ &\quad + (-1)^{(m-1)/2} y_N(k-(N-4mn),i+N/(8n)) \\ &= y_N(k,i)-(-1)^{(m-1)/2} y_N(k,i+N/(8n)), \\ &\quad \text{for } k=N-2mn+1,\ldots,N-2mn+n; \\ &\quad m=1,3,5,\ldots,N/(4n)-1; \quad n=1,2,4,\ldots,N/8. \end{aligned} \tag{E1}$$

(E1) is equal to

$$\begin{aligned} &y_N(2mn+k_0,i)+(-1)^{(m-1)/2} y_N(2mn+k_0,i+N/(8n)) \\ &\quad = y_N(N-2mn+k_0,i) \\ &\qquad -(-1)^{(m-1)/2} y_N(N-2mn+k_0,i+N/(8n)), \\ &\qquad \text{for } k=N-2mn+k_0; \quad k_0=1,2,4,8,\ldots,N/8; \\ &m=1,3,5,\ldots,N/(4n)-1; \quad n=1,2,4,\ldots,N/8. \end{aligned} \tag{E2}$$

Replacing k by N-k in (E2), we obtain (31). □

ACKNOWLEDGMENT

The authors are thankful to Dr. Aye Aung for providing his MATLAB code for constructing the CS-SCHT matrix, Dr. Yair Moshe for providing his C code for GCK WHT algorithm, Dr. Wanli Ouyang for many helpful discussions. The authors are also grateful to the anonymous reviewers for their constructive comments and suggestions to greatly improve the quality of this work and the clarity of the presentation.

REFERENCES

[1] N. Ahmed and K. R. Rao, *Orthogonal Transforms for Digital Signal Processing*. New York, NY, USA: Springer, 1975.

[2] A. D. Poularikas, *Transforms and Applications Handbook*, 3rd ed. Boca Raton, FL, USA: CRC Press, 2009.

[3] W. H. Zheng, K. L. Li, and K. Q. Li, "Scaled radix-2/8 algorithm for efficient computation of length-$N=2^m$ DFTs," *IEEE Trans. Signal Process.*, vol. 62, no. 10, pp. 2492–2503, May 2014.

[4] K. L. Li, W. H. Zheng, and K. Q. Li, "A fast algorithm with less operations for length-N=q×2^m DFTs," *IEEE Trans. Signal Process.*, vol. 63, no. 3, pp. 673–683, Feb. 2015.

[5] W. Zheng, K. Li, K. Li, and H. C. So, "A modified multiple alignment fast Fourier transform with higher efficiency," *IEEE/ACM Trans. Comput. Biol. Bioinf.*, vol. 14, no. 3, pp. 634–645, May/Jun. 2017.

[6] Y. A. Geadah and M. J. Corinthios, "Natural, dyadic, and sequency order algorithms and processors for the Walsh-Hadamard transform," *IEEE Trans. Comput.*, vol. C-26, no. 5, pp. 435–442, May 1977.

[7] S. Boussakta and A. G. J. Holt, "Fast algorithm for calculation of both Walsh-Hadamard and Fourier transforms (FWFTs)," *Electron. Lett.*, vol. 25, no. 20, pp. 1352–1354, Sep. 1989.

[8] D. Sundararajan and M. O. Ahmad, "Fast computation of the discrete Walsh and Hadamard transforms," *IEEE Trans. Image Process.*, vol. 7, no. 6, pp. 898–904, Jun. 1998.

[9] S. Rahardja and B. J. Falkowski, "Family of unified complex Hadamard transforms," *IEEE Trans. Circuits Syst. II, Analog Digit. Signal Process.*, vol. 46, no. 8, pp. 1094–1100, Aug. 1999.

[10] S. Rahardja and B. J. Falkowski, "Complex composite spectra of unified complex Hadamard transform for logic functions," *IEEE Trans. Circuits Syst. II, Analog Digit. Signal Process.*, vol. 47, no. 11, pp. 1291–1297, Nov. 2000.

[11] S. Xie and S. Rahardja, "Performance evaluation for quaternary DS-SSMA communications with complex signature sequences over Rayleigh-fading channels," *IEEE Trans. Wireless Commun.*, vol. 4, no. 1, pp. 266–277, Jan. 2005.

[12] A. Aung, B. P. Ng, and S. Rahardja, "Sequency-ordered complex Hadamard transform: Properties, computational complexity and applications," *IEEE Trans. Signal Process.*, vol. 56, no. 8, pp. 3562–3571, Aug. 2008.

[13] A. Aung, B. P. Ng, and S. Rahardja, "A robust watermarking scheme using sequency-ordered complex Hadamard transform," *J. Signal Process. Syst.*, vol. 64, no. 3, pp. 319–333, 2011.

[14] A. Aung, B. P. Ng, and S. Rahardja, "Performance of SCHT sequences in asynchronous CDMA system," *IEEE Commun. Lett.*, vol. 11, no. 8, pp. 641–643, Aug. 2007.

[15] B. Wang, J. Wu, H. Shu, and L. Luo, "Shape description using sequency-ordered complex Hadamard transform," *Opt. Commu.*, vol. 284, no. 12, pp. 2726–2729, Jun. 2011.

[16] G. Bi, A. Aung, and B. P. Ng, "Pipelined hardware structure for sequency-ordered complex Hadamard transform," *IEEE Signal Process. Lett.*, vol. 15, pp. 401–404, 2008.

[17] A. Aung and B. P. Ng, "Natural-ordered complex Hadamard transform," *Signal Process.*, vol. 90, no. 3, pp. 874–879, Mar. 2010.

[18] A. Aung, B. P. Ng, and S. Rahardja, "Conjugate symmetric sequency-ordered complex Hadamard transform," *IEEE Trans. Signal Process.*, vol. 57, no. 7, pp. 2582–2593, Jul. 2009.

[19] S. Bouguezel, M. O. Ahmad, and M. N. S. Swamy, "An efficient algorithm for the conjugate symmetric sequency-ordered complex Hadamard transform," in *Proc. IEEE ISCAS*, May 2011, pp. 1516–1519.

[20] S. Kyochi and Y. Tanaka, "General factorization of conjugate-symmetric Hadamard transforms," *IEEE Trans. Signal Process.*, vol. 62, no. 13, pp. 3379–3392, Jul. 2014.

[21] S.-C. Pei, C.-C. Wen, and J.-J. Ding, "Conjugate symmetric discrete orthogonal transform," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 61, no. 4, pp. 284–288, Apr. 2014.

[22] D. Jabeen, G. Monir, and F. Azim, "Sequency domain signal processing using complex Hadamard transform," *Circuits Syst. Signal Process.*, vol. 35, no. 5, pp. 1783–1793, 2016.

[23] D. Jabeen and G. Monir, "Two-dimensional spatiochromatic signal processing using concept of phasors in sequency domain," *Electron. Lett.*, vol. 52, no. 11, pp. 968–970, May 2016.

[24] G. H. Hostetter, "Recursive discrete Fourier transformation," *IEEE Trans. Acoust., Speech, Signal Process.*, vol. ASSP-28, no. 2, pp. 184–190, Apr. 1980.

[25] J.-C. Liu and T.-P. Lin, "Running DHT and real-time DHT analyser," *Electron. Lett.*, vol. 24, no. 12, pp. 762–763, 1988.

[26] B. Farhang-Boroujeny and Y. C. Lim, "A comment on the computational complexity of sliding FFT," *IEEE Trans. Circuits Syst. II, Analog Digit. Signal Process.*, vol. 39, no. 12, pp. 875–876, Dec. 1992.

[27] B. Farhang-Boroujeny and S. Gazor, "Generalized sliding FFT and its application to implementation of block LMS adaptive filters," *IEEE Trans. Signal Process.*, vol. 42, no. 3, pp. 532–538, Mar. 1994.
[28] B. Farhang-Boroujeny, "Order of N complexity transform domain adaptive filters," *IEEE Trans. Circuits Syst. II, Analog Digit. Signal Process.*, vol. 42, no. 7, pp. 478–480, Jul. 1995.
[29] C.-S. Park, "Guaranteed-stable sliding DFT algorithm with minimal computational requirements," *IEEE Trans. Signal Process.*, vol. 65, no. 20, pp. 5281–5288, Oct. 2017.
[30] K. Y. Byun, C. S. Park, J. Y. Sun, and S. J. Ko, "Vector radix 2 × 2 sliding fast Fourier transform," *Math. Problems Eng.*, vol. 2016, Dec. 2016, Art. no. 2416286.
[31] L. F. Richardson and W. F. Eddy. (Jul. 2017). "The 2D tree sliding window discrete Fourier transform." [Online]. Available: https://arxiv.org/abs/1707.08213
[32] Y. Hel-Or and H. Hel-Or, "Real-time pattern matching using projection kernels," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 27, no. 9, pp. 1430–1445, Sep. 2005.
[33] W. Ouyang and W.-K. Cham, "Fast algorithm for Walsh Hadamard transform on sliding windows," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 32, no. 1, pp. 165–171, Jan. 2010.
[34] C.-S. Park, "Recursive algorithm for sliding Walsh Hadamard transform," *IEEE Trans. Signal Process.*, vol. 62, no. 11, pp. 2827–2836, Jun. 2014.
[35] J. Wu, H. Shu, L. Wang, and L. Senhadji, "Fast algorithms for the computation of sliding sequency-ordered complex Hadamard transform," *IEEE Trans. Signal Process.*, vol. 58, no. 11, pp. 5901–5909, Nov. 2010.
[36] J. S. Wu, L. Wang, L. Senhadji, and H. Z. Shu, "Sliding conjugate symmetric sequency-ordered complex Hadamard transform: Fast algorithm and applications," in *Proc. EUSIPCO*, Aalborg, Denmark, Aug. 2010, pp. 1742–1746.
[37] J. Wu, L. Wang, G. Yang, L. Senhadji, L. Luo, and H. Shu, "Sliding conjugate symmetric sequency-ordered complex Hadamard transform: Fast algorithm and applications," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 59, no. 6, pp. 1321–1334, Jun. 2012.
[38] J. A. R. Macias and A. G. Exposito, "Efficient moving-window DFT algorithms," *IEEE Trans. Circuits Syst. II, Analog Digit. Signal Process.*, vol. 45, no. 2, pp. 256–260, Feb. 1998.
[39] E. Jacobsen and R. Lyons, "The sliding DFT," *IEEE Signal Process. Mag.*, vol. 20, no. 2, pp. 74–80, Mar. 2003.
[40] E. Jacobsen and R. Lyons, "An update to the sliding DFT," *IEEE Signal Process. Mag.*, vol. 21, no. 1, pp. 110–111, Jan. 2004.
[41] K. Duda, "Accurate, guaranteed stable, sliding discrete Fourier transform [DSP tips & tricks]," *IEEE Signal Process. Mag.*, vol. 27, no. 6, pp. 124–127, Nov. 2010.
[42] C. S. Park and S. J. Ko, "The hopping discrete Fourier transform [sp tips&tricks]," *IEEE Signal Process. Mag.*, vol. 31, no. 2, pp. 135–139, Mar. 2014.
[43] C.-S. Park, "Fast, accurate, and guaranteed stable sliding discrete Fourier transform [sp tips&tricks]," *IEEE Signal Process. Mag.*, vol. 32, no. 4, pp. 145–156, Jul. 2015.
[44] Q. Wang, X. Yan, and K. Qin, "High-precision, permanently stable, modulated hopping discrete Fourier transform," *IEEE Signal Process. Lett.*, vol. 22, no. 6, pp. 748–751, Jun. 2015.
[45] D. A. Gudovskiy and L. Chu, "An accurate and stable sliding DFT computed by a modified CIC filter [tips & tricks]," *IEEE Signal Process. Mag.*, vol. 34, no. 1, pp. 89–93, Jan. 2017.
[46] W.-H. Juang, S.-C. Lai, K.-H. Chen, W.-K. Tsai, and C.-H. Luo, "Low-complexity hopping DFT design based on a compact recursive structure," *Electron. Lett.*, vol. 53, no. 1, pp. 25–27, Jan. 2017.
[47] W.-H. Juang, S.-C. Lai, C.-H. Luo, and S.-Y. Lee, "VLSI architecture for novel hopping discrete Fourier transform computation," *IEEE Access*, vol. 6, pp. 30491–30500, 2018.
[48] C. S. Park, "2D discrete Fourier transform on sliding windows," *IEEE Trans. Image Process.*, vol. 24, no. 3, pp. 901–907, Mar. 2015.
[49] Z. Dong, J. Wu, and J. Y. Gui, "A novel sliding window algorithm for 2D discrete Fourier transform," *Proc. SPIE*, vol. 9814, p. 981406, Dec. 2015.
[50] P. Yip and K. Rao, "On the shift property of DCT's and DST's," *IEEE Trans. Acoust., Speech, Signal Process.*, vol. ASSP-35, no. 3, pp. 404–406, Mar. 1987.
[51] N. R. Murthy and M. N. S. Swamy, "On the computation of running discrete cosine and sine transform," *IEEE Trans. Signal Process.*, vol. 40, no. 6, pp. 1430–1437, Jun. 1992.
[52] J. A. R. Macias and A. G. Exposito, "Recursive formulation of short-time discrete trigonometric transforms," *IEEE Trans. Circuits Syst. II, Analog Digit. Signal Process.*, vol. 45, no. 4, pp. 525–527, Apr. 1998.
[53] J. Xi and J. F. Chicharo, "Computing running DCTs and DSTs based on their second-order shift properties," *IEEE Trans. Circuits Syst. I, Fundam. Theory Appl.*, vol. 47, no. 5, pp. 779–783, May 2000.
[54] V. Kober, "Fast algorithms for the computation of sliding discrete sinusoidal transforms," *IEEE Trans. Signal Process.*, vol. 52, no. 6, pp. 1704–1710, Jun. 2004.
[55] C.-S. Park, "Two-dimensional discrete cosine transform on sliding windows," *Digital Signal Process.*, vol. 58, pp. 20–25, Nov. 2016.
[56] J. Xi and J. F. Chicharo, "Computing running discrete Hartley transform and running discrete W transforms based on the adaptive LMS algorithm," *IEEE Trans. Circuits Syst. II, Analog Digit. Signal Process.*, vol. 44, no. 3, pp. 257–260, Mar. 1997.
[57] V. Kober, "Fast algorithms for the computation of sliding discrete Hartley transforms," *IEEE Trans. Signal Process.*, vol. 55, no. 6, pp. 2937–2944, Jun. 2007.
[58] J. S. Bhat and C. Vijaya, "Sliding discrete fractional transforms," *Signal Process.*, vol. 88, no. 2, pp. 247–254, Feb. 2008.
[59] R. Tao, Y.-L. Li, and Y. Wang, "Short-time fractional Fourier transform and its applications," *IEEE Trans. Signal Process.*, vol. 58, no. 5, pp. 2568–2580, May 2010.
[60] B. Mozafari and M. H. Savoji, "An efficient recursive algorithm and an explicit formula for calculating update vectors of running Walsh-Hadamard transform," in *Proc. IEEE ISSPA*, Feb. 2007, pp. 1–4.
[61] G. Ben-Artz, H. Hel-Or, and Y. Hel-Or, "The gray-code filter kernels," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 29, no. 3, pp. 382–393, Mar. 2007.
[62] Y. Moshe and H. Hel-Or, "Video block motion estimation based on gray-code kernels," *IEEE Trans. Image Process.*, vol. 18, no. 10, pp. 2243–2254, Oct. 2009.
[63] J. M. Cioffi, "Limited-precision effects in adaptive filtering," *IEEE Trans. Circuits Syst.*, vol. CAS-34, no. 7, pp. 821–833, Jul. 1987.

JIASONG WU (M'09) received the B.S. degree in biomedical engineering from the University of South China, Hengyang, China, in 2005, and the joint Ph.D. degree with the Laboratory of Image Science and Technology (LIST), Southeast University, Nanjing, China, and Laboratoire Traitement du signal et de l'Image, University of Rennes 1, Rennes, France, in 2012. He is currently with LIST as a Lecturer. His research interest mainly includes deep learning, fast algorithms of digital signal processing and its applications. He received the Eiffel Doctorate Scholarship of Excellence (2009) from the French Ministry of Foreign Affairs and also the Chinese Government Award for Outstanding Self-Financed Student Abroad (2010) from the China Scholarship Council.

FUZHI WU received the B.E. degree from Anhui Normal University in 2017. He is currently pursuing the M.S. degree with the School of Computer Science and Engineering, Southeast University. His research interests include signal and image processing.

ZHIFANG DONG received the B.S., M.S., and Ph.D. degrees in biological science from Southeast University, Nanjing, China, in 1996, 2001, and 2011, respectively. She is currently an Associate Professor with the School of Electronic Science and Engineering, Southeast University. Her research interests include signal and image processing.

KAIWEN SONG received the B.S. degree from the Honor School, Nanjing Normal University, in 2014, and the M.S. degree from the School of Electronic Science and Engineering, Southeast University. His research interests include sliding window transforms.

YOUYONG KONG received the B.S. and M.S. degrees in computer science and engineering from Southeast University, Nanjing, China, in 2008 and 2011, respectively, and the Ph.D. degree in imaging and diagnostic radiology from the Chinese University of Hong Kong, Hong Kong, in 2014. He is currently an Assistant Professor with the College of Computer Science and Engineering, Southeast University. His current research interests include machine learning, and medical image processing and brain network analysis.

LOTFI SENHADJI (M'95–SM'99) received the Ph.D. degree in signal processing and telecommunications from the University of Rennes 1, Rennes, France, in 1993. He is currently a Professor and the Head of the INSERM Research Laboratory LTSI. He is also the Co-Director of the French-Chinese Laboratory CRIBs "Centre de Recherche en Information Biomédicale Sino-Français." He has published over 80 research papers in journals and conferences, and he contributed five handbooks. His main research efforts are focused on nonstationary signal processing with particular emphasis on wavelet transforms and time-frequency representations for detection, classification, and interpretation of biosignals. He is a Senior Member of the IEEE EMBS and the IEEE Signal Processing Society.

HUAZHONG SHU (M'00–SM'06) received the B.S. degree in applied mathematics from Wuhan University, China, in 1987, and the Ph.D. degree in numerical analysis from the University of Rennes 1, Rennes, France, in 1992. He is currently a Professor with the LIST Laboratory and the Co-Director of the CRIBs. His recent work concentrates on the image analysis, pattern recognition, and fast algorithms of digital signal processing. He is a Senior Member of the IEEE Society.