Sliding Conjugate Symmetric Sequency-Ordered Complex Hadamard Transform: Fast Algorithm and Applications

Jiasong Wu, Lu Wang, Guanyu Yang, Lotfi Senhadji, Limin Luo, Huazhong Shu

***Abstract*—This paper presents a fast algorithm for the computation of sliding conjugate symmetric sequency-ordered complex Hadamard transform (CS-SCHT). The algorithm calculates the values of window $i+N/4$ from those of window i, one length-$N/4$ Walsh Hadamard transform (WHT) and one length-$N/4$ Modified WHT (MWHT). The proposed algorithm requires $\mathrm{O}(N)$ arithmetic operations, which is more efficient than the block-based algorithms of various transforms and the sliding FFT algorithm, but less efficient than the sliding WHT algorithms. Compared to the recently proposed sliding inverse SCHT (ISCHT) algorithm, the proposed algorithm is more efficient for real input but less efficient for complex input. The applications of the sliding CS-SCHT in transform domain adaptive filtering (TDAF) to complex signal channel equalization and real speech signal acoustic echo cancellation are also provided.**

***Index Terms*—Conjugate symmetric sequency-ordered complex Hadamard transform, fast algorithm, sliding algorithm.**

I. Introduction

THE DISCRETE orthogonal transforms 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]. During the past years, the problem of the fast computation of these transforms has been extensively investigated [3]–[12]. At the same time, attention was also paid to finding new transforms and to developing their fast algorithms [13]–[22]. In particular, Bouguezel *et al.* [13], [14] proposed a new class of parametric transforms, including reciprocal-orthogonal WHT (RWHT), reciprocal-orthogonal DFT (RDFT), and reciprocal-orthogonal DHT (RDHT). Rahardja and Falkowski [15] derived a family of unified complex Hadamard transforms (UCHTs), which find their applications in many areas, such as multiple-valued logic design [16]. Aung *et al.* [17] introduced the so-called sequency-ordered complex Hadamard transform (SCHT), which find its applications in spectrum analysis [17], image watermarking [18], and shape-based image retrieval [19]. More recently, based on natural-ordered complex Hadamard transform (NCHT) [21], the same authors [22] introduced a new transform named conjugate symmetric SCHT (CS-SCHT) with the half spectrum property, which made it more suitable than SCHT for signal spectrum analysis. They further proposed a fast block-based decimation-in-sequency (DIS) algorithm for the computation of CS-SCHT [22], and showed that it can be an alternative to DFT and DCT in some applications requiring lower computational complexity such as spectrum estimation and image compression.

When dealing with a nonstationary process, such as speech, radar, biomedical, and communication signals, the commonly used method is sliding orthogonal transform, which is defined by [23], [24]

$$\mathbf{Y}_N(k,i)=\sum_{m=0}^{N-1}x_{i+m}w_m\psi(m,k), \tag{1}$$

where w_m is a window function, and $\{\psi(m,k)\}$ is an orthogonal basis set. $\mathbf{Y}_N(k,i)$ represent the orthogonal transform of the windowed signal around time i.

Since the computation of sliding transform is an intensive task, many fast algorithms have been proposed to speed up the computation efficiency [23]–[38]. By using the radix-2 decimation-in-time (DIT) FFT structure, Farhang-Boroujeny *et al.* [25], [26] derived a sliding FFT algorithm, which requires only N complex multipliers to update the N-point FFT for all N bins and is very suitable for serial-in serial-out implemental structure. Jacobsen and Lyons [27], [28] proposed the sliding DFT by using the circular shift property of DFT. Their algorithm is very different from sliding FFT and more suitable for parallelizing all N bins to construct the parallel-in parallel-out structure. The research on the fast computation of sliding WHT is also very active [29]–[34]. Farhang-Boroujeny [29] developed a radix-2 DIT sliding WHT algorithm. Hel-Or and Hel-Or [31] proposed a radix-2 DIS fast algorithm, which evaluates the projection values of a length-N WHT from that of two length-$N/2$ WHTs. Ben-Artzi *et al.* [32] further proposed a Gray Code Kernel (GCK) WHT algorithm, which is more efficient than the algorithms reported in [29] and [31] when a small number of projection values are computed. If all projection values are computed, their algorithm needs two more additions. Ouyang and Cham [33] presented a more efficient algorithm to compute the length-N WHT of window $i+N/4$

This work was supported by the National Basic Research Pogram of China under Grant 2011CB707904, the National Natural Science Foundation of China under Grants 60873048, 60911130370, 61073138, and 81101104, and the Natural Science Foundation of Jiangsu Province under Grant SBK 200910055.

J. Wu, L. Wang, G. Yang, L. Luo, and H. Shu are with the Laboratory of Image Science and Technology, School of Computer Science and Engineering, Southeast University, Nanjing 210096, China, and also with the Centre de Recherche en Information Biomédicale Sino-Français (CRIBs), Nanjing 210096, China (e-mail: jswu@seu.edu.cn; wanglu@seu.edu.cn; yang.list@seu.edu.cn; luo.list@seu.edu.cn; shu.list@seu.edu.cn).

L. Senhadji is with INSERM, U 1099, 35000 Rennes, France, with the Laboratoire Traitement du Signal et de l'Image (LTSI), Université de Rennes 1, 35000 Rennes, France, and with the Centre de Recherche en Information Biomédicale Sino-Français (CRIBs), 35000 Rennes, France (e-mail: lotfi.senhadji@univ-rennes1.fr).

TABLE I
SYMBOLS USED IN THE PAPER WITH A BRIEF DESCRIPTION

Symbols	Brief descriptions	Symbols	Brief descriptions	Symbols	Brief descriptions
$\mathbf{B}, \mathbf{B}', \mathbf{P}, \mathbf{Q}, \mathbf{Q}', \mathbf{R}, \mathbf{R}'$	Permutation matrices	$\mathbf{Y}$	Output vector	β	Constant
$\mathbf{C}$	NCHT matrix	A	Additions	$\sigma^2(k,i)$	Estimated input powers
$\mathbf{C}'$	Matrix related to WHT	$d_N(i)$	Intermediate variable related to input		Step-size
$\mathbf{D}, \mathbf{D}', \mathbf{I}', \mathbf{S}$	Diagonal matrices	$d'(i)$	Desired signal	$\psi(m,k)$	Orthogonal basis
$\mathbf{E}$	Matrix composed by sub-block identity matrices	$e(i)$	Error signal	ω	Angular frequency
$\mathbf{F}, \tilde{\mathbf{G}}, \hat{\mathbf{G}}$	Vectors related to elements of CS-SCHT matrix	$h(g,f)$	CS-SCHT matrix elements	Superscript H	Hermitian Transposition
$\mathbf{H}$	CS-SCHT matrix	M	Multiplications	Superscripts o and c	Row and column vectors
$\mathbf{I}$	Identity matrix	Me	Memory	Superscript T	Transposition
$\mathbf{J}$	reverse identity matrix	$t_N(k,i)$	Intermediate variable related to projection values	Superscript *	Complex conjugate
$\mathbf{W}$	WHT matrix	w_m	Window function	Superscripts *cCS, cISCHT, cW, cMW*	Complex CS-SCHT, ISCHT, WHT, and Modified WHT
$\mathbf{W}'$	Weight vector	$y_N(k,i)$	k^{th} CS-SCHT projection value for the i^{th} window	Superscripts *rCS, rISCHT, rW, rMW*	Real CS-SCHT, ISCHT, WHT, and Modified WHT
$\mathbf{X}$	Input vector	$z(i)$	Filter output signal		

from that of window i and one length-$N/4$ WHT. More recently, Wu *et al.* [37] proposed two fast algorithms for the computation of sliding inverse SCHT (ISCHT) by using the structures of radix-2 and radix-4 DIS fast ISCHT algorithms.

Since most signals in radar, sonar, and communications have in-phase and quadrature components, i.e., they are complex signals, which have nonsymmetrical power spectral density with respective to $\omega = 0$ and are more effectively processed by complex transforms [38, p. 224]. Even for some real input applications, they still need the phase information, for example, phase slope index (PSI) measure [39], [40], phase based image retrieval [41]. For more detail about the applications that need complex transform, please refer to [42]. Hence there is a need to develop the sliding complex transforms.

In this paper, we focus our attention on the fast computation of sliding CS-SCHT. The proposed algorithm computes the values of window $i + N/4$ from those of window i, one length-$N/4$ WHT and one length-$N/4$ Modified WHT (MWHT). A preliminary study was presented in [43], we expand this idea here and also provide a rigorous mathematical proof of the algorithm as well as an in-depth analysis of its computational complexity. Application to complex signal channel equalization and real speech signal filtering is discussed. The rest of the paper is organized as follows. In Section II, preliminaries about the sliding CS-SCHT are given. The proposed sliding CS-SCHT algorithm is described and the comparison results with other algorithms are provided in Section III. Transform domain adaptive filtering for complex signal channel equalization and real speech signal filtering is given in Section IV to illustrate the potential applications of sliding CS-SCHT. Section V concludes the paper. In Table I we give a list of variables and symbols used in this paper together with a brief description.

II. PRELIMINARY

Let $\mathbf{X}_N(i) = [x_i, x_{i+1}, \ldots, x_{i+N-1}]^T$ and $\mathbf{Y}_N(i) = [y_i, y_{i+1}, \ldots, y_{i+N-1}]^T$ be respectively the complex or real input vector and the corresponding transformed vector of the ith window, where the superscript T denotes the transpose, the length-N forward and backward sliding CS-SCHT are respectively defined as [22]

$$\mathbf{Y}_N(i) = \mathbf{H}_N \mathbf{X}_N(i), \tag{2}$$

$$\mathbf{X}_N(i) = \frac{1}{N} \mathbf{H}_N^H \mathbf{Y}_N(i), \tag{3}$$

where the length N is assumed to be a power of two, i.e., $N = 2^n, n \geq 1$, the superscript H denotes the Hermitian transposition. $\mathbf{H}_N$ is the order-N CS-SCHT matrix whose elements are given by

$$h(g,f) = (-1)^{\tilde{\mathbf{G}} \cdot \mathbf{F}} (-j)^{\hat{\mathbf{G}} \cdot \mathbf{F}}, \quad 0 \leq g, \quad f \leq 2^n - 1, \quad n = \log_2 N, \tag{4}$$

where $\tilde{\mathbf{G}} = (\tilde{g}_{n-1}, \ldots, \tilde{g}_1, \tilde{g}_0)$, $\hat{\mathbf{G}} = (\hat{g}_{n-1}, \ldots, \hat{g}_1, \hat{g}_0)$, and $\mathbf{F} = (f_{n-1}, \ldots, f_1, f_0)$. The dot "$\cdot$" denotes the inner product of two vectors. g_r and f_r are, respectively, the binary representation of g and $f, r = 0, 1, \ldots, n-1$, being the index of the binary bit position. $\tilde{g}_r$ is a binary gray code of the bit reversal of g_r and $\hat{g}_r$ is the rth bit of the binary bits of the highest power of 2 in $c(g)/2$ where $c(g)$ is the decimal number obtained through a bit-reversed conversion of the decimal g.

From (4), we have

$$\mathbf{H}_1 = [1],$$

$$\mathbf{H}_2 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}, \quad \mathbf{H}_4 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & j & -1 & -j \\ 1 & -1 & 1 & -1 \\ 1 & -j & -1 & j \end{bmatrix},$$

$$\mathbf{H}_8 = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & j & j & -1 & -1 & -j & -j \\ 1 & j & -1 & -j & 1 & j & -1 & -j \\ 1 & -1 & -j & j & -1 & 1 & j & -j \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & -1 & j & -j & -1 & 1 & -j & j \\ 1 & -j & -1 & j & 1 & -j & -1 & j \\ 1 & 1 & -j & -j & -1 & -1 & j & j \end{bmatrix}. \tag{5}$$

Let us introduce some notations:

$$\mathbf{H}_N = [\mathbf{H}_N^o(0), \mathbf{H}_N^o(1), \ldots, \mathbf{H}_N^o(N-1)]^T, \quad (6)$$

$$\mathbf{H}_N = [\mathbf{H}_N^c(0), \mathbf{H}_N^c(1), \ldots, \mathbf{H}_N^c(N-1)], \quad (7)$$

$$\begin{aligned}\mathbf{H}_N^{1/m} &= [\mathbf{H}_N^c(0), \mathbf{H}_N^c(1), \ldots, \mathbf{H}_N^c(N/m-1)] \\ &= \left[\mathbf{H}_N^{1/m}(0), \mathbf{H}_N^{1/m}(1), \ldots, \mathbf{H}_N^{1/m}(N-1)\right]^T, \\ & \qquad m = 2, 4, \end{aligned} \quad (8)$$

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

$$\begin{aligned}\mathbf{H}_4^{1/4} &= [\mathbf{H}_4^c(0)] = [1, 1, 1, 1]^T \\ &= \left[\mathbf{H}_4^{1/4}(0), \mathbf{H}_4^{1/4}(1), \mathbf{H}_4^{1/4}(2), \mathbf{H}_4^{1/4}(3)\right]^T, \end{aligned} \quad (9)$$

$$\begin{aligned}\mathbf{H}_8^{1/4} &= [\mathbf{H}_8^c(0), \mathbf{H}_8^c(1)] \\ &= \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & j & -1 & -1 & -1 & -j & 1 \end{bmatrix}^T \\ &= \left[\mathbf{H}_8^{1/4}(0), \mathbf{H}_8^{1/4}(1), \ldots, \mathbf{H}_8^{1/4}(7)\right]^T. \end{aligned} \quad (10)$$

Let $y_N(k, i)$ be the kth CS-SCHT projection value for the ith window:

$$\begin{aligned} y_N(k, i) &= \mathbf{H}_N^o(k)\mathbf{X}_N(i), \\ & \qquad \text{for } k = 0, 1, \ldots N-1; \\ i &= 0, 1, \ldots, M-N, N = 2^n, n \geq 1, \end{aligned} \quad (11)$$

where M is the length of the input data sequence.

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

$$y_N(N-k, i) = y_N^*(k, i), \quad k = 1, 2, \ldots, N/2-1, \quad (12)$$

where the superscript * denotes the complex conjugate.

III. Fast Algorithm for Sliding CS-SCHT

In this section, we first derive a relationship between the CS-SCHT and WHT matrices, and then propose a fast algorithm for computing the sliding CS-SCHT.

In [22], Aung *et al.* derived a matrix decomposition of $\mathbf{H}_N$ as follows:

$$\mathbf{H}_N = \mathbf{R}_N\mathbf{C}_N, \quad (13)$$

where $\mathbf{R}_N$ is a permutation matrix, which permutes the NCHT matrix $\mathbf{C}_N$ to CS-SCHT matrix $\mathbf{H}_N$, and

$$\mathbf{C}_N = \begin{bmatrix} \mathbf{C}_{N/2} & \\ & \mathbf{C}'_{N/2}\mathbf{S}_{N/2} \end{bmatrix} \mathbf{E}_N, \quad (14)$$

$$\mathbf{E}_N = \begin{bmatrix} \mathbf{I}_{N/2} & \mathbf{I}_{N/2} \\ \mathbf{I}_{N/2} & -\mathbf{I}_{N/2} \end{bmatrix}, \quad (15)$$

$$\mathbf{S}_{N/2} = \begin{bmatrix} \mathbf{I}_{N/4} & \\ & j\mathbf{I}_{N/4} \end{bmatrix}, \quad (16)$$

$$\mathbf{C}'_{N/2} = \begin{bmatrix} \mathbf{C}'_{N/4} & \mathbf{C}'_{N/4} \\ \mathbf{C}'_{N/4}\mathbf{I}'_{N/4} & -\mathbf{C}'_{N/4}\mathbf{I}'_{N/4} \end{bmatrix}, \quad (17)$$

$$\mathbf{I}'_{N/4} = \begin{bmatrix} \mathbf{I}_{N/8} & \\ & -\mathbf{I}_{N/8} \end{bmatrix}, \quad (18)$$

where $\mathbf{I}_N$ is the identity matrix.

TABLE II
Fast Algorithm for Length-4 CS-SCHT

	x_i	x_{i+1}	x_{i+2}	x_{i+3}	x_{i+4}	Proposed algorithm
$y_4(0,i)$	1	1	1	1		$y_4(0,i+1) = y_4(0,i) - (x_i - x_{i+4})$
$y_4(0,i+1)$		1	1	1	1	
$y_4(1,i)$	1	j	-1	$-j$		$y_4(1,i+1) = -j[y_4(1,i) - (x_i - x_{i+4})]$
$y_4(1,i+1)$		1	j	-1	$-j$	
$y_4(2,i)$	1	-1	1	-1		$y_4(2,i+1) = -[y_4(2,i) - (x_i - x_{i+4})]$
$y_4(2,i+1)$		1	-1	1	-1	
$y_4(3,i)$	1	$-j$	-1	j		$y_4(3,i+1) = j[y_4(3,i) - (x_i - x_{i+4})]$
$y_4(3,i+1)$		1	$-j$	-1	j	

In the following, we derive another matrix decomposition of $\mathbf{H}_N$. Using (14) and the properties

$$\mathbf{R}_N^{-1} = \mathbf{R}_N, \quad \mathbf{C}_{N/2} = \mathbf{R}_{N/2}^{-1}\mathbf{H}_{N/2}. \quad (19)$$

Equation (13) becomes

$$\begin{aligned}\mathbf{H}_N = \mathbf{R}_N &\begin{bmatrix} \mathbf{R}_{N/2} & \\ & \mathbf{R}_{N/2} \end{bmatrix} \\ &\times \begin{bmatrix} \mathbf{H}_{N/2} & \\ & \mathbf{R}_{N/2}\mathbf{C}'_{N/2}\mathbf{S}_{N/2} \end{bmatrix} \mathbf{E}_N. \end{aligned} \quad (20)$$

It can be proved that (20) is equivalent to (the proof is shown in Appendix A)

$$\mathbf{H}_N = \mathbf{P}_N \begin{bmatrix} \mathbf{H}_{N/2} & \\ & \mathbf{Q}_{N/2}\mathbf{P}_{N/2}^T\mathbf{W}_{N/2}\mathbf{S}_{N/2} \end{bmatrix} \mathbf{E}_N, \quad (21)$$

where $\mathbf{S}_{N/2}$ is defined in (16), $\mathbf{W}_N$ is the order-N WHT matrix, and

$$\mathbf{Q}_{N/2} = \begin{bmatrix} \mathbf{I}_{N/4} & \\ & \mathbf{J}_{N/4} \end{bmatrix}, \quad (22)$$

$$\begin{aligned}&[x_i, x_{i+N/2}, x_{i+1}, x_{i+N/2+1}, \ldots, x_{i+N/2-1}, x_{i+N-1}]^T \\ &\quad = \mathbf{P}_N[x_i, x_{i+1}, \ldots, x_{i+N-1}]^T, \end{aligned} \quad (23)$$

where $\mathbf{J}_N$ is the reverse identity matrix, that is, all the elements of vice diagonal line are one, $\mathbf{P}_N$ is the perfect (or ideal) shuffle permutation matrix [44], which can be implemented by linear time, in-place algorithms [45], [46], and

$$\mathbf{P}_N^{-1} = \mathbf{P}_N^T. \quad (24)$$

For example,

$$\mathbf{P}_8 = \begin{bmatrix} 1 & & & & & & & \\ & & & 1 & & & & \\ & 1 & & & & & & \\ & & & & 1 & & & \\ & & 1 & & & & & \\ & & & & & 1 & & \\ & & & 1 & & & & \\ & & & & & & & 1 \end{bmatrix}. \quad (25)$$

Since [9]

$$\mathbf{P}_{N/2}^T\mathbf{W}_{N/2} = \begin{bmatrix} \mathbf{W}_{N/4} & \mathbf{D}_{N/4}\mathbf{W}_{N/4} \\ \mathbf{W}_{N/4} & -\mathbf{D}_{N/4}\mathbf{W}_{N/4} \end{bmatrix}, \quad (26)$$

TABLE III
FAST ALGORITHM FOR LENGTH-8 CS-SCHT

	x_i	x_{i+1}	x_{i+2}	x_{i+3}	x_{i+4}	x_{i+5}	x_{i+6}	x_{i+7}	x_{i+8}	x_{i+9}	Proposed algorithm
$y_8(0,i)$	1	1	1	1	1	1	1	1			$y_8(0,i+2)=$
$y_8(0,i+2)$			1	1	1	1	1	1	1	1	$y_8(0,i)-[(x_i - x_{i+8})+(x_{i+1} - x_{i+9})]$
$y_8(1,i)$	1	1	j	j	-1	-1	$-j$	$-j$			$y_8(1,i+2)=$
$y_8(1,i+2)$			1	1	j	j	-1	-1	$-j$	$-j$	$-j\{y_8(1,i)-[(x_i - x_{i+8})+(x_{i+1} - x_{i+9})]\}$
$y_8(2,i)$	1	j	-1	$-j$	1	j	-1	$-j$			$y_8(2,i+2)=$
$y_8(2,i+2)$			1	j	-1	$-j$	1	j	-1	$-j$	$-\{y_8(2,i)-[(x_i - x_{i+8})+j(x_{i+1} - x_{i+9})]\}$
$y_8(3,i)$	1	-1	$-j$	j	-1	1	j	$-j$			$y_8(3,i+2)=$
$y_8(3,i+2)$			1	-1	$-j$	j	-1	1	j	$-j$	$j\{y_8(3,i)-[(x_i - x_{i+8})-(x_{i+1} - x_{i+9})]\}$
$y_8(4,i)$	1	-1	1	-1	1	-1	1	-1			$y_8(4,i+2)=$
$y_8(4,i+2)$			1	-1	1	-1	1	-1	1	-1	$y_8(4,i)-[(x_i - x_{i+8})-(x_{i+1} - x_{i+9})]$
$y_8(5,i)$	1	-1	j	$-j$	-1	1	$-j$	j			$y_8(5,i+2)=$
$y_8(5,i+2)$			1	-1	j	$-j$	-1	1	$-j$	j	$-j\{y_8(5,i)-[(x_i - x_{i+8})-(x_{i+1} - x_{i+9})]\}$
$y_8(6,i)$	1	$-j$	-1	j	1	$-j$	-1	j			$y_8(6,i+2)=$
$y_8(6,i+2)$			1	$-j$	-1	j	1	$-j$	-1	j	$-\{y_8(6,i)-[(x_i - x_{i+8})-j(x_{i+1} - x_{i+9})]\}$
$y_8(7,i)$	1	1	$-j$	$-j$	-1	-1	j	j			$y_8(7,i+2)=$
$y_8(7,i+2)$			1	1	$-j$	$-j$	-1	-1	j	j	$j\{y_8(7,i)-[(x_i - x_{i+8})+(x_{i+1} - x_{i+9})]\}$

(21) can also be expressed as

$$\mathbf{H}_N = \mathbf{P}_N \begin{bmatrix} \mathbf{H}_{N/2} & \\ & \mathbf{Q}_{N/2} \begin{bmatrix} \mathbf{W}_{N/4} & \mathbf{D}_{N/4}\mathbf{W}_{N/4} \\ \mathbf{W}_{N/4} & -\mathbf{D}_{N/4}\mathbf{W}_{N/4} \end{bmatrix} \mathbf{S}_{N/2} \end{bmatrix} \mathbf{E}_N, \tag{27}$$

where $\mathbf{D}_N$ is a diagonal matrix whose elements alternate between $+1$ and -1.

We are now ready to propose our fast algorithm, which computes the values of length-N sliding CS-SCHT of window $i + N/4$ from those of window i and one length-$N/4$ WHT and one length-$N/4$ MWHT.

A. Fast Algorithm for $N = 4$

The proposed algorithm is shown in Table II, from which we have

$$y_4(k, i+1) = (-j)^k [y_4(k,i) - t_4(k,i)], \quad k = 0, 1, 2, 3, \tag{28}$$

$$[t_4(0,i), t_4(1,i), t_4(2,i), t_4(3,i)]^T = \mathbf{H}_4^{1/4}[d_4(i)] = [d_4(i), d_4(i), d_4(i), d_4(i)]^T, \tag{29}$$

$$d_4(i) = x_i - x_{i+4}, \tag{30}$$

where $\mathbf{H}_4^{1/4}$ is shown in (9). For complex input data, 2 multiplications with j, 10 real additions, and a memory size of 10 are needed. For the real input data, from (12) and (28), we have

$$y_4(0, i+1) = [y_4(0,i) - t_4(0,i)], \tag{31}$$

$$y_4(1, i+1) = -j[y_4(1,i) - t_4(1,i)] = y_4^*(3, i+1), \tag{32}$$

$$y_4(2, i+1) = -[y_4(2,i) - t_4(2,i)], \tag{33}$$

Since $y_4(0,i), y_4(2,i)$ and $t_4(k,i), k = 0, 1, 2$ are real values, but $y_4(1,i)$ is a complex value, for the implementation of $y_4(1,i) - t_4(1,i)$, we can just read the real part of $y_4(1,i)$ and then subtract the real value $t_4(1,i)$. Therefore, 1 multiplication with j, 4 real additions, and a memory size of 5 are needed for real input data.

B. Fast Algorithm for $N = 8$

The proposed algorithm is shown in Table III, from which we have

$$y_8(k, i+2) = (-j)^k [y_8(k,i) - t_8(k,i)], \; k = 0, 1, \ldots, 7, \tag{34}$$

$$[t_8(0,i), t_8(1,i), \ldots, t_8(7,i)]^T = \mathbf{H}_8^{1/4}[d_8(i), d_8(i+1)]^T, \tag{35}$$

$$\mathbf{H}_8^{1/4} = \mathbf{P}_8 \begin{bmatrix} \mathbf{H}_4^{1/2} \\ \mathbf{W}_2 \\ \mathbf{J}_2\mathbf{W}_2 \end{bmatrix} = \mathbf{P}_8 \begin{bmatrix} \mathbf{P}_4 \begin{bmatrix} \mathbf{H}_2 \\ \mathbf{W}_2\mathbf{S}_2 \end{bmatrix} \\ \mathbf{W}_2 \\ \mathbf{J}_2\mathbf{W}_2 \end{bmatrix}, \tag{36}$$

$$\mathbf{S}_2 = \begin{bmatrix} 1 & \\ & j \end{bmatrix}, \tag{37}$$

$$d_8(i+u) = x_{i+u} - x_{i+8+u}, \quad u = 0, 1, \tag{38}$$

where $\mathbf{H}_8^{1/4}$ and $\mathbf{P}_8$ are shown in (10) and (23), respectively. For complex input data, 5 multiplications with j, 26 real additions, and 36 size of memory are needed. For real input data, similar to the analysis of length-4 CS-SCHT, 3 multiplications with j, 10 real additions, and 16 size of memory are needed.

C. Fast Algorithm for $N = 2^n, n \geq 3$

By using the same strategy as for $N = 4$ and $N = 8$, we have

$$y_N(k, i+N/4) = (-j)^k [y_N(k,i) - t_N(k,i)], \quad k = 0, 1, \ldots, N-1, \tag{39}$$

$$[t_N(0,i), t_N(1,i), \ldots, t_N(N-1,i)]^T = \mathbf{H}_N^{1/4}[d_N(i), d_N(i+1), \ldots, d_N(i+N/4-1)]^T, \tag{40}$$

$$\mathbf{H}_N^{1/4} = \mathbf{P}_N \begin{bmatrix} \mathbf{H}_{N/2}^{1/2} \\ \mathbf{W}_{N/4} \\ \mathbf{J}_{N/4}\mathbf{W}_{N/4} \end{bmatrix} = \mathbf{P}_N \begin{bmatrix} \mathbf{P}_{N/2} \begin{bmatrix} \mathbf{H}_{N/4} \\ \mathbf{Q}_{N/4}\mathbf{P}_{N/4}^T\mathbf{W}_{N/4}\mathbf{S}_{N/4} \end{bmatrix} \\ \mathbf{W}_{N/4} \\ \mathbf{J}_{N/4}\mathbf{W}_{N/4} \end{bmatrix}, \tag{41}$$

$$d_N(i+u) = x_{i+u} - x_{i+N+u}, u = 0, 1, \ldots, N/4-1, \tag{42}$$

where $\mathbf{H}_N^{1/4}$ and $\mathbf{H}_N^{1/2}$ are defined in (8). $\mathbf{S}_N, \mathbf{Q}_N$, and $\mathbf{P}_N$ are defined in (16), (22), and (23), respectively.

The derivation of (39) is given in Appendix B. Fig. 1 shows the signal graph of the proposed algorithm, whose computational complexity and memory storage requirement are analyzed as follows:

1) The computation of (42) for $u = N/4 - 1$ needs only 2 real additions for complex input data (1 real addition for

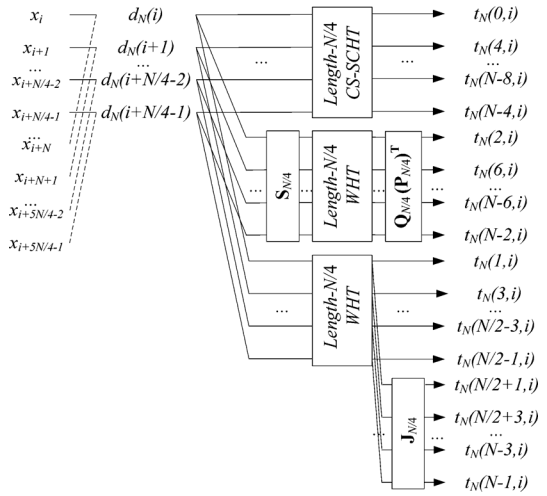


Fig. 1. Signal flow graph of the length-N sliding CS-SCHT transform.

real input data). Note that the values of $d_N(i+u), u = 0,1,\ldots,N/4-2$, have already been obtained during the computation of $y_N(k,i+v), v = 1,2,\ldots,N/4-1$, respectively. A memory size of $N/2$ for complex input data ($N/4$ for real input data) is required for storing $d_N(i+u)$, $u = 0,1,\ldots,N/4-1$. The input x_{i+u} and x_{i+N+u} for $u = 0,1,\ldots,N/4-1$, needs N memory for complex input data ($N/2$ for real input data), which can be released after performing (42) since it will not be used in the following steps.

2) The computation of (41) needs one length-$N/4$ CS-SCHT, one length-$N/4$ WHT, which can be computed by the algorithms [29], [31]–[33], one length-$N/4$ modified WHT ($\mathbf{W}_{N/4}\mathbf{S}_{N/4}$). Note that for the modified WHT, the input data is first multiplied by $\mathbf{S}_{N/4}$, resulting in the change of two inputs: $d_N(i)$ replaced by $jd_N(i+N/4)$ and $jd_N(i+N/8)$ by $d_N(i+N/8)$. This change makes the implementation of $\mathbf{W}_{N/4}$ $\mathbf{S}_{N/4}$ not exactly the same as that of $\mathbf{W}_{N/4}$. It seems that the algorithms in [31]–[33] are difficult to deal with the modified WHT. However, we notice that the two changed inputs are just that of necessary updated in the algorithm of [29], which can be chosen to implement the modified WHT, costing 2 additional multiplications with j when compared to WHT. For the implementation, size $3N/2$ memory for complex input data is needed for storing the values $t_N(k,i+u), u \in \{0,1,\ldots,N/2-2,N/2-1\} \cup \{N/2,N/2+2,\ldots,N-4,N-2\}$, since $\mathbf{J}_{N/4}$ $\mathbf{W}_{N/4}$ is just row change operations of $\mathbf{W}_{N/4}$. (Size $3N/4$ memory for real input data for $t_N(k,i+u), u \in \{0,4,\ldots,N/2-4,N/2\} \cup \{2,6,\ldots,N/2-6,N/2-2\} \cup \{1,3,\ldots,N/2-3,N/2-1\}$.) We also assume that the memory storage requirements of length-$N/4$ complex (real) CS-SCHT, length-$N/4$ complex (real) WHT, and length-$N/4$ complex (real) modified WHT are $Me_{N/4}^{cCS}(Me_{N/4}^{rCS})$, $Me_{N/4}^{cW}$ $(Me_{N/4}^{rW})$, and $Me_{N/4}^{cMW}$ $(Me_{N/4}^{rMW})$, respectively. Note that the multiplication by j or $-j$ can be realized by switching the real and imaginary parts of the input with one sign changing, so that there is no memory requirement.

3) The computation of (39) needs $N/2$ multiplications with j and $2N$ real additions for complex input data ($N/4$ multiplications with j and $3N/4$ real additions for real input data). The values of $y_N(k,i), y_N(k,i+1),\ldots,y_N(k,i+N/4-1)$ can be obtained by simply using the CS-SCHT block algorithm [22]. For the implementation, we first distribute $2N$ memory for $y_N(k,i), k = 0,1,\ldots,N-1$ for complex input data, which is then overlaid by $y_N(k,i+N/4), k = 0,1,\ldots,N-1$ after performing (39). (For the real input data, we should distribute $3N/4$ memory for $\mathrm{Re}\{y_N(k,i)\}, k \in \{0,N/2\} \cup \{1,3,\ldots,N/2-3,N/2-1\}$ and $y_N(k,i), k \in \{2,4,6,\ldots,N/2-4,N/2-2\}$.)

Thus, the computational complexity and memory requirement of the proposed algorithm for complex and real input data is given by

$$M_N^{cCS} = M_{N/4}^{cCS} + N/2 + 2, \tag{43}$$

$$A_N^{cCS} = A_{N/4}^{cCS} + A_{N/4}^{cW} + A_{N/4}^{cMW} + 2N + 2, \tag{44}$$

$$Me_N^{cCS} = N/2 + \max\left\{7N/2, Me_{N/4}^{cCS} + Me_{N/4}^{cW} + Me_{N/4}^{cMW}\right\}, \tag{45}$$

where M_N^{cCS} and A_N^{cCS} mean the multiplications and additions needed by length-N complex CS-SCHT, $A_{N/4}^{cW}$ and $A_{N/4}^{cMW}$ mean the additions needed by length-$N/4$ complex WHT and MWHT, respectively, and the initial values are $M_4^{cCS} = 2, A_4^{cCS} = 10; M_8^{cCS} = 5, A_8^{cCS} = 26$, and $Me_4^{cCS} = 10, Me_8^{cCS} = 36$.

$$M_N^{rCS} = M_{N/4}^{rCS} + N/4 + 2, \tag{46}$$

$$A_N^{rCS} = A_{N/4}^{rCS} + A_{N/4}^{rW} + A_{N/4}^{rMW} + 3N/4 + 1, \tag{47}$$

$$Me_N^{rCS} = N/4 + \max\left\{3N/2, Me_{N/4}^{rCS} + Me_{N/4}^{rW} + Me_{N/4}^{rMW}\right\}, \tag{48}$$

where M_N^{rCS} and A_N^{rCS} mean the multiplications and additions needed by length-N real CS-SCHT, $A_{N/4}^{rW}$ and $A_{N/4}^{rMW}$ mean the additions needed by length-$N/4$ real WHT and MWHT, respectively, and the initial values are $M_4^{rCS} = 1, A_4^{rCS} = 4; M_8^{rCS} = 3, A_8^{rCS} = 10$; and $Me_4^{rCS} = 5, Me_8^{rCS} = 16$.

Since we use the algorithm presented in [29] to compute the Modified WHT, we have

$$A_{N/4}^{cMW} = N - 4, \tag{49}$$

$$Me_{N/4}^{cMW} = (N/2)\log_2 N - 5N/4, \tag{50}$$

and

$$A_{N/4}^{rMW} = A_{N/4}^{cMW} - 4 = N - 8, \tag{51}$$

$$Me_{N/4}^{rMW} = Me_{N/4}^{cMW} - N/2 = (N/2)\log_2 N - 7N/4. \tag{52}$$

In the following, we discuss two different ways for computing the length-$N/4$ WHT. The computational complexity and the memory storage requirements of the sliding WHT algorithms in [29] and [33], we use in the following, are shown in Table V.

Scheme 1: Implementation of the Length-$N/4$ WHT by [33] and the Length-$N/4$ MWHT by [29]: From (44), (45), (49), (50), and Table IV, we have (53) and (54) at the bottom of the

page. From (47), (48), (51), (52), and Table V, we have (55) and (56) at the bottom of the page.

Scheme 2: Implementation Both the Length-N/4 WHT and the MWHT by [29]: From (44), (45), (49), (50) and Table IV, we have

$$\begin{aligned} A_N^{cCS} &= A_{N/4}^{cCS} + (N-4) + (N-4) + 2N + 2 \\ &= A_{N/4}^{cCS} + 4N - 6 \end{aligned} \tag{57}$$

$$Me_N^{cCS} = \frac{N}{2} + \max\left\{\frac{7}{2}N, Me_{N/4}^{cCS} + N\left(\log_2 N - \frac{5}{2}\right)\right\} \tag{58}$$

From (47), (48), (51), (52) and Table V, we have

$$\begin{aligned} A_N^{rCS} &= A_{N/4}^{rCS} + (N/2-2) + 7N/4 - 7 \\ &= A_{N/4}^{rCS} + 9N/4 - 9 \end{aligned} \tag{59}$$

$$\begin{aligned} Me_N^{rCS} = \frac{N}{4} + \max\Bigg\{&\frac{3}{2}N, Me_{N/4}^{rCS} \\ &+ \left(\frac{3N}{4}\log_2 N - \frac{19}{8}N\right)\Bigg\} \end{aligned} \tag{60}$$

It can be seen from (53) to (60) that *Scheme 1* requires less number of additions and memory complexity than *Scheme 2*. However, *Scheme 1* needs two different modules to implement length-$N/4$ WHT and MWHT, while *Scheme 2* only needs one module which makes its implementation more simpler than the one of *Scheme 1*. Note that length-$N/4$ WHT can also be implemented by GCK algorithm [32], whose most important advantage is that it requires less computation complexity than [29] and [31] when only a small number of projection values are computed, however, two more additions are needed when all projection values are computed. Since in the proposed algorithm, all projection values are needed, so, it seems more suitable to use [29] and [33] than GCK algorithm [32] in terms of computational complexity.

Compared to our previous conference paper [43], we mainly have the following four improvements: 1) We provided a rigorous mathematical proof of the algorithm and also applied the algorithm to complex signal channel equalization and real speech signal filtering. 2) We reanalyzed the computational complexity of sliding WHT in [33], which leads to the reduced computational complexity compared to [43]. 3) We provided the computational complexity analysis of sliding CS-SCHT algorithm for real input, for which the conjugate symmetric property can be used to reduce the computational complexity significantly. 4) The expression of (41), which is the most important equation in the proposed algorithm, is optimized than that of (6) in [43], which leads to more regular and fast permutation operations.

Note that the proposed sliding CS-SCHT algorithm shares the same idea as that of sliding ISCHT one [37], that is, computing the projection values of window $i + N/4$ from those of window i. However, the construction of CS-SCHT matrix is different from that of ISCHT matrix. In fact, the CS-SCHT matrix is generated based on the WHT matrix and direct block matrix operation while ISCHT matrix is generated based on the products of the row vectors of complex Rademacher matrices [17], [22]. Therefore, the sliding fast algorithms for CS-SCHT and ISCHT are also different. The key of the proposed sliding CS-SCHT algorithm is to establish the relationships between length-N CS-SCHT matrix and the length-$N/4$ WHTs, the latter can be computed by many mature sliding algorithms [29], [31]–[33]. However, the sliding ISCHT algorithm is based on the relationships between length-N ISCHT matrix and the length-$N/4$ ISCHTs.

The comparison results of the proposed algorithm and the algorithms in [13], [14], [22], [25], [26], [29], [31]–[33], [37] are shown in Tables IV (complex input) and V (real input). It can be seen from the tables that the proposed algorithm reduces significantly the real additions compared to the block CS-SCHT algorithm [22], block parametric WHT algorithm [13], and block parametric DFT/DHT algorithm [14], but at the cost of more memory requirement. For complex input, the proposed sliding CS-SCHT algorithm is less efficient than that of sliding ISCHT [37]; however, for real input, it is more efficient than that of [37] owing to the conjugate symmetric property of CS-SCHT shown in (12). The proposed algorithm is more efficient than the

$$\begin{aligned} A_N^{cCS} &= A_{N/4}^{cCS} + A_{N/4}^{cW} + (N-4) + 2N + 2 \\ &= \begin{cases} A_{N/4}^{cCS} + \left(\dfrac{11}{3}N + 2\log_4 N - \dfrac{20}{3}\right), N = 2^n, n = 4, 6, \ldots \\ A_{N/4}^{cCS} + \left(\dfrac{11}{3}N + 2\log_4\left(\dfrac{N}{2}\right) - \dfrac{16}{3}\right), N = 2^n, n = 5, 7, \ldots \end{cases} \end{aligned} \tag{53}$$

$$Me_N^{cCS} = \frac{N}{2} + \max\left\{\frac{7}{2}N, Me_{N/4}^{cCS} + \frac{N}{2}(\log_2 N - 1)\right\}. \tag{54}$$

$$\begin{aligned} A_N^{rCS} &= A_{N/4}^{rCS} + A_{N/4}^{rW} + 7N/4 - 7 \\ &= \begin{cases} A_{N/4}^{rCS} + (25N/12 + \log_4 N - 28/3), & N = 2^n, \quad n = 4, 6, \ldots \\ A_{N/4}^{rCS} + (25N/12 + \log_4(N/2) - 26/3), & N = 2^n, \quad n = 5, 7, \ldots \end{cases} \end{aligned} \tag{55}$$

$$Me_N^{rCS} = \frac{N}{4} + \max\left\{\frac{3}{2}N, Me_{N/4}^{rCS} + \left(\frac{N}{2}\log_2 N - \frac{11}{8}N\right)\right\}. \tag{56}$$

TABLE IV
COMPARISON RESULTS OF THE PROPOSED ALGORITHM OF THE COMPLEX INPUT DATA WITH THE BLOCK-BASED ONE [22], SLIDING ISCHT [37], THE SLIDING FFT [25], [26], AND RECIPROCAL-ORTHOGONAL DFT TRANSFORM [14] FOR $N = 2^n, n \geq 4$. "$Muls$" REPRESENTS REAL MULTIPLICATIONS, "$Muls(j)$" MEANS MULTIPLICATION WITH j, "$Adds$" MEANS REAL ADDITIONS. "*ME*" DENOTES MEMORY (WORDS). SUPERSCRIPT "#"DENOTES "$Muls(j)$"

Complex input		4	8	16	32	$N = 2^n$, $n \geq 4$
Proposed algorithm	*Muls(j)*	2	5	12	23	$2N/3+2\log_4 N-8/3$, $N=2^n$, n=4,6,… $2N/3+2\log_4(N/2)-7/3$, $N=2^n$, n=5,7,…
	Adds	10	26	66	142	$44N/9+(\log_4 N-14/3)(\log_4 N-1)-86/9$, $N=2^n$, n=4,6,… $44N/9+(\log_4(N/2)-10/3)(\log_4(N/2)-1)-118/9$, $N=2^n$, n=5,7,…
	Me	10	36	64	128	$N/2+\max\{7N/2,\ Me^{cCS}_{N/4}+N(\log_2 N-1)/2\}$
Block CS-SCHT algorithm [22]	*Muls(j)*	1	3	7	15	$N/2-1$
	Adds	16	48	128	320	$2N\log_2 N$
	Me	8	16	32	64	$2N$
Sliding ISCHT [37]	*Muls(j)*	2	5	13	28	$5N/6+\log_4 N-7/3$, $N=2^n$, n=4,6,… $5N/6+\log_4(N/2)-8/3$, $N=2^n$, n=5,7,…
	Adds	10	26	56	120	$4N-2\log_4 N-4$, $N=2^n$, n=4,6,… $4N-2\log_4(N/2)-4$, $N=2^n$, n=5,7,…
	Me	10	28	56	112	$N/2+\max\{3N,\ Me^{cISCHT}_{N/4}+N(\log_2 N-5/2)/2\}$
Sliding FFT [25, 26]	*Muls*	0	8	32	88	$4N-8\log_2 N$
	Muls(j)	1	2	3	4	$\log_2 N-1$
	Adds	6	18	46	106	$4N-4\log_2 N-2$
	Me	12	44	124	316	$2N\log_2 N-4$
Reciprocal-orthogonal DFT (RDFT) [14]	*Muls*	$1^{\#}$	4	16	68	$(4/3)N\log_2 N-(44/9)N-(10/9)(-1)^{\log_2 N}+10$
	Shifts	0	0	8	16	$(2/3)N+(4/3)(-1)^{\log_2 N}-4$
	Adds	16	52	144	372	$(24/9)N\log_2 N-(16/9)N+(24/9)\log_2 N-(24/9)(-1)^{\log_2 N}\log_2 N+(118/9)(-1)^{\log_2 N}-34/3$
	Me	8	18	48	96	$3N$

sliding FFT in [25] and [26]. This is because the proposed algorithm only needs the multiplications with j and real additions. The proposed algorithm also requires less memory complexity than sliding FFT [25], [26]. But it requires more computational and memory complexity than that of sliding WHT algorithms shown in [29], [31]–[33]. For comparison purpose, Tables IV and V show the real multiplications, multiplications with j and real additions where one complex multiplication (or one rotational matrix) is implemented by four real multiplications and two real additions. Note that for the particularity of multiplications with j, we count once whatever j multiplied by a complex number or a real number.

IV. TWO APPLICATION EXAMPLES

In this section, we provide two application examples of the sliding CS-SCHT.

Transform domain least-mean-square adaptive filters (TDLMSAF), introduced by Narayan *et al.* [47], exploit the de-correlation properties of some well-known signal transforms such as DFT, DCT, DHT and WHT, in order to pre-whiten the input data and speed up filter convergence (p. 413, [48]).

Similar to the DFT domain LMS adaptive filter [47], [48], the CS-SCHT domain LMS adaptive filter algorithm, shown in Fig. 2, is described as follows:

$$\begin{aligned} \mathbf{Y}_N(i) &= \mathbf{H}_N \mathbf{X}_N(i), \\ z(i) &= (\mathbf{W}'_N(i))^H \mathbf{Y}_N(i), \quad e(i) = d'(i) - z(i), \end{aligned} \tag{61}$$

$$\mathbf{W}'_N(i+1) = \mathbf{W}'_N(i) + 2\,\mu(\mathbf{D}'(i))^{-1} e(i) \mathbf{Y}^*_N(i), \tag{62}$$

where * denotes the complex conjugate operator, $\mathbf{X}_N(i)$ is the input signal vector, $\mathbf{Y}_N(i) = [y_N(0,i), y_N(1,i), \ldots, y_N(N-1,i)]^T$ is the CS-SCHT domain coefficients. $\mathbf{W}'_N(i) = [w'_N(0,i), w'_N(1,i), \ldots, w'_N(N-1,i)]^T$ is the adaptive weight vector. $z(i), d'(i), e(i)$ are the filter output signal, the desired signal, the error signal, respectively. μ is a positive step-size and $\mathbf{D}'(i)$ is a diagonal matrix of the estimated input powers which is given by

$$\begin{aligned} \mathbf{D}'(i) &= \mathrm{diag}\{\sigma^2(k,i)\}, \quad k = 0, \ldots, N-1, \\ \sigma^2(k,i) &= \beta\sigma^2(k,i-1) + (1-\beta)|y_N(k,i)|^2,\ 0 < \beta < 1. \end{aligned} \tag{63}$$

In the following, we apply the aforementioned transform domain LMS adaptive filters for both complex and real input. Note that the fast algorithms have been implemented using "C" programming language. The comparison results of execution time are carried out on a PC machine, which has an AMD single core CPU with speed of 3200 MHz and 4096 MB RAM. The run time of these algorithms have been calculated using MinGW GCC complier version 3.4.5.

A. Channel Equalization (Complex Input)

In this example, a quadrature phase shift keying (QPSK) signal of length 1024 is transmitted over the additive white Gaussian noise channel (AWGN channel). The channel introduces intersymbol interference using a finite impulse response type model. The transfer function of the channel $H(z)$ can be expressed as

$$H(z) = \frac{-0.7e^{j\frac{\pi}{4}} + e^{j\frac{\pi}{4}} z^{-1}}{1 - 0.7z^{-1}}. \tag{64}$$

At the output of the channel, a white Gaussian noise sequence with variance $\sigma^2 = 0.1$ is added. The input signal, which is the sum of the channel output and the noise sequence, is processed by the 32-tap equalizer (filter). The parameters are set as $\mu = 0.3$ and $\beta = 0.2$. Fig. 3 shows the received signal scatter plot and the equalized signal scatter plot by sliding CS-SCHT based sequency domain equalizer (filter). Fig. 4 illustrates the learning curves for sliding FFT/ISCHT/CS-SCHT based adaptive equalizer (filter) using the aforementioned QPSK signal. Table VI shows the execution time of the sliding transforms of the corresponding adaptive equalizer (filters). It can be seen

TABLE V

COMPARISON RESULTS OF THE PROPOSED ALGORITHM OF THE REAL INPUT DATA WITH THE BLOCK-BASED ONE [22], THE SLIDING ISCHT [37], THE SLIDING WHT [29], [31]–[33], THE SLIDING FFT [25], [26], RECIPROCAL-ORTHOGONAL WHT [13] AND DHT [14] FOR $N = 2^n$, $n \geq 4$. "*MULS*" REPRESENTS REAL MULTIPLICATIONS, "$Muls(j)$" MEANS MULTIPLICATION WITH j, "*ADDS*" MEANS REAL ADDITIONS. "Me" DENOTES MEMORY (WORDS)

Real input			4	8	16	32	$N = 2^n$, $n \geq 4$
Proposed algorithm		*Muls(j)*	1	3	7	13	$N/3+2\log_4 N-7/3$, $N=2^n$, $n=4,6,\ldots$ $N/3+2\log_4(N/2)-5/3$, $N=2^n$, $n=5,7,\ldots$
		Adds	4	10	30	70	$25N/9+(\log_4 N-1)(\log_4 N-50/3)/2-64/9$, $N=2^n$, $n=4,6,\ldots$ $25N/9+(\log_4(N/2)-1)(\log_4(N/2)-46/3)/2-110/9$, $N=2^n$, $n=5,7,\ldots$
		Me	5	16	28	60	$N/4+\max\{3N/2,\ Me^{rCS}_{N/4}+(N/2)\log_2 N-11N/8\}$
Block CS-SCHT algorithm [22]		*Muls(j)*	1	3	7	15	$N/2-1$
		Adds	7	21	57	145	$N\log_2 N-N/2+1$
		Me	4	8	16	32	N
Sliding ISCHT [37]		*Muls(j)*	2	5	13	26	$5N/6+\log_4 N-7/3$, $N=2^n$, $n=4,6,\ldots$ $5N/6+\log_4(N/2)-8/3$, $N=2^n$, $n=5,7,\ldots$
		Adds	5	15	38	90	$4N+3\sqrt{N}-23-(6\log_2 N-9)(\log_4 N-1)$, $N=2^n$, $n=4,6,\ldots$ $4N+3\sqrt{N/2}-29-(6\log_2 N-9)(\log_4(N/2)-1)$, $N=2^n$, $n=5,7,\ldots$
		Me	5	18	48	100	$N/4+\max\{3N-4,\ Me^{rISCHT}_{N/4}+(N/2)\log_2 N-15N/8\}$
Sliding WHT	Radix-2 DIT [29] Radix-2 DIS [31]	*Adds*	6	14	30	62	$2N-2$
		Me	6	20	56	144	$N\log_2 N-N/2$
	GCK [32]	*Adds*	8	16	32	64	$2N$
		Me	8	16	32	64	$2N$
	Radix-4 DIS [33]	*Adds*	5	11	22	44	$4N/3+\log_4 N-4/3$, $N=2^n$, $n=4,6,\ldots$ $4N/3+\log_4(N/2)-2/3$, $N=2^n$, $n=5,7\ldots$
		Me	5	12	24	48	$3N/2-1$, $N=4$; $3N/2$, $N\geq 8$
Sliding FFT [25, 26]		*Muls*	0	4	16	44	$2N-4\log_2 N$
		Muls(j)	1	2	3	4	$\log_2 N-1$
		Adds	4	12	32	76	$3N-4\log_2 N$
		Me	5	20	58	150	$N\log_2 N-N/4-2$
Reciprocal-orthogonal WHT (RWHT) [13]		*Muls*	6	12	24	48	$3N/2$
		Adds	8	24	64	160	$N\log_2 N$
		Me	10	20	40	80	$5N/2$
Reciprocal-orthogonal DHT (RDHT) [14]		*Muls*	0	2	8	34	$(13/18)N\log_2 N-(169/54)N+10\log_2 N-(4/9)(-1)^{\log_2 N}\log_2 N+(98/27)(-1)^{\log_2 N}-30$
		Shifts	0	0	0	2	$(1/36)N\log_2 N-(11/54)N+2\log_2 N+(4/9)(-1)^{\log_2 N}\log_2 N-(62/27)(-1)^{\log_2 N}-6$
		Adds	8	22	62	176	$(55/36)N\log_2 N-(161/54)N+10\log_2 N+(28/9)(-1)^{\log_2 N}\log_2 N-(446/27)(-1)^{\log_2 N}-24$
		Me	4	9	24	48	$3N/2$

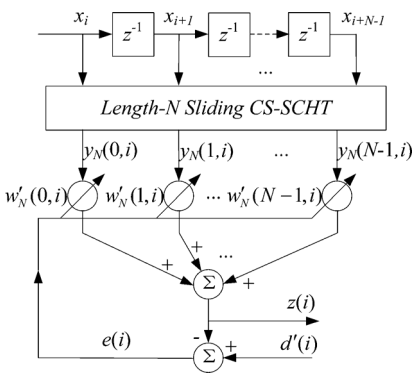


Fig. 2. Block diagram of CS-SCHT domain adaptive filtering.

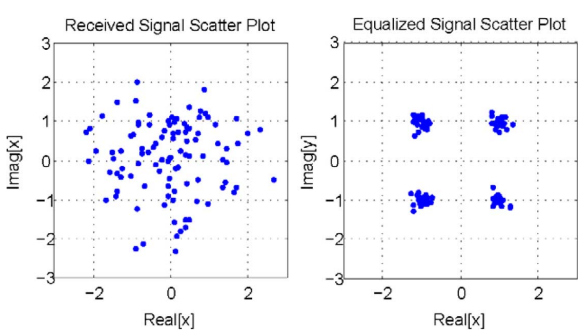


Fig. 3. Scatter plot of received signal and equalized signal.

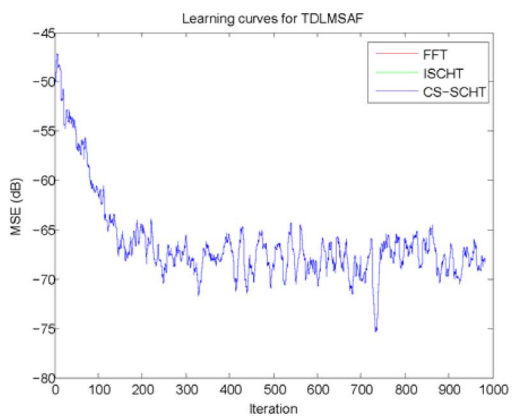


Fig. 4. Learning curves for sliding FFT/ISCHT/CS-SCHT based adaptive equalizer (filter) using a QPSK signal.

from Fig. 4 that the learning curves for TDLMSAF using the sliding CS-SCHT algorithm, sliding ISCHT algorithm, and sliding FFT algorithm are exactly the same. The proposed *Scheme 1* of sliding CS-SCHT algorithm (1.178 ms) allows us to save 40.0% in terms of computational time compared to block CS-SCHT one (1.962 ms), 28.8% compared to sliding FFT one (1.515 ms), and 79.7% compared to block FFT one (5.791 ms) in the process of sliding transformations. However, the proposed sliding CS-SCHT algorithm is 9.3% slower than sliding ISCHT one (1.069 ms).

TABLE VI
COMPARISON OF EXECUTION TIME FOR SLIDING TRANSFORMS IN QPSK SIGNAL ADAPTIVE EQUALIZATION

	Sliding CS-SCHT		Block CS-SCHT	Sliding FFT	Block FFT	Sliding ISCHT
	Scheme 1	Scheme 2				
Execution time (ms)	1.178	1.195	1.962	1.515	5.791	1.069

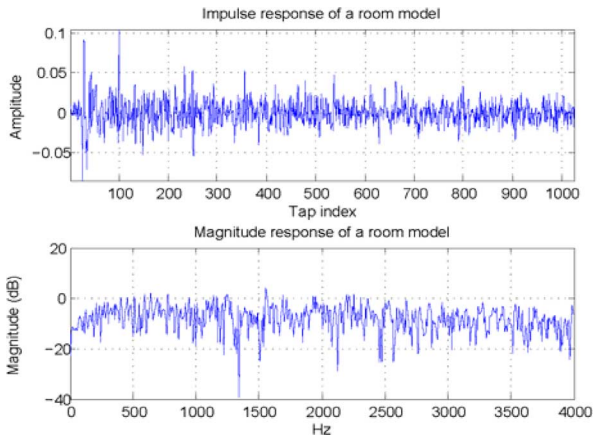


Fig. 5. Impulse and frequency response of a room model.

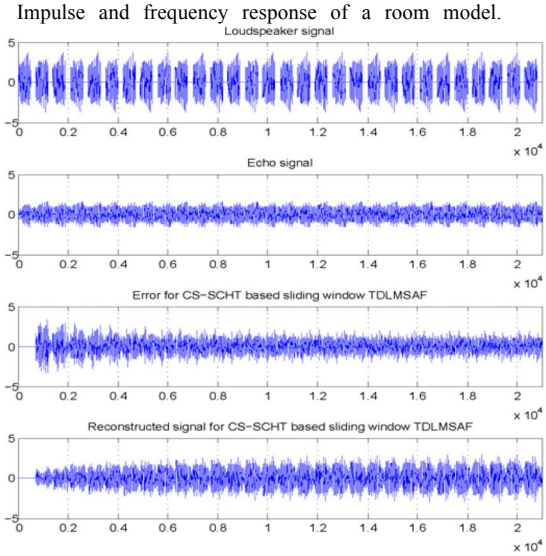


Fig. 6. The top row shows the loudspeaker signal, the second row shows the corresponding echo and the last two rows show the resulting error and reconstructed signal for CS-SCHT based sliding window TDLMSAF.

Note that the QPSK signal is a complex one that is employed in many wireless network standards, such as IEEE 802.11a [49] in which the DFT-based complex adaptive equalizer [50] is used for multicarrier demodulation. In this situation, we can simply substitute the DFT module by that of CS-SCHT with little change. However, if we want to use the WHT as the complex adaptive equalizer, similar to DCT-based system reported in [51], it would be necessary to construct an intermediate complex transform by using two WHTs. This may lead to higher computational complexity and larger change to conventional DFT-based system.

B. Acoustic Echo Cancellation (Real Input)

According to the Computer Project VI (acoustic echo cancellation) in [48], we use a synthetic signal of 1400 samples that emulates the properties of speech. Concatenating 15 such blocks to form a loudspeaker signal and feed it into the echo path. Fig. 5 illustrates the measured impulse and frequency

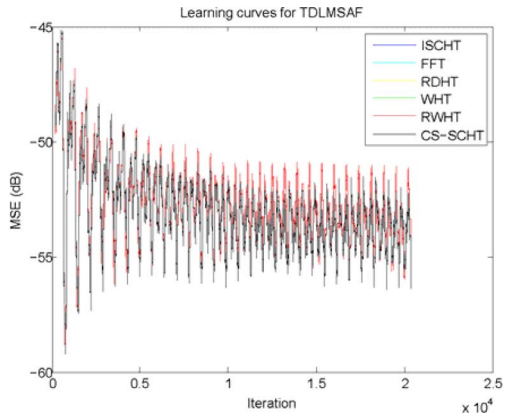


Fig. 7. Learning curves for sliding transforms based TDLMSAF for acoustic echo cancellation. RWHT means reciprocal-orthogonal WHT. RDHT means reciprocal-orthogonal DHT.

response sequence of an echo path in a room, which contains 1024 samples. In this example, we process the acoustic echo canceller with 512 taps TDLMSAF with $\mu = 0.15$ and $\beta = 0.45$. We use the echo as an input signal and the loudspeaker signal as the desired signal to the adaptive filter. Fig. 6 shows the loudspeaker signal, the echo signal, the error signal and filtered signal after CS-SCHT based sliding window TDLMSAF. Because the 512-tap sliding algorithms use the first 512 tap signals as the first input block, so, the first 511 tap indices of error and filtered signals are zeros. Fig. 7 illustrates the learning curves for FFT, RDHT, WHT, RWHT, ISCHT, CS-SCHT based sliding window TDLMSAF for aforementioned echo cancellation scheme. Considering the $3N/2$ independent parameters of the RWHT introduced in [13], we simply set all the N parameters in $\mathbf{S}_1$ to 0.75 and the first $N/2$ parameters in $\mathbf{S}_{-1}$ to 1.25. It can be seen from the Fig. 7 that the learning curves for TDLMSAF using the sliding CS-SCHT, sliding ISCHT, sliding FFT, sliding DHT, sliding WHT are exactly the same and are somewhat better than sliding RWHT. Table VII shows the execution time of the sliding transforms of the corresponding TDLMSAF. The proposed *Scheme 1* of sliding CS-SCHT (0.901 s) saves 34.6% compared to block CS-SCHT (1.379 s), 42.1% compared to sliding FFT (1.557 s), 66.1% compared to RDFT (2.661 s), 11.5% compared to sliding ISCHT (1.018 s), -14.3% compared to sliding WHT (0.772 s), 25.0% compared to RWHT (1.202 s) and 68.2% compared to RDHT (2.831 s) in terms of the execution time.

From the two application examples, we can see that, compared to other sliding transforms, the proposed sliding CS-SCHT seems to be more appropriate for the adaptive filtering system requiring low computational complexity when dealing with both complex and real signals. Compared to real transforms, the sliding CS-SCHT is more suitable for applications where the phase information is needed, for example, PSI measure [39], [40], phase based image retrieval [41]. Further research is still in progress on these applications.

TABLE VII
COMPARISON OF EXECUTION TIME FOR SLIDING TRANSFORMS IN ACOUSTIC ECHO CANCELLATION

	Sliding CS-SCHT		Block CS-SCHT	Sliding FFT	Reciprocal -orthogonal DFT	Sliding ISCHT	Sliding WHT	Reciprocal -orthogonal WHT	Reciprocal -orthogonal DHT
	Scheme 1	Scheme 2							
Execution time (s)	0.901	0.921	1.379	1.557	2.661	1.018	0.772	1.202	2.831

V. CONCLUSION

In this paper, we have presented a fast algorithm for computing the sliding CS-SCHT. The arithmetic complexity order of the proposed algorithm is N, a factor of $\log_2 N$ improvement is made over the block-based algorithm for the length-N CS-SCHT. The proposed algorithm is also more efficient than the sliding FFT algorithm, but less efficient than the sliding WHT algorithms. Compared to the recently proposed sliding ISCHT algorithm, the proposed algorithm is more efficient for real input but less efficient for complex input. The application of the sliding CS-SCHT in TDAF to complex signal channel equalization and real speech signal acoustic echo cancellation has also been investigated.

APPENDIX A
DERIVATION OF (21)

To demonstrate the equivalence between (21) and (20), it suffices to verify that the following two relationships are true:

$$\mathbf{P}_N = \mathbf{R}_N \begin{bmatrix} \mathbf{R}_{N/2} & \\ & \mathbf{R}_{N/2} \end{bmatrix} = \mathbf{R}_N \mathbf{B}_N, \quad \text{(A1)}$$

$$\mathbf{R}_N \mathbf{C}'_N = \mathbf{Q}_N \mathbf{P}_N^T \mathbf{W}_N, \quad \text{(A2)}$$

where

$$\mathbf{B}_N = \begin{bmatrix} \mathbf{R}_{N/2} & \\ & \mathbf{R}_{N/2} \end{bmatrix}. \quad \text{(A3)}$$

Proof of (A1): Let $i = (i_{n-1}, i_{n-2}, \ldots, i_1, i_0)$ and $l = (l_{n-1}, l_{n-2}, \ldots, l_1, l_0)$, $n = \log_2 N$, be respectively the binary representation of the two integers i and l, let $\mathbf{R}_N = (R_{il}), \mathbf{B}_N = (B_{il})$ and $\mathbf{B}'_N = (B'_{il})$ be the product of $\mathbf{R}_N$ and $\mathbf{B}_N$, by the definition, we have

$$R_{il} = \begin{cases} 1, & \text{if } l_k = i_{n-1-k}, \quad k = 0, 1, \ldots, n-1, \\ 0, & \text{otherwise.} \end{cases} \quad \text{(A4)}$$

$$B_{il} = \begin{cases} 1, & \text{if } l_k = i_{(n-2-k) \bmod n}, \quad k = 0, 1, \ldots, n-1, \\ 0, & \text{otherwise.} \end{cases} \quad \text{(A5)}$$

Using (A4), we have

$$B'_{il} = \sum_{m=0}^{N-1} R_{im} B_{ml} = B_{i'l}, \quad \text{(A6)}$$

where i' is the integer corresponding to the bit reversed of i.

Using (A5), (A6) becomes (A7) at the bottom of the page. (A7) is equivalent to

$$B'_{il} = \begin{cases} 1, & \text{if } l_k = i_{(k+1) \bmod n}, \quad k = 0, 1, \ldots, n-1, \\ 0, & \text{otherwise.} \end{cases} \quad \text{(A8)}$$

which is just the (i, l)th element of the matrix $\mathbf{P}_N$. □

Proof of (A2): Using (19), it can be easily seen that (A2) is equivalent to

$$\mathbf{C}'_N = \mathbf{R}_N \mathbf{Q}_N \mathbf{P}_N^T \mathbf{W}_N = \mathbf{R}_N \mathbf{Q}'_N \mathbf{W}_N = \mathbf{R}'_N \mathbf{W}_N, \quad \text{(A9)}$$

where

$$\mathbf{Q}'_N = \mathbf{Q}_N \mathbf{P}_N^T, \quad \text{and} \quad \mathbf{R}'_N = \mathbf{R}_N \mathbf{Q}'_N. \quad \text{(A10)}$$

Let $\mathbf{P}_N^T = (p_{il}), \mathbf{Q}_N = (Q_{il}), \mathbf{Q}'_N = (Q'_{il})$ and $\mathbf{R}'_N = (R'_{il})$. From (A8), we have

$$p_{il} = \begin{cases} 1, & \text{if } i_k = l_{(k+1) \bmod n}, \quad k = 0, 1, \ldots, n-1, \\ 0, & \text{otherwise.} \end{cases} \quad \text{(A11)}$$

From (22), we have

$$Q_{il} = \begin{cases} 1, & \text{if } i = l \quad \text{and} \quad 0 \le i \le N/2 - 1, \\ 1, & \text{if } l = 3N/2 - 1 - i \text{ and } N/2 \le i \le N - 1, \\ 0, & \text{otherwise.} \end{cases} \quad \text{(A12)}$$

Using (A12), we have

$$Q'_{il} = \left(\mathbf{Q}_N \mathbf{P}_N^T\right)_{il} = \sum_{m=0}^{N-1} Q_{im} p_{ml} = \begin{cases} p_{il}, & \text{if } 0 \le i \le N/2 - 1, \\ p_{3N/2-1-i,l}, & \text{if } N/2 \le i \le N - 1. \end{cases} \quad \text{(A13)}$$

The elements of the matrix $\mathbf{R}'_N$ are computed as

$$R'_{il} = \sum_{m=0}^{N-1} R_{im} Q'_{ml}. \quad \text{(A14)}$$

To determine the value of R'_{il}, two cases are distinguished. Let $i' = (i'_{n-1}, i'_{n-2}, \ldots, i'_1, i'_0)$ be the integer corresponding to the bit reversed of $i = (i_{n-1}, i_{n-2}, \ldots, i_1, i_0)$.
Case 1. i is even $(i_0 = 0)$

$$B'_{il} = B_{i'l} = \begin{cases} 1, & \text{if } l_k = i'_{(n-2-k) \bmod n} = i_{n-1-[(n-2-k) \bmod n]}, \quad k = 0, 1, \ldots, n-1, \\ 0, & \text{otherwise.} \end{cases} \quad \text{(A7)}$$

In this case, we have $i'_{n-1}=0$, so that $i' \le N/2-1$. Using (A4), (A13) and (A11), (A14) becomes

$$R'_{il}=Q'_{i'l}=p_{i'l} = \begin{cases} 1, & \text{if } i'_k = l_{(k+1) \bmod n}, \quad k=0,1,\ldots,n-1, \\ 0, & \text{otherwise.} \end{cases} \tag{A15}$$

Equation (A15) shows that R'_{il} takes no-zero value only if the following relationship holds

$$l_{n-1}=i'_{n-2}=i_1, l_{n-2}=i'_{n-3}=i_2,\ldots, \\ l_2=i'_1=i_{n-2}, l_1=i'_0=i_{n-1}, l_0=i'_{n-1}=0, \tag{A16}$$

so that

$$l=\sum_{j=0}^{n-1} l_j 2^j = \sum_{j=0}^{n-2} i_{n-1-j} 2^{j+1} = 2i'. \tag{A17}$$

Therefore, for even value of i

$$R'_{il} = \begin{cases} 1, & \text{if } l=2i', \\ 0, & \text{otherwise.} \end{cases} \tag{A18}$$

Case 2. i is odd ($i_0=1$)

In this case, we have

$$i'=N/2+\sum_{k=0}^{n-2} i_{k+1} 2^{n-2-k} = N/2+i''. \tag{A19}$$

(A14) becomes

$$R'_{il}=Q'_{i'l}=p_{3N/2-1-i',l}=p_{N-1-i'',l} = \begin{cases} 1, & \text{if } (N-1-i'')_k = l_{(k+1) \bmod n}, k=0,1,\ldots,n-1, \\ 0, & \text{otherwise.} \end{cases} \tag{A20}$$

Since

$$N-1-i''=\sum_{k=0}^{n-1} 2^k - \sum_{k=0}^{n-2} i_{k+1} 2^{n-2-k} \\ = 2^{n-1}+\sum_{k=0}^{n-2} (1-i_{n-1-k}) 2^k, \tag{A21}$$

thus

$$l_0=1, \quad l_1=1-i_{n-1}, \\ l_2=1-i_{n-2},\ldots,l_{n-2}=1-i_2, \quad l_{n-1}=1-i_1, \tag{A22}$$

so that

$$l=\sum_{j=0}^{n-1} l_j 2^j = \sum_{j=1}^{n-1} (1-i_{n-j}) 2^j + 1. \tag{A23}$$

From (A21), we also have

$$2(N-1-i'')=N+\sum_{k=1}^{n-1} (1-i_{n-k}) 2^k. \tag{A24}$$

Combining (A19) with (A24), we have

$$\sum_{k=1}^{n-1} (1-i_{n-k}) 2^k = 2N-2-2i'. \tag{A25}$$

Substituting (A25) into (A23), we have

$$l=2N-1-2i'. \tag{A26}$$

Therefore, for odd value of i

$$R'_{il} = \begin{cases} 1, & \text{if } l=2N-1-2i', \\ 0, & \text{otherwise.} \end{cases} \tag{A27}$$

Combination of (A18) and (A27) leads to

$$R'_{il} = \begin{cases} 1, & \text{if } l=2i', i \text{ is even}, \\ 1, & \text{if } l=2N-1-2i', i \text{ is odd}, \\ 0, & \text{otherwise.} \end{cases} \tag{A28}$$

So, we have constructed the matrix $\mathbf{R}'_N$. Using the definition of the two matrices $\mathbf{C}'_N$ and $\mathbf{W}_N$, it can be easily verified that (A9) is true. □

Appendix B
Derivation of (39)

To prove (39), we need the following lemma:

Lemma 1: Let $h_N(k,l)$ be the (kth, lth) element of the matrix $\mathbf{H}_N$, then we have

$$\begin{cases} h_N(k,l+N/4)=j^k h_N(k,l), & \text{for } l=0,1,\ldots,3N/4-1, \\ h_N(k,l+3N/4)=(-j)^k h_N(k,l), & \text{for } l=0,1,\ldots,N/4-1. \end{cases}$$

Proof: To prove the lemma, we need the following relationship

$$(-1)^{g_{n-1}}(-j)^{f_{n-1}}=(-1)^k, \quad (-1)^{g_{n-2}}(-j)^{f_{n-2}}=j^k, \tag{B1}$$

where $\mathbf{G}=(g_{n-1},\ldots,g_1,g_0)$ and $\mathbf{F}=(f_{n-1},\ldots,f_1,f_0)$, g_r is a binary gray code of the bit reversal of k_r and f_r is the r th bit of the binary bits of the highest power of 2 in $c(k)/2$ where $c(k)$ is the decimal number obtained through a bit-reversed conversion of the decimal $k=(k_{n-1},\ldots,k_1,k_0)$. By the definition, we have

$$f_s = \begin{cases} 0, & \text{if } s=n-1, \\ 1, & \text{if } s=n-2-t, \\ 0, & \text{otherwise.} \end{cases} \tag{B2}$$

and

$$g_{n-1}=k_0, \quad g_{n-2}=(k_1+k_0) \bmod 2, \\ g_{n-r}=(k_{r-1}+k_{r-2}) \bmod 2, \quad \text{for } 2 \le r \le n, \tag{B3}$$

where t appeared in (B2) is the minimal value between 0 and $n-2$ such that $k_t=1$.

To prove (B1), two cases are distinguished.

a) k is even, that is, $k_0=0$.

From (B2) and (B3), we have

$$f_{n-1}=g_{n-1}=0, \\ f_{n-2}=0, \quad g_{n-2}=k_1 \bmod 2 = k_1. \tag{B4}$$

So

$$(-1)^{g_{n-2}}(-j)^{f_{n-2}} = (-1)^{k_1}, \quad (-1)^{g_{n-1}}(-j)^{f_{n-1}} = 1, \tag{B5}$$

and

$$\begin{aligned} j^k &= j^{\sum_{r=1}^{n-1} k_r 2^r} \\ &= j^{2k_1} = (-1)^{k_1},\ (-1)^k = 1. \end{aligned} \tag{B6}$$

Combination of (B5) and (B6) leads to (B1).

b) k is odd, i.e., $k_0 = 1$.

In this case, we have

$$\begin{aligned} &f_{n-1} = 0, \quad g_{n-1} = 1, \\ &f_{n-2} = 1, \quad g_{n-2} = (k_1 + 1) \bmod 2. \end{aligned} \tag{B7}$$

So

$$\begin{aligned} (-1)^{g_{n-1}}(-j)^{f_{n-1}} &= -1, \\ (-1)^{g_{n-2}}(-j)^{f_{n-2}} &= (-1)^{((k_1+1) \bmod 2)+1} j = (-1)^{k_1} j, \end{aligned} \tag{B8}$$

and

$$\begin{aligned} (-1)^k = -1, \quad j^k &= j^{\sum_{r=0}^{n-1} k_r 2^r} \\ &= j^{2k_1+1} = (-1)^{k_1} j. \end{aligned} \tag{B9}$$

It can be deduced from (B8) and (B9) that (B1) is also true for odd value of k.

We are now ready to prove the Lemma. By the definition, we have

$$\begin{aligned} &h_N(k, l + N/4) \\ &\quad = (-1)^{\sum_{r=0}^{n-1} g_r (l_r + (N/4)_r)} (-j)^{\sum_{r=0}^{n-1} f_r (l_r + (N/4)_r)} \\ &\quad = (-1)^{\sum_{r=0}^{n-1} g_r (2^{n-2})_r} (-j)^{\sum_{r=0}^{n-1} f_r (2^{n-2})_r} h_N(k, l) \\ &\quad = (-1)^{g_{n-2}} (-j)^{f_{n-2}} h_N(k, l) \\ &\quad = j^k h_N(k, l). \end{aligned} \tag{B10}$$

For $h_N(k, l + 3N/4)$, we have

$$\begin{aligned} &h_N(k, l + 3N/4) \\ &\quad = (-1)^{\sum_{r=0}^{n-1} g_r (l_r + (3N/4)_r)} (-j)^{\sum_{r=0}^{n-1} f_r (l_r + (3N/4)_r)} \\ &\quad = (-1)^{\sum_{r=0}^{n-1} g_r (2^{n-1} + 2^{n-2})_r} \\ &\qquad \times (-j)^{\sum_{r=0}^{n-1} f_r (2^{n-1} + 2^{n-2})_r} h_N(k, l) \\ &\quad = (-1)^{g_{n-1} + g_{n-2}} (-j)^{f_{n-1} + f_{n-2}} h_N(k, l) \\ &\quad = (-j)^k h_N(k, l). \end{aligned} \tag{B11}$$

The proof of Lemma has been completed. □

Based on the above lemma, we provide the derivation of (39) in the following.

(11) can be written as

$$\begin{aligned} &y_N(k, i) \\ &\quad = \mathbf{H}_N^o(k) \mathbf{X}_N(i) = \sum_{l=0}^{N-1} h_N(k, l) x_{i+l} \\ &\quad = \sum_{l=0}^{N/4-1} h_N(k, l) x_{i+l} + \sum_{l=0}^{3N/4-1} h_N(k, l + N/4) x_{i+N/4+l}. \end{aligned} \tag{B12}$$

Similarly,

$$\begin{aligned} &y_N(k, i+N/4) \\ &\quad = \mathbf{H}_N^o(k) \mathbf{X}_N(i+N/4) = \sum_{l=0}^{N-1} h_N(k, l) x_{i+N/4+l} \\ &\quad = \sum_{l=0}^{3N/4-1} h_N(k, l) x_{i+N/4+l} + \sum_{l=0}^{N/4-1} h_N(k, l+3N/4) x_{i+N+l}. \end{aligned} \tag{B13}$$

Using Lemma 1, (B12) and (B13) become

$$\begin{aligned} y_N(k, i) = &\sum_{l=0}^{N/4-1} h_N(k, l) x_{i+l} \\ &+ j^k \sum_{l=0}^{3N/4-1} h_N(k, l) x_{i+N/4+l}, \end{aligned} \tag{B14}$$

$$\begin{aligned} y_N(k, i + N/4) = &\sum_{l=0}^{3N/4-1} h_N(k, l) x_{i+N/4+l} \\ &+ (-j)^k \sum_{l=0}^{N/4-1} h_N(k, l) x_{i+N+l}. \end{aligned} \tag{B15}$$

(B14) can be rewritten as

$$\begin{aligned} \sum_{l=0}^{3N/4-1} h_N(k, l) x_{i+N/4+l} &= (-j)^k y_N(k, i) \\ &- (-j)^k \sum_{l=0}^{N/4-1} h_N(k, l) x_{i+l}. \end{aligned} \tag{B16}$$

Substituting (B16) into (B15), we have

$$\begin{aligned} &y_N(k, i + N/4) \\ &\quad = (-j)^k \left[y_N(k, i) - \sum_{l=0}^{N/4-1} h_N(k, l) d_N(i + l) \right] \\ &\quad = (-j)^k [y_N(k, i) - t_N(k, i)], \end{aligned} \tag{B17}$$

where

$$d_N(i + l) = x_{i+l} - x_{i+N+l}, l = 0, 1, \ldots, N/4 - 1, \tag{B18}$$

$$\begin{aligned} t_N(k, i) = &[h_N(k, 0), h_N(k, 1), \ldots, h_N(k, N/4 - 1)] \\ &\times [d_N(i), d_N(i + 1), \ldots, d_N(i + N/4 - 1)]^T, \\ &k = 0, 1, \ldots, N - 1. \end{aligned} \tag{B19}$$

The above equation can be expressed in a matrix representation as

$$\begin{bmatrix} t_N(0, i) \\ t_N(1, i) \\ \cdots \\ t_N(N - 1, i) \end{bmatrix} = \mathbf{H}_N^{1/4} \begin{bmatrix} d_N(i) \\ d_N(i + 1) \\ \cdots \\ d_N(i + N/4 - 1) \end{bmatrix}. \tag{B20}$$

Since

$$\mathbf{H}_N^{1/4} = \mathbf{H}_N \begin{bmatrix} \mathbf{I}_{N/4} \\ \mathbf{0}_{(3N/4) \times (N/4)} \end{bmatrix}, \tag{B21}$$

$$\mathbf{H}_N^{1/2} = \mathbf{H}_N \begin{bmatrix} \mathbf{I}_{N/2} \\ \mathbf{0}_{N/2} \end{bmatrix}, \tag{B22}$$

Substituting (27) into (B21), we have

$$\begin{aligned}
&\mathbf{H}_N^{1/4} \\
&= \mathbf{P}_N \begin{bmatrix} \mathbf{H}_{N/2} & \\ & \mathbf{Q}_{N/2} \begin{bmatrix} \mathbf{W}_{N/4} & \mathbf{D}_{N/4}\mathbf{W}_{N/4} \\ \mathbf{W}_{N/4} & -\mathbf{D}_{N/4}\mathbf{W}_{N/4} \end{bmatrix} \mathbf{S}_{N/2} \end{bmatrix} \begin{bmatrix} \mathbf{I}_{N/4} \\ \mathbf{0}_{N/4} \\ \mathbf{I}_{N/4} \\ \mathbf{0}_{N/4} \end{bmatrix} \\
&= \mathbf{P}_N \begin{bmatrix} \mathbf{H}_{N/2}^{1/2} \\ \begin{bmatrix} \mathbf{W}_{N/4} \\ \mathbf{J}_{N/4}\mathbf{W}_{N/4} \end{bmatrix} \end{bmatrix}. \qquad \text{(B23)}
\end{aligned}$$

Substituting (21) into (B22), we have

$$\mathbf{H}_N^{1/2} = \mathbf{P}_N \begin{bmatrix} \mathbf{H}_{N/2} \\ \mathbf{Q}_{N/2}\mathbf{P}_{N/2}^T\mathbf{W}_{N/2}\mathbf{S}_{N/2} \end{bmatrix}. \qquad \text{(B24)}$$

Substituting (B24) into (B23), we obtain (41). The proof of (39) has been completed. □

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, and Dr. Gouenou Coatrieux for careful corrections on the manuscript. The authors are also grateful to the associate editor and 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: Springer, 1975.

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

[3] S. G. Johnson and M. Frigo, "A modified split-radix FFT with fewer arithmetic operations," *IEEE Trans. Signal Process.*, vol. 55, no. 1, pp. 111–119, Jan. 2007.

[4] Y. Voronenko and M. Puschel, "Algebraic signal processing theory: Cooley-Tukey type algorithms for real DFTs," *IEEE Trans. Signal Process.*, vol. 57, no. 1, pp. 205–222, Jan. 2009.

[5] L. Tao and H. K. Kwan, "Novel DCT-based real-valued discrete Gabor transform and its fast algorithms," *IEEE Trans. Signal Process.*, vol. 57, no. 6, pp. 2151–2164, Jun. 2009.

[6] M. Garrido, K. K. Parhi, and J. Grajal, "A pipelined FFT architecture for real-valued signals," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 56, no. 12, pp. 2634–2643, Dec. 2009.

[7] O. Nibouche, S. Boussakta, and M. Darnell, "Pipeline architectures for radix-2 new Mersenne number transform," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 56, no. 8, pp. 1668–1680, Aug. 2009.

[8] J. S. Wu, H. Z. Shu, L. Senhadji, and L. M. Luo, "Mixed-radix algorithm for the computation of forward and inverse MDCTs," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 56, no. 4, pp. 784–794, Apr. 2009.

[9] Y. A. Geadah and M. J. G. 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.

[10] 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, 1989.

[11] 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.

[12] B. J. Falkowski and S. X. Yan, "Ternary Walsh transform and its operations for completely and incompletely specified boolean functions," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 54, no. 8, pp. 1750–1764, 2007.

[13] S. Bouguezel, M. O. Ahmad, and M. N. S. Swamy, "A new class of reciprocal-orthogonal parametric transforms," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 56, no. 4, pp. 795–805, Apr. 2009.

[14] S. Bouguezel, M. O. Ahmad, and M. N. S. Swamy, "New parametric discrete Fourier and Hartley transforms, and algorithms for fast computation," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 58, no. 3, pp. 562–575, 2011.

[15] 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, 1999.

[16] 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, 2000.

[17] 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.

[18] A. Aung, B. P. Ng, and S. Rahardja, "A robust watermarking scheme using sequency-ordered complex Hadamard transform," *J. Signal Process. Syst.*, vol. 64, pp. 319–333, 2011.

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

[20] 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.

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

[22] A. Aung, B. P. Ng, and S. Rahardja, "Conjugate symmetric sequency-ordered complex Hadamard transform," *IEEE Trans. Signal Process.*, vol. 57, pp. 2582–2593, Jul. 2009.

[23] V. Kober, "Fast algorithms for the computation of sliding discrete sinusoidal transforms," *IEEE Trans. Signal Process.*, vol. 52, pp. 1704–1710, Jun. 2004.

[24] 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.

[25] B. Farhang-Boroujeny and Y. C. Lim, "A comment on the computational complexity of sliding FFT," *IEEE Trans. Circuits Syst.*, vol. 39, pp. 875–876, Dec. 1992.

[26] 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.

[27] E. Jacobsen and R. Lyons, "The sliding DFT," *IEEE Signal Process. Mag.*, vol. 20, pp. 74–80, Mar. 2003.

[28] E. Jacobsen and R. Lyons, "An update to the sliding DFT," *IEEE Signal Process. Mag.*, vol. 21, no. 1, pp. 110–111, Jan. 2004.

[29] 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.

[30] 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.

[31] Y. Hel-Or and H. Hel-Or, "Real time pattern matching using projection kernels," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 27, pp. 1430–1445, Sep. 2005.

[32] G. Ben-Artzi, H. Hel-Or, and Y. Hel-Or, "The gray-code filter kernels," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 29, pp. 382–393, Mar. 2007.

[33] W. Ouyang and W. K. Cham, "Fast algorithm for Walsh Hadamard transform on sliding windows," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 32, pp. 165–171, Jan. 2010.

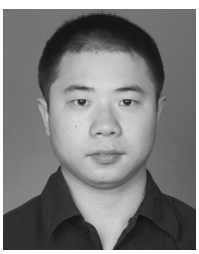
[34] Y. Moshe and H. Hel-Or, "Video block motion estimation based on gray-code kernels," *IEEE Trans. Image Process.*, vol. 18, pp. 2243–2254, Oct. 2009.

[35] W. Ouyang, R. Zhang, and W. K. Cham, "Fast pattern matching using orthogonal Haar transform," in *Proc. IEEE CVPR*, San Francisco, CA, Jun. 2010, pp. 3050–3057.

[36] 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.

[37] J. S. Wu, H. Z. 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.

[38] B. Farhang-Boroujeny, *Adaptive Filters: Theory and Applications*. New York: Wiley, 1998.
[39] G. Nolte, A. Ziehe, V. V. Nikulin, A. Schlögl, N. Krämer, T. Brismar, and K. R. Müller, "Robustly estimating the flow direction of information in complex physical systems," *Phys. Rev. Lett.*, vol. 100, pp. 234101:1–234101:4, Jun. 2008.
[40] C. Yang, R. Le Bouquin Jeannès, G. Faucon, and H. Shu, "Extracting information on flow direction in multivariate time series," *IEEE Signal Process. Lett.*, vol. 18, no. 4, pp. 251–254, 2011.
[41] I. Bartolini, P. Ciaccia, and M. Patella, "Warp: Accurate retrieval of shapes using phase of Fourier descriptors and time warping distance," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 27, pp. 142–147, Jan. 2005.
[42] T. Adali and S. Haykin, *Adaptive Signal Processing: Next-Generation Solutions*. Hoboken, NJ: Wiley, 2010.
[43] J. S. Wu, L. Wang, L. Senhadji, and H. Z. Shu, "Sliding conjugate symmetric sequency-ordered complex Hadamard transform: Fast algorithm and applications," in *EUSIPCO*, Aalborg, Denmark, Aug. 2010, pp. 1742–1746.
[44] H. S. Stone, "Parallel processing with the perfect shuffle," *IEEE Trans. Comput.*, vol. C-20, no. 2, pp. 153–161, 1971.
[45] J. Ellis, T. Krahn, and H. Fan, "Computing the cycles in the perfect shuffle permutation," *Inf. Process. Lett.*, vol. 75, no. 5, pp. 217–224, Oct. 2000.
[46] P. Jain, "A simple in-place algorithm for in-shuffle," 2008, Arxiv preprint arXiv:0805.1598.
[47] S. S. Narayan, A. M. Peterson, and M. J. Narashima, "Transform domain LMS algorithm," *IEEE Trans. Acoust., Speech, Signal Process.*, vol. 31, pp. 609–615, Jun. 1983.
[48] A. H. Sayed, *Adaptive Filters*. New York: Wiley, 2008.
[49] *High-Speed Physical Layer in the 5 GHz Band—Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*, IEEE Std. 802.11a, 1999.
[50] P. Tan and N. C. Beaulieu, "A comparison of DCT-based OFDM and DFT-based OFDM in frequency offset and fading channels," *IEEE Trans. Commun.*, vol. 54, no. 11, pp. 2113–2125, Nov. 2006.
[51] S. U. H. Qureshi, "Adaptive equalization," *Proc. IEEE*, vol. 73, no. 9, pp. 1349–1387, Sep. 1985.

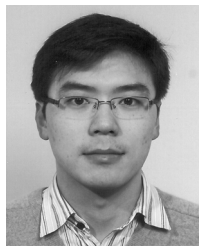


Jiasong Wu (M'09) received the B.S. degree in biomedical engineering from the University of South China, Hengyang, in 2005, and is currently working toward the joint Ph.D. degree with the Laboratory of Image Science and Technology (LIST), Southeast University, Nanjing, China, and Laboratoire Traitement du signal et de l'Image (LTSI), University of Rennes 1, Rennes, France.

His research interest mainly includes fast algorithms of digital signal processing and its applications.

Lu Wang (SM'11) received the B.S. degree in computer science and engineering from the University of Shanghai for Science and Technology, Shanghai, China, in 2008, and the joint M.S. degree from the Southeast University, Nanjing, China, and University of Rennes 1, Rennes, France, in 2011. He is currently working toward the Ph.D. degree with the LTSI, University of Rennes 1, Rennes, France.

His research interest mainly includes fast algorithms of digital signal processing and its applications.



Guanyu Yang received the B.S. and M.S. degrees in biomedical engineering from Southeast University, China, in 2002 and 2004, respectively, and the Ph.D. degree in signal processing and telecommunications from the University of Rennes 1, Rennes, France, in 2008.

He was a Postdoctoral Fellow with the Division of Image Processing, Leiden University Medical Center, Leiden, The Netherlands, from 2009 to 2011. He is now with the LIST of the Department of Computer Science and Engineering of Southeast University, Nanjing, China. His research interest mainly focuses on image analysis and pattern recognition.

Lotfi Senhadji (M'95–SM'99) received the Ph.D. degree from the University of Rennes 1, Rennes, France, in signal processing and telecommunications in 1993.

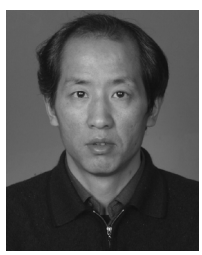
He is a Professor and the Head of the INSERM Research Laboratory (LTSI), Rennes, France. His is also Co-Director of the French-Chinese Laboratory CRIBs "Centre de Recherche en Information Biomédicale Sino-Français." His main research efforts are focused on nonstationary signal processing with particular emphasis on wavelet transforms.

Dr. Senhadji is a Senior Member of the IEEE EMBS and the IEEE Signal Processing Society.

Limin Luo (M'90–SM'97) received the Ph.D. degree in signal processing and telecommunications from the University of Rennes 1, Rennes, France, in 1986.

He is a Professor and the Head of the Laboratory of Image Science and Technology, Southeast University, Nanjing. His research interests include image analysis, computer-assisted systems for diagnosis and therapy in medicine, and computer vision.

Dr. Luo is a Senior Member of the IEEE Engineering in Medicine and Biology 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 a Professor of the LIST Laboratory and the Codirector of the CRIBs, Nanjing, China. His recent work concentrates on the image analysis, pattern recognition, and fast algorithms of digital signal processing.