Hardware Implementation of Iterative Method With Adaptive Thresholding for Random Sampling Recovery of Sparse Signals

Mohammad Fardad, Sayed Masoud Sayedi, and Ehsan Yazdian

***Abstract*— Sparse recovery methods find extensive applications in various fields such as image and audio processing, wireless communication, and spectral estimation. In this paper, a hardware architecture of iterative method with adaptive thresholding (IMAT) is presented to recover a sparse signal from its random samples. To demonstrate the effectiveness of IMAT, a comparison is performed between the IMAT algorithm and the compressive sensing recovery method, orthogonal matching pursuit, in terms of complexity and reconstruction quality. In the context of image reconstruction, as a practical case, simulation results show the advantages of IMAT in improving the performance of the signal reconstruction via peak signal-to-noise ratio and structural similarity index metrics. Since IMAT employs discrete transform as a principal operation in each iteration, using fast or fast approximate algorithms allows more efficient implementation. In this paper, two multiplication-free transforms, Walsh–Hadamard transform (WHT) and approximate discrete cosine transform (ADCT), are used to reduce computational complexity of IMAT implementation. The IMAT recovery algorithm is implemented on Virtex6 FPGA using two above transforms, and the results are compared with respect to the hardware resource utilization, power consumption, and recovery time. The results demonstrate that using WHT for IMAT implementation is more efficient than using ADCT in terms of hardware resources. Also, the comparison implies that the performance of the hardware implementation is very close to its floating point simulated counterpart. For a block size of 32 × 32, the IMAT implementation using WHT provides the reconstruction time of 185 μs and the dynamic power of 123 mW.**

***Index Terms*— Discrete transform (DT), hardware reconstruction, iterative method with adaptive thresholding (IMAT), low-complexity architecture, random sampling.**

I. Introduction

IN MANY practical applications in signal processing, signals are sparse in some domain. The sparse signals can be represented with significantly fewer samples, and the conventional Nyquist sampling rate is not necessary for such signals. One sampling approach to relax the Nyquist sampling rate is compressive sensing (CS) [1], [2]. In this method, a linear process is used as a measurement matrix to capture intrinsic information content of a sparse signal. However, the CS approach has certain drawbacks. First, using the measurement matrices, especially in large scale problems, have some limitations including their storage requirements and computational costs. Second, its reconstruction methods becomes complex when the size of input signal is large. To overcome the above shortcomings, in this paper, random sampling has been used. Random sampling scheme, as a special case of CS, takes random samples of the signal entries. In this scheme, a signal is sampled directly on some nonuniform points.

For sparse signal recovery after sampling, different algorithms have been introduced. Convex relaxation techniques [3], greedy pursuit (GP) approaches such as orthogonal matching pursuit (OMP) [4], iterative thresholding schemes such as iterative hard thresholding (IHT) [5], and approximate message passing (AMP) [6] are some of the most popular ones. Convex optimization methods require significant computing power, complicated structures, and high numerical precision. These requirements make convex optimization inappropriate for hardware implementation. GP algorithms are alternative approach to alleviate computational complexity. These algorithms operate in an iterative manner which greedily computes an approximation of the original signal. Several studies have been devoted to the implementation of the greedy algorithms [7]–[14]. However, since the complexity of GP algorithms scales approximately linearly with the signal's sparsity level, these algorithms are well suited for high sparse signals. To recover low sparse signals, such as images or videos, GP algorithms become inefficient in terms of hardware resources and throughput. Furthermore, the complexity burden of GP approaches increases significantly for large size problems. The recent subject is not a major issue for offline processing, however, for real-time applications or for portable devices with limited energy resource, it becomes extremely challenging. Different from GP algorithms, iterative thresholding algorithms typically have an uncomplicated structure with a simpler data-flow control. Therefore, for applications featuring low sparse signals, such algorithms are good options for efficient real-time hardware implementations [15]. In this paper, we present an architecture that implement an iterative method known as the iterative method with adaptive thresholding (IMAT) [16], [17], for sparse signal reconstruction from its random samples. Compared to other iterative thresholding methods like IHT and AMP, IMAT has a simpler thresholding function which makes it a suitable choice for hardware implementation. Using discrete transform (DT) as a principal operation in IMAT, allows for the design of more efficient hardware implementations by employing the fast or fast approxi-

Manuscript received August 10, 2017; revised November 29, 2017; accepted January 4, 2018. Date of publication January 25, 2018; date of current version April 24, 2018. *(Corresponding author: Mohammad Fardad.)*

The authors are with the Department of Electrical and Computer Engineering, Isfahan University of Technology, Isfahan 8415683111, Iran (e-mail: m.fardad@ec.iut.ac.ir; m_sayedi@cc.iut.ac.ir; yazdian@cc.iut.ac.ir).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TVLSI.2018.2791351

1063-8210 © 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

Authorized licensed use limited to: Universidad Federal de Pernambuco. Downloaded on June 01,2025 at 22:41:25 UTC from IEEE Xplore. Restrictions apply.

mate algorithms. Simulation results compare the performance of IMAT with the OMP algorithm, stating that using IMAT recovery after random sampling provides superior performance in dealing with image processing applications.

A. Contribution

In this paper, we present—to the best of our knowledge—the first hardware architecture of the IMAT algorithm for signal reconstruction from random samples. Whereas most of the reconstruction methods are designed to handle 1-D signals, IMAT can be used directly for 2-D signals such as images and videos. DT and its inverse are the main operations of the IMAT algorithm. Here, we propose using the Walsh–Hadamard transform (WHT) and the approximate discrete cosine transform (ADCT) for hardware complexity reduction. In our design, both transforms are multiplication-free and implemented using fast algorithms. The results of using two above transforms for IMAT recovery implementation are compared with each other for image recovery in terms of performance and hardware complexity.

B. Outline of the Paper

The remainder of this paper is organized as follows. The IMAT algorithm, its performance evaluation, and complexity analysis are presented in Section II. Section III describes the hardware architecture design of IMAT. The details of DT matrices and performance analysis are also included in this section. Section IV gives timing analysis. Implementation results and comparison are given in Section V. Finally, the conclusion is provided in Section VI.

II. Overview of IMAT Algorithm, Its Performance Evaluation, and Complexity Analysis

In this section, the IMAT algorithm is introduced and an analysis is performed to show its computational complexity for some DTs.

A. Iterative Method With Adaptive Thresholding (IMAT)

In the IMAT algorithm, nonlinear thresholds are used iteratively to recover a sparse signal from its samples. The random sampled signal is obtained by applying a binary mask to the original signal. The pseudocode for reconstruction of the sampled signal using IMAT is given in Algorithm 1. Here, $I_{\max}$ denotes the maximum number of iterations, $x^{(i)}$ represents the estimate of x in the ith iteration, $\theta^{(i)}$ indicates the iteration-dependent threshold, and $m^{(i)}$ is the thresholding binary mask in the ith iteration. The thresholding function is used to select the elements with the largest contribution. The domain where the signal exhibits sparse characteristics is called sparse domain and the domain where the signal samples are captured is called information domain. The DT and IDT are discrete transform and its inverse, respectively. These two transforms are used for projection of the signal from the information domain to the sparsity domain and vice versa. To initialize, the sampled signal in the information domain (e.g., time domain) is transformed into sparse domain [e.g., discrete cosine transform (DCT) domain]. Next, the signal is passed through an adaptive thresholding filter which does not pass the components below a specific threshold value.

Algorithm 1 IMAT Recovery Algorithm

1. $x^{(0)}$: random sampled signal, S_{mask}: sampling mask
2. for $i = 1 \cdots I_{max}$ do
3. $\quad X^{(i-1)} = DT(x^{(i-1)})$
4. $\quad \theta^{(i)} = \beta e^{-\alpha(i-1)}$
5. $\quad m^{(i)} = \left|X^{(i-1)}\right| > \theta^{(i)}$
6. $\quad x_{\theta}^{(i-1)} = IDT(m^{(i)} \cdot X^{(i-1)})$
7. $\quad x^{(i)} = x^{(0)} + (1 - S_{mask})\, x_{\theta}^{(i-1)}$
8. end for;
9. return $x^{(I_{max})}$

A large value is assigned as an initial threshold value and by increasing the iteration number; the threshold decreases exponentially. The gradually variations of the threshold value leads to retrieve all the coefficients of the signal after a number of iterations. IMAT parameters, α and β, are experimental parameters and should be chosen appropriately considering the type of input signal. After passing through the thresholding filter, the signal is converted back to the information domain via the IDT. After applying the IDT, in locations where the sampling mask entries are "1," all the entries of the information domain signal are replaced by their counterparts from the sampled input signal (line 7 of Algorithm 1). Eventually, the algorithm gives an estimation of the original signal after $I_{\max}$ iterations.

B. Performance Evaluation and Comparison

In this section, the recovery performance of IMAT is compared with the well-known OMP recovery method, in image processing applications. IMAT is used for recovering a signal from random samples, whereas, OMP is utilized for recovering a signal from a linear combination of its samples. Here, the performances of both recovery algorithms are evaluated by getting the same number of samples without any adaptation in the sampling process. As, the computational complexity of OMP considerably increases with the size of signal, image should be divided into smaller size blocks. Dividing the image into small blocks is not required for IMAT due to its simpler structure. However, to have similar conditions, the same procedure is used for IMAT. To evaluate the effect of block size (N) on performance and computational complexity of the algorithm, different block sizes of 8×8, 16×16, ..., 128×128 are examined. First, according to the size of blocks, for IMAT a 2-D random sampling mask is generated with the sampling rate of 0.25 (25% of its entries are "1" and the others are "0"). An identical mask is used for all blocks of a particular image and each block is sampled separately using this mask and then recovered using IMAT. For OMP, sampling is carried out with a random Gaussian matrix with the sampling ratio of 0.25 (the ratio of the number of rows to the number of columns of the measurement matrix). Since OMP is designed for 1-D signals, each image block is vectorized and then sampled by the measurement matrix. Similar to IMAT, an identical measurement matrix is used for all blocks of each image. The sampling and reconstruction procedure is performed for 49 512×512 8-bit grayscale images, obtained from a standard public image bank [18]. All images are normalized to the

Authorized licensed use limited to: Universidad Federal de Pernambuco. Downloaded on June 01,2025 at 22:41:25 UTC from IEEE Xplore. Restrictions apply.

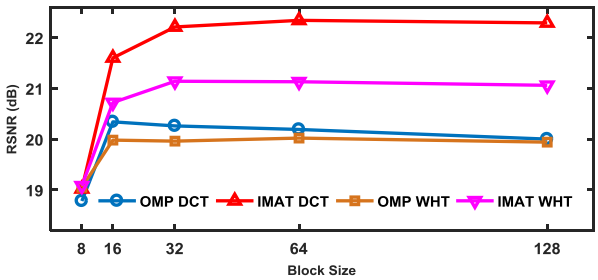


Fig. 1. Average PSNR versus block size for IMAT and OMP with DCT and WHT as sparse domains.

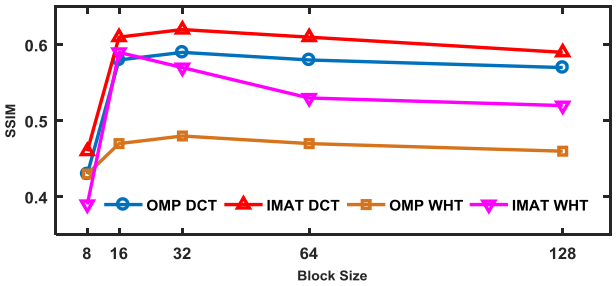


Fig. 2. Average SSIM versus block size for IMAT and OMP with DCT and WHT as sparse domains.

maximum pixel value (255), and the mentioned procedure is repeated 10 times for each image with different random masks for IMAT and random matrices for OMP. DCT and WHT are considered as the sparsity domain since the image signals have sparse representation in these domains. Here, we use exact DCT without any approximation. The IMAT algorithm iterates 30 times and its parameters α and β are considered to be 0.3 and 200, respectively, for all cases. For OMP, the required number of iterations depends on the signal's sparsity order. The sparsity order scales with the block size and its value is increased with increasing the block size. Here, according to the sparsity order of signals in the typical CS applications [7]–[10], we consider that only 3% of image block coefficients in the transform domain are nonzero. Consequently, the sparsity order is assumed to be $0.03 \times N^2$, where N is the block size. Fig. 1 depicts the average peak signal-to-noise ratio (PSNR) of the recovered images versus the block size. The structural similarity (SSIM) index [19] is also calculated for two algorithms. SSIM is a perception-based model that considers image degradation as perceived change in structural information. The plots of average SSIM values obtained for IMAT and OMP are shown in Fig. 2 According to Figs. 1 and 2, the average PSNR and SSIM values of the recovered images using the IMAT method is higher than that of OMP for both transforms. Also, exact DCT performs better than WHT for both algorithms. For block sizes larger than 16×16, no meaningful improvement occurs in IMAT performance.

C. Computational Complexity Analysis and Comparison

This section presents an analysis of the computational complexity of the IMAT algorithm. DT and its inverse are the main computational burden of IMAT. For a 1-D signal with size N, direct computation of these transforms would require $\mathcal{O}(N^2)$ operations. It is possible to reduce the order to $\mathcal{O}(N\log_2 N)$ by using the fast algorithms. The exact

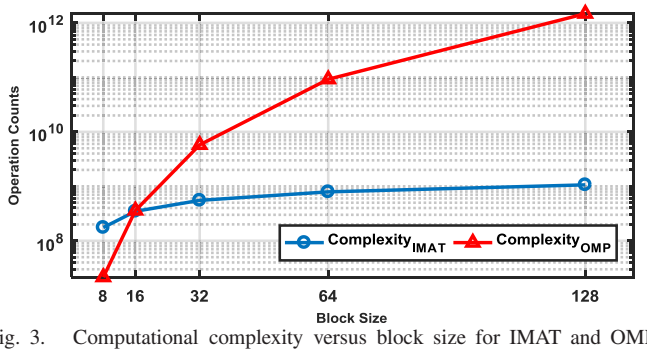


Fig. 3. Computational complexity versus block size for IMAT and OMP with DCT as the sparse domain.

number of arithmetic operations, depends on the type of the transform. The power-of-two fast WHT can be implemented without multiplication operation. It requires only $N\log_2 N$ additions or subtractions and N bit shifting operations. For DCT, there are several variants with slightly modified definitions. As the most commonly used type, the power-of-two DCT-II calculation using fast Fourier transform, combined with preprocessing and postprocessing steps, requires $2(N+1)\log_2 N$ multiplication and $(N+1)\log_2 N$ addition operations. For a 2-D signal like an $N \times N$ image block, by utilizing row–column decomposition scheme, the complexity increases by a factor of N. In each iteration of the IMAT algorithm, a 2-D DT and its inverse is calculated. Each iteration also contains N^2 comparisons and additions. Therefore, in each iteration, total arithmetic operations of the IMAT algorithm that use DCT can be calculated as $6(N^2+N)\log_2 N + 2N^2$. For $I_{\max}$ iterations, the total computational complexity of IMAT for each block of an image with size of N is given by

$$\text{complexity}_{\text{IMAT}} = (6(N^2+N)\log_2 N + 2N^2) \times I_{\max}. \quad (1)$$

For comparison, a complexity analysis of OMP algorithm, which is presented in [8], is considered. According to [8], for the OMP recovery of a 1-D k-sparse signal with length of n that sampled by an $m \times n$ Gaussian measurement matrix, the total number of multiplications, additions, comparisons, negates, and divisions is, respectively, $7k^3/6 + (m+1)k^2 + (\text{nm} + m - 2)k$, $4k^3/6 + (m-1)k^2 + (\text{nm} - n - 2/3)k$, $(n-1)k$, $k(k-1)/2$, and k. Therefore, the following number of arithmetic operations are required for OMP:

$$\frac{11}{6}k^3 + \left(2m + \frac{1}{2}\right)k^2 + \left(2\text{nm} + m - \frac{19}{6}\right)k. \quad (2)$$

To recover an $N \times N$ image block using OMP, it is vectorized into a 1-D signal with length of N^2 ($n = N^2$). As mentioned before, for OMP, the number of measurements (m) and sparsity order (k) are considered to be $0.25N^2$ and $0.03N^2$, respectively. Thus, the computational complexity of OMP for each block of an image with size of N can be simplified as follows:

$$\text{complexity}_{\text{OMP}} \cong 0.02N^6. \quad (3)$$

Fig. 3 shows the computational complexity of the two methods (OMP and IMAT) versus the block size. Here, we consider DCT as the transform. As shown in Fig. 3, with increasing the block size, the computational complexity increases slightly for

Authorized licensed use limited to: Universidad Federal de Pernambuco. Downloaded on June 01,2025 at 22:41:25 UTC from IEEE Xplore. Restrictions apply.

IMAT, whereas it increases considerably for OMP. For block sizes larger than 16×16, IMAT outperforms OMP in terms of computational complexity. Moreover, it presents better recovery performance than OMP. Due to its good complexity–performance tradeoff, we consdier 16×16 block size for IMAT hardware implementation.

III. Hardware Architecture Design

To reconstruct a random sampled image using IMAT, it is not necessary to divide the image into small blocks. However, it is an efficient approach for hardware implementation especially for large size problems. In hardware, all blocks are processed sequentially. For each selected block, the projection between the information domain and the sparsity domain requires a 2-D DT and its inverse. Thus, to expedite real-time implementations, using an efficient architecture for computing the 2-D DT is of great interest. Therefore, we first describe our structure for 2-D DT calculation and then use this structure to implement the IMAT algorithm.

A. 2-D Discrete Transform Implementation

Since the computation of the 2-D DT requires a large number of operations, it is decomposed into two 1-D DTs. The most commonly used algorithms are based on the row–column decomposition scheme because of its simplicity. In such scheme, a 2-D DT is built by running a 1-D DT over every row and then every column (or vice versa). In the other word, if D is considered as the DT matrix, for 2-D signal x, the 2-D DT can be computed as

$$\mathrm{DT}(x) = D \times x \times D^T \tag{4}$$

where $x \times D^T$ is the DT of the rows of x and the superscript "T" denotes matrix transposition. Equation (4) can be rewritten as follows:

$$\mathrm{DT}(x) = ((x \times D^T)^T \times D^T)^T. \tag{5}$$

According to (5), the DT of a 2-D signal can be calculated by the following procedure. First, 1-D transform of the rows of x is calculated, and then the result is transposed. Again, 1-D transform is applied to the rows of the resulted matrix and finally, the 2-D DT is obtained by performing a transposition operation. To implement the above procedure in hardware, the corresponding image block is stored in a row-major order in a dual-port on chip RAM which allows two reads or writes simultaneously. In each clock, the RAM outputs are forwarded to a serial in parallel out (SIPO) block. When the SIPO block receives all the elements of a row, it forms an array from those and send it to the DT block. The DT block computes the 1-D transform of each row and results an array with the same size of the input array. The elements of the resulting array are stored in another RAM discontinuously in such a way that it considered as a column of a matrix stored in row-major order. By using this approach, transposing the matrix is done during its saving process. This procedure is performed by a parallel in serial out (PISO) block. The input array is received by the PISO block and two of its elements are transferred to the second RAM in each clock. The PISO block also provides a corresponding address for each output to save it in an appropriate location on the second memory. Conceptual block diagram of the above procedure that calculates $(x \times D^T)^T$ is shown in Fig. 4.

On completion of one cycle of $(x \times D^T)^T$ computation, the two RAM's swap their functionality and this cycle is performed again for calculation of (5). After these two cycles, the first RAM contains 2-D DT of the input signal which stored in row-major order. For inverse transform calculations, D is replaced by D^T in the above computations.

In this paper, WHT and the ADCTADCT and their inverses are employed for projections between the information and sparsity domains. Both transforms are implemented with null multiplicative complexity.

B. Walsh–Hadamard Transform Implementation

WHT is a multiplication-free transform intrinsically. This property makes it attractive for hardware implementation. The forward and inverse WHT pair for a signal $x[u]$ of length N are

$$X[w] = (1/N) \sum_{u=0}^{N-1} x[u] H_N(u, w), \quad w = 0, \ldots, N-1 \tag{6}$$

$$x[u] = \sum_{w=0}^{N-1} X[w] H_N(u, w), \quad u = 0, \ldots, N-1. \tag{7}$$

H_N, known as the natural-ordered WHT matrix, is a symmetric $N \times N$ matrix as follows:

$$H_N(u, w) = (-1)^{u \cdot w}, \qquad u, w = 0, \ldots, N-1 \tag{8}$$

where $u \cdot w$ is the bitwise dot product of the binary formats of u and w. Both the forward and inverse transformations are identical operations except for the scaling factor of $1/N$. Since the entries of Hadamard matrix are 1 and -1, apart from the scaling factor, the transform has no multiplication operation and can be implemented efficiently in hardware. For the power-of-two sizes, the scaling factor can be implemented with bit shifting operation.

In this paper, a fast algorithm, similar to the Cooley–Tukey algorithm for discrete Fourier transform, is used to implement the WHT with complexity $\mathcal{O}(N \log N)$. The flow graph shown in Fig. 5 illustrates the computation of the fast WHT for $N = 16$. Dashed arrows represent multiplication by -1. Implementation of this structure requires 64 additions. Due to Hadamard matrix symmetry, the inverse WHT can be calculated using the same structure. Input and output nodes both appear in normal order. Since the geometry is identical for each stage, data and storage can be accessed sequentially.

C. Approximate Discrete Cosine Transform Implementation

The DCT is extensively used in image and video applications due to its good energy compaction property [20], [21]. Here, an approximated version of DCT is proposed as an alternative approach to implement the IMAT algorithm with low arithmetic cost. The forward DCT $X[w]$ of a sequence $x[u]$ with length N is defined as

$$X[w] = c_w \sqrt{\frac{2}{N}} \sum_{u=1}^{N-1} x[u] \cos\left[\frac{\pi w(2u+1)}{2N}\right], \quad w = 0, \ldots, N-1 \tag{9}$$

with $c_w = 1/\sqrt{2}$ for $w = 0$ and $c_w = 1$ otherwise.

Authorized licensed use limited to: Universidad Federal de Pernambuco. Downloaded on June 01,2025 at 22:41:25 UTC from IEEE Xplore. Restrictions apply.

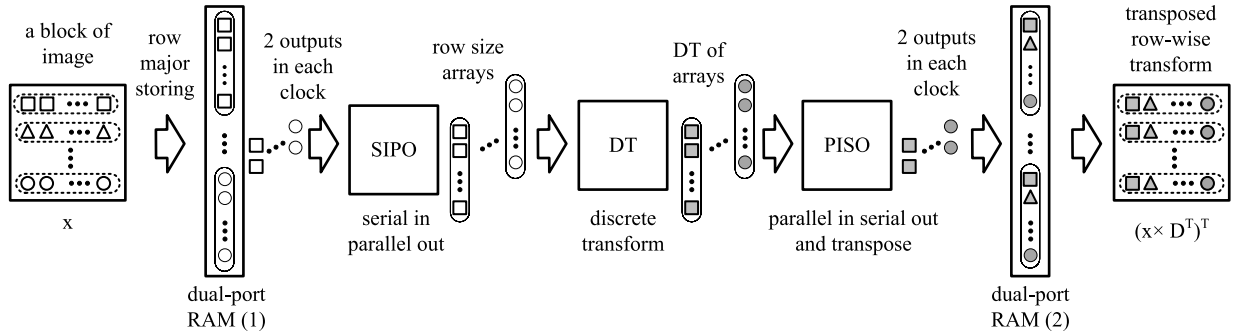


Fig. 4. Conceptual block diagram for computation of the 2-D DT.

To distinguish (9) from approximated forms of DCT, we refer to it as exact DCT. Equation (9) can be expressed in matrix form as

$$X = D \times x \tag{10}$$

where D is the transformation matrix of size $N \times N$ whose elements are given by

$$D(u, w) = c_w \sqrt{2/N} \cos\left[\frac{\pi w(2u+1)}{2N}\right], \quad 0 \le u, w \le N-1. \tag{11}$$

D is an orthogonal matrix, so the inverse transform can be expressed as

$$x = D^T \times X. \tag{12}$$

Several algorithms are proposed for fast computing of exact DCT. However, further complexity reduction can be obtained using approximate DCT algorithms in order to avoid multiplication. The approximation should preserve the global properties of the DCT as follows [20].

$$D_0 = \begin{bmatrix}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & -1 & -1 & 0 & 0 & 1 & 1 & 0 & 0 & -1 & -1 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & -1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & -1 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\
1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\
0 & -1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & -1 & 0 \\
0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & -1 & 1 & 0 & 0 & 1 & -1 & 0 & 0 & -1 & 1 & 0 & 0 & 1 & -1 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\
0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix} \tag{14}$$

$$D_0 = \begin{bmatrix}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\
1 & 0 & 0 & -1 & -1 & 0 & 0 & 1 & 1 & 0 & 0 & -1 & -1 & 0 & 0 & 1 \\
0 & 1 & -1 & 0 & 0 & -1 & 1 & 0 & 0 & 1 & -1 & 0 & 0 & -1 & 1 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 1 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0
\end{bmatrix} \tag{17}$$

Authorized licensed use limited to: Universidad Federal de Pernambuco. Downloaded on June 01,2025 at 22:41:25 UTC from IEEE Xplore. Restrictions apply.

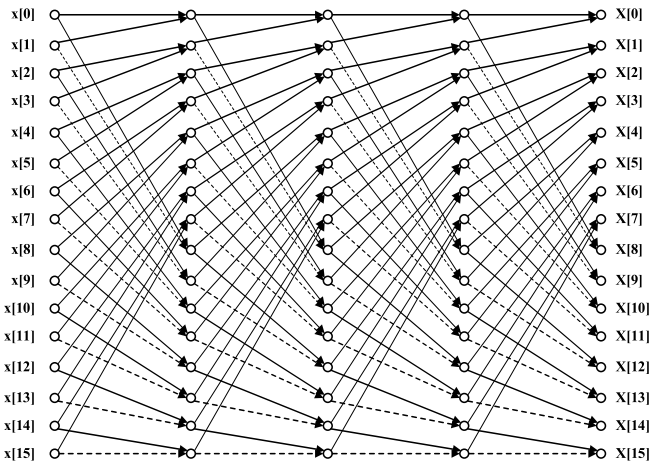


Fig. 5. Fast WHT implementation.

1) The approximated DCT matrix should satisfy the following orthogonality-like property

$$D \times D^T = \text{diagonal matrix.} \tag{13}$$

2) DCT symmetries should be conserved.
3) The approximated DCT matrix should exhibit the property of energy compaction.

In recent years, various approximate DCT methods have been proposed. In [20]–[22], various orthogonal 16-point approximations for DCT and their hardware realizations are proposed. Jridi *et al.* [23] present a generalized recursive algorithm to obtain orthogonal approximation of DCT where an approximate DCT of length N could be derived from a pair of DCTs of length $N/2$. A systematic method for developing a binary version of DCT is proposed in [24]. This transform is a permuted version of WHT and from implementation point of view, it possesses all the advantages of WHT. All of the above methods provide appropriate DCT approximation without using multiplication. In this paper, a new approximation scheme with acceptable quality is proposed to reduce the complexity of DCT even further. The proposed transformation matrix can also be implemented without multipliers. In the proposed method, the matrix is derived by a systematic modification of the exact DCT matrix. To reduce the computational complexity, in each row of the exact DCT matrix, the elements whose absolute values are the largest in that particular row are replaced by 1 or -1 (depending on its sign) and all the other elements are replaced by 0. For a 16×16 DCT matrix, the above mentioned procedure leads to the binary matrix in (14), shown at the bottom of the previous page. This matrix is an orthogonal-like matrix. However, it is possible to make it an orthogonal matrix ($D_0^{-1} = D_0^T$) using an adjustment matrix S as follows [20]:

$$D = S \times D_0 \tag{15}$$

where

$$S = \sqrt{\left(D \times D_0^T\right)^{-1}}. \tag{16}$$

For efficient hardware implementation, we rearrange the matrix in the order presented in (17), shown at the bottom of the previous page. This matrix has a regular signal flow graph, and simulation results show that this rearrangement does not affect the performance of the IMAT algorithm. For (17), we have

$$S = \text{diag}\left(\frac{1}{4}, \frac{1}{4}, \frac{1}{2\sqrt{2}}, \frac{1}{2\sqrt{2}}, \frac{1}{2}, \frac{1}{2}, \frac{1}{2}, \frac{1}{2}, \frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}\right). \tag{18}$$

In the IMAT algorithm, the diagonal matrix S is used in the computation of the forward transform (line 3 of Algorithm 1) and the inverse transform (line 6 of Algorithm 1) as follows:

$$X^{(i)} = S \times D_0 \times (X^{(i)}) \times D_0^T \times S \tag{19}$$

$$\theta_x = D_0^T \times S \times (\theta_{\text{mask}} \times X^{(i)}) \times S \times D_0. \tag{20}$$

By transferring matrix S from (19) into (20), these two equations can be rewritten as

$$X^{(i)} = D_0 \times (X^{(i)}) \times D_0^T \tag{21}$$

$$\theta_x = D_0^T \times S \times (\theta_{\text{mask}} \times X^{(i)}) \times S^2 \times D_0. \tag{22}$$

As can be seen, the computation cost of (21) is comparable to the computational complexity of matrix D_0. In (22), the left and right multiplication by S^2 can be implemented by bit shifting operations and the remaining operations are performed with the same low computational complexity of D_0.

The flow graph of Fig. 6 describes the implementation algorithm for computation of the proposed approximate DCT. The proposed algorithm requires only 30 additions without any multiplication operations. Since the proposed transformation matrix is not symmetric, the inverse transform operation requires different structures. The flow graph of Fig. 7 depicts a structure for the approximated inverse transform calculation. By rearranging this structure as shown in Fig. 8, all stages can be calculated by a similar structure. To achieve this goal, some multiplexers are inserted between the stages.

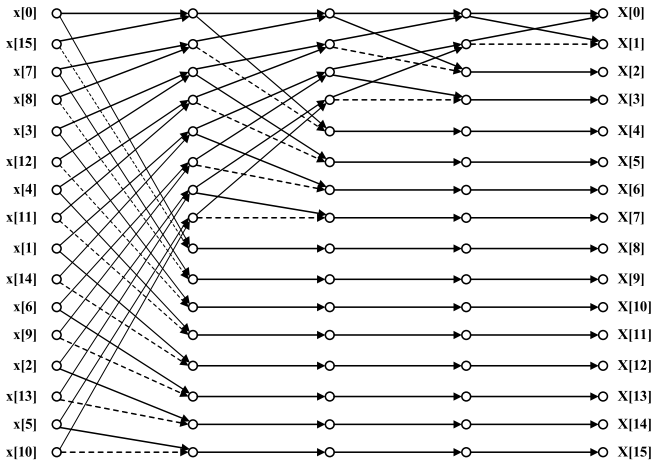


Fig. 6. Fast implementation of the proposed approximated DCT.

D. Performance Assessment

To compare the performance of the proposed approximated DCT with other approximated transforms and also with WHT, first some known grayscale test images such as *Barbara*, *Lena*, and *Cameraman* are randomly sampled by a 16×16 binary mask in the block-wise manner. Then, the IMAT reconstruction algorithm using different transforms are applied

Authorized licensed use limited to: Universidad Federal de Pernambuco. Downloaded on June 01,2025 at 22:41:25 UTC from IEEE Xplore. Restrictions apply.

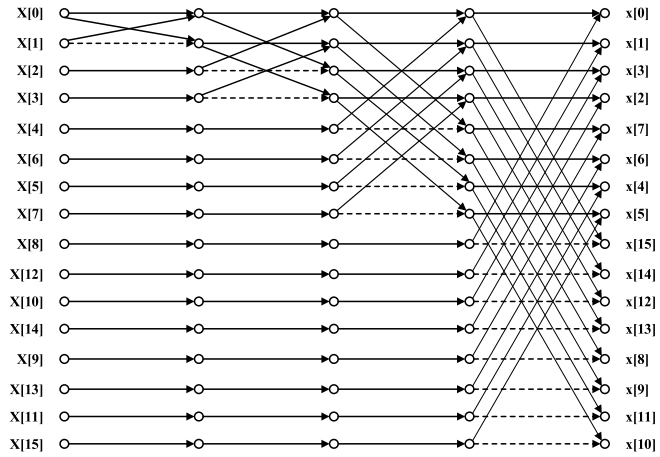


Fig. 7. Fast implementation of the proposed approximated IDCT.

TABLE I

COMPARISON OF PERFORMANCES AND COMPUTATIONAL COMPLEXITIES FOR 16 × 16 BLOCK SIZE

Transform	Operations		PSNR(dB) / SSIM					
	Mult	Add	Barbara		Lena		Cameraman	
DCT	136	68	25.09	0.79	27.97	0.83	22.05	0.73
[20]	0	44	22.49	0.66	25.18	0.79	20.60	0.71
[21]	0	60	23.07	0.70	26.06	0.77	21.12	0.70
[22]	0	72	23.11	0.69	25.37	0.75	20.64	0.69
[23]	0	60	23.41	0.72	26.30	0.80	21.22	0.74
[24][a]	0	64	23.16	0.71	25.98	0.78	21.12	0.71
WHT[a]	0	64	23.16	0.71	25.98	0.78	21.12	0.71
ADCT	0	30	21.41	0.61	23.45	0.69	19.58	0.66

[a] The IMAT algorithm is not sensitive to the displacement of rows in the transformation matrix. So, these two transforms have identical performance.

to each sampled image. Table I compares PSNR and SSIM measurements for all of the recovered images. The computational complexity of each transform is also depicted in the table for a block of size 16 × 16. In comparison with the other methods, the proposed approximation requires less arithmetic operations while presents an acceptable performance.

To illustrate the energy compaction capability of the ADCT, the energy compaction properties of the ADCT and the DCT are compared as follows. First, the normalized cumulative energy is calculated for each block of an image, and then the mean value is computed for all blocks. This process is performed for all images from the database, and then the average value is obtained. The comparison between the performances of two transforms is shown in Fig. 9. According to Fig. 9, the ADCT has an acceptable performance in terms of energy compaction compared to the DCT.

E. Proposed Architecture for IMAT Using WHT and ADCT

The proposed hardware architecture for processing each block using IMAT algorithm is shown in Fig. 10. The architecture includes three dual-port RAMs (RAM1–RAM3) that for simplification, only one port of them are depicted in Fig. 10. The architecture also includes a single-port RAM for storing the threshold values. The binary sampling mask is also stored row by row in the mask_register. At the beginning of the processing, the entries of a new image block are stored row by row in RAM1 and RAM2. For a specific image block, the content of RAM1 remains unchanged during the iterations and at the end of each iteration, its content is used for residual updating (line 7 of Algorithm 1). RAM2 and

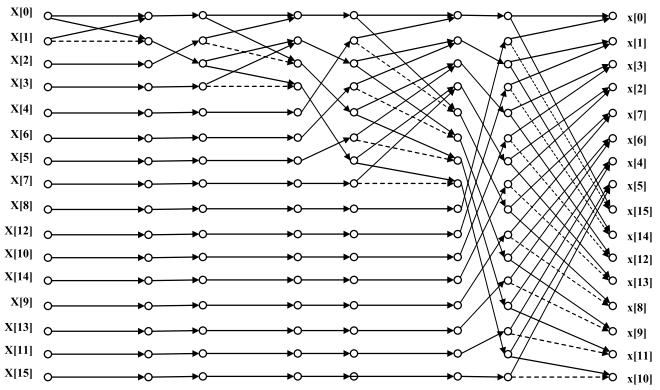


Fig. 8. Modified structure of the proposed approximated IDCT.

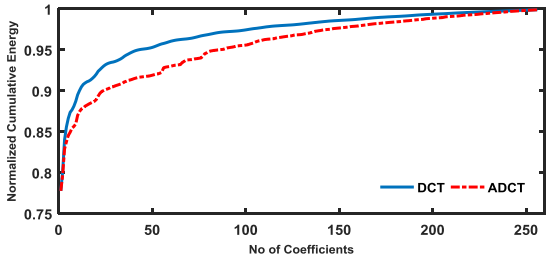


Fig. 9. Energy compaction capability of ADCT and DCT.

RAM3 are used for DT and IDT calculations based on the conceptual block diagram of Fig. 4 and their contents are changed several times during the algorithm execution on an image block. When RAM2 acquires the data of a new block, it delivers the recovered version of the preceding block to the output. A multiplexer-like block with some additional functions, as shown in Table II, is placed before each RAM. Based on the value of the "select" input, the MUX block forward the selected input (including data and/or address) into RAMs. The DT/IDT block operates in two different modes. The forward transform is calculated in one mode and the inverse transform in another mode. Switching between two modes is controlled by "sel" input, which can be either zero or one for DT and IDT, respectively. For WHT, the two modes are identical.

The threshold values depend only on the iteration number. So, instead of computing line 4 of Algorithm 1, threshold values are stored in the threshold_RAM. The output of the threshold_RAM depends on the iteration number which specified by the control block. The threshold values are given to the multiplexers, and the thresholding can be performed in the MUX block when the data passing through it. Algorithm procedure is controlled by a control unit that contains a finite-state machine (FSM). This unit generates all control signals necessary for the algorithm implementation. The algorithm execution procedure to reconstruct an image block is controlled as follows. The controller outputs "mux1_sel," "mux2_sel" are set to "01" and "001," respectively. As a result, the input data are transferred to RAM1 and RAM2 through the multiplexers (In_A). Appropriate addresses are also provided for corresponding inputs by the controller block (Addr_A) in such a way that the input image block is stored row by row in the RAMs. The recovery algorithm is then started by reading data from RAM2. The controller gives addresses to the RAM2 via MUX2, and the data are fed to the SIPO block

Authorized licensed use limited to: Universidad Federal de Pernambuco. Downloaded on June 01,2025 at 22:41:25 UTC from IEEE Xplore. Restrictions apply.

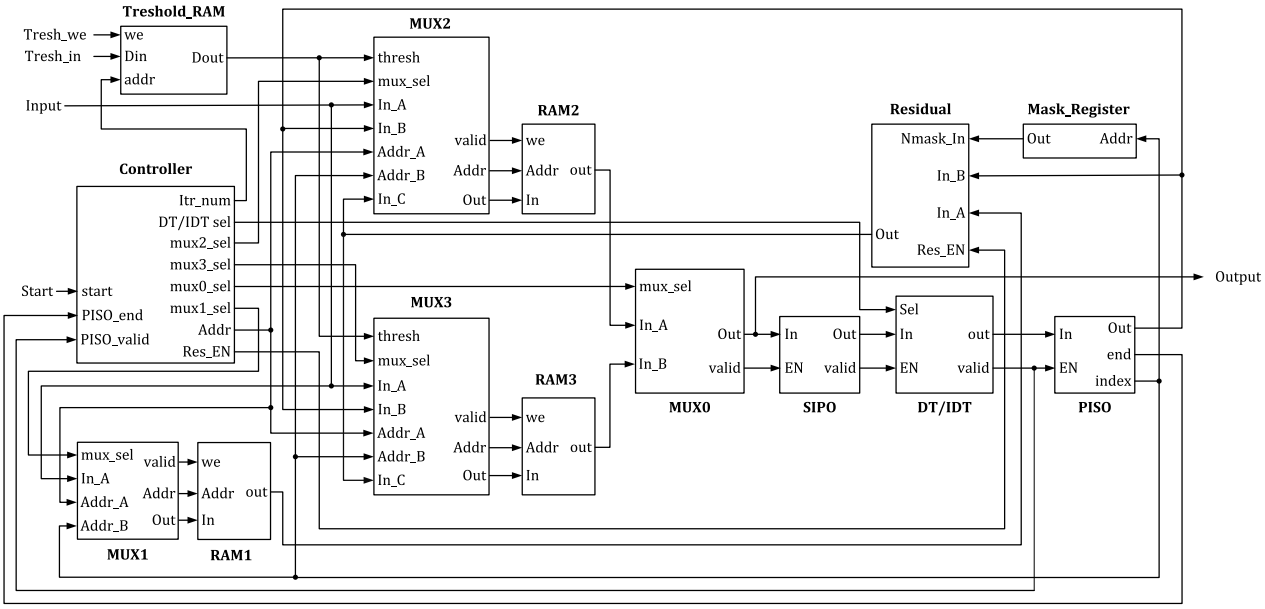


Fig. 10. Proposed architecture for IMAT recovery algorithm implementation.

through MUX0. The SIPO block brings all the elements of a row together in form of an array. The resulting array is passed through DT/IDT block which calculates the transform of the array in six clock cycles. Two clock cycles are devoted for DT/IDT input and output rearrangement and four clock cycles for DT/IDT calculations. The resulting array is then fed to the PISO block. The PISO block disjoints the array and sends its elements with suitable addresses to the RAM3. Now, the content of RAM3 is the transposed row-wise DT of RAM2. To calculate 2-D DT, the row-wise DT is applied again to the content of RAM3 and the result is stored in the transposed form in RAM2. Now, the content of RAM2 is the 2-D DT of the input block (line 3 of Algorithm 1). In this step, before storing data in RMA2, the threshold is applied to data in MUX2 when the data are transferred from PISO block to RAM2 (line 5 of Algorithm 1). After applying the threshold, the above procedure is repeated again for 2-D IDT calculation (line 6 of Algorithm 1). Finally, the residual is updated using $x^{(0)}$, S_{mask}, and $x_\theta^{(i-1)}$ according to line 7 of Algorithm 1. These parameters are received from RAM1, mask_register, and RAM2, respectively, and the updated residual is stored in RAM3. For the next iteration, all of the above procedures are repeated for the new residual and the new threshold. After sufficient number of iterations, the data of the recovered block are delivered to the output port.

IV. Timing Analysis

In this section, the necessary number of cycles to reconstruct an input image is calculated. The control procedure described in the previous section is implemented using the Moore FSM. This part uses a clock counter to control the timing of each state. The operations at each state of the FSM controller and corresponding number of clock cycles are shown in Table III. Also, timing diagram of the first three states (S0, S1, and S2) are given in Fig. 11. Each number in Fig. 11 represents an address of the RAM where the data is written or read. The areas shaded by the gray color exhibit the transform domain data, and the areas marked by "X" correspond to the invalid

TABLE II
Functionality of the Multiplexers

	select	Operations
MUX0	00	No Transfer
	01	Out ← In_A, valid ← "1"
	10	Out ← In_B, valid ← "1"
MUX1	00	No Transfer
	01	Addr ← Addr_B, valid ← "0"
	10	Addr ← Addr_B , Out ← In_B, valid ← "1"
	11	Addr ← Addr_A, Out ← In_A, valid ← "1"
MUX2 MUX3	000	No Transfer
	001	Addr ← Addr_B, valid ← "0"
	010	Addr ← Addr_B, Out ← In_B, valid ← "1"
	011	Addr ← Addr_B, Out ← Tresh (In_B), valid ← "1"
	100	Addr ← Addr_B ,Out ← In_C, valid ← "1"
	101	Addr ← Addr_A, Out ← In_A, valid ← "1"

data. Adding the numbers shows that the total number of clock cycles for recovery a 16×16 block is $708 \times I_{\max}$. Thus, to recover an image with size of 512×512, $1024 \times 708 \times I_{\max}$ clock cycles are required. In our design, IMAT is configured to perform a fixed number of $I_{\max} = 30$ iterations. The reconstruction time is calculated by multiplication of minimum cycle period and the necessary number of cycles.

V. Implementation Results and Comparison

Since FPGA is a flexible platform in both system level design and circuit level implementation we mapped our IMAT architecture to Xilinx Virtex6 FPGA using VHDL. The architecture supports user defined parameters. Customization of the parameters at the compile time provides the configurability of the implemented hardware. The fixed point data format is used for all signals and the number of bits for the integer and the fractional parts are defined as parameters. The input image is normalized to the maximum pixel value (255), and then it converted into the fixed point representation. Two different data precisions, 16 and 32 bit, are considered for hardware design and the appropriate numbers of bits for integer and fractional parts of internal signals are determined by comparing the hardware model with the MATLAB simu-

Authorized licensed use limited to: Universidad Federal de Pernambuco. Downloaded on June 01,2025 at 22:41:25 UTC from IEEE Xplore. Restrictions apply.

TABLE III
OPERATIONS AT EACH STATE OF THE FSM CONTROLLER AND THE REQUIRED CLOCK NUMBERS FOR EACH STATE

State	Operation	Control signals	CLKs
S0	Loading data of a block (16×16) into RAM1 and RAM2 (Generating addresses for writing to RAM1 and RAM2)	mux0_sel ←"00", mux1_sel ←"11" mux2_sel ←"101", mux3_sel ←"000" Addr_A(MUX1)←Addr (controller), Addr_A(MUX2)←Addr (controller)	128
S1	Generating addresses for reading data from RAM2 or RAM3	mux2_sel ←"001", DT/IDT sel ←"0", mux0_sel ←"01" if PISO_valid="1": mux3_sel ←"010" Addr_A(MUX2)←Addr (controller), Addr_B(MUX3)←PISO_Index	128
S2	Waiting	mux0_sel ←"00", Waiting for PISO_end="1" and then going to S3	16
S3	Generating addresses for reading data from RAM2 or RAM3 (RAM2 and RAM3 swap their functionality)	mux3_sel ←"001", mux0_sel ←"10", DT/IDT sel ←"0" if PISO_valid="1": mux2_sel ←"011", Addr (Thresh_RAM)←Iter_Num Addr_A(MUX3)←Addr (controller), Addr_B(MUX2)←PISO_Index	128
S4	Waiting	mux0_sel ←"00", Waiting for PISO_end="1" and then going to S5	16
S5	Generating addresses for reading data from RAM2/RAM3	mux2_sel ←"001", mux0_sel ←"01", DT/IDT sel ←"1" if PISO_valid="1": mux3_sel ←"010" Addr_A(MUX2)←Addr (controller), Addr_B(MUX3)←PISO_Index	128
S6	Waiting	mux0_sel ←"00", Waiting for PISO_end="1" and then going to S7	16
S7	Generating addresses for reading data from RAM2/RAM3 (residual calculation)	mux3_sel ←"001", mux0_sel ←"10", DT/IDT sel ←"1", Res_EN←"1" if PISO_valid="1": mux2_sel ←"100", Addr_B(MUX1)←PISO_Index Addr_A(MUX3)←Addr (controller), Addr_B(MUX2)←PISO_Index	132
S8	if the iteration number equals to its maximum value, the next state becomes S0 for receiving a new block	mux0_sel ←"00", Res_EN←"0" Waiting for PISO_end="1" and then going to S1	16

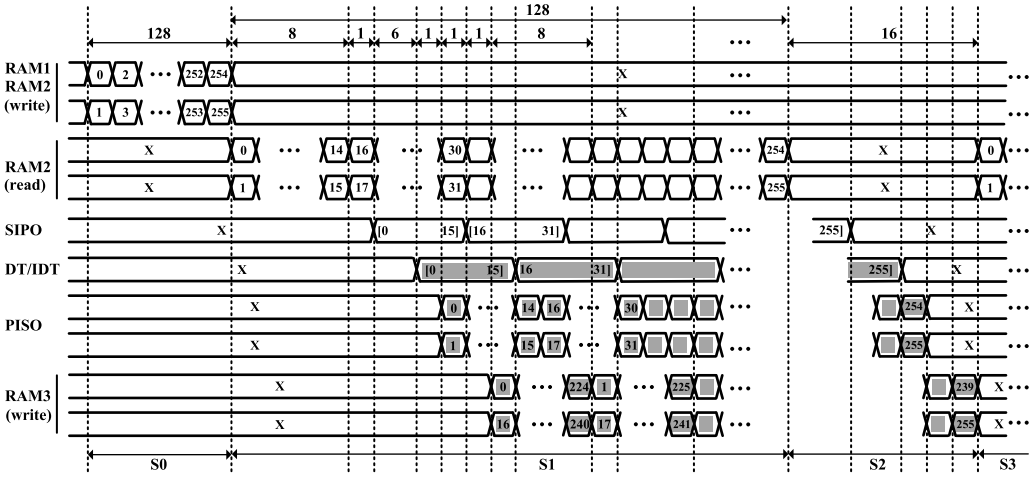


Fig. 11. Timing diagram for the control sates S0, S1, and S2.

lation. To evaluate the reconstruction quality of the proposed hardware implementation, first, the hardware outputs and its floating point model (MATLAB) results are compared with the original image in terms of PSNR and SSIM metrics. Second, to illustrate the closeness of hardware and software results, the difference between these two results is computed directly using the PSNR metric. To demonstrate the performance of the designed hardware, we consider standard 512 × 512 *Barbara* image as a case study. The input image is divided into nonoverlapping 16 × 16 blocks which are then randomly sampled by a binary mask. The IMAT algorithm is applied on the disjoint blocks of the image separately in both hardware and software (MATLAB). Fig. 12 shows the qualitative and quantitative comparison between hardware and software implementation of the IMAT algorithm using WHT and ADCT compared to the original image. Direct comparison between the hardware implementation results and the MATLAB results as the reference shows that the implementation using Hadamard transform provides the PSNR of 42.52 and 33.97 dB for 32- and 16-bit precision, respectively. Also, the implementation using ADCT transform provides the PSNR of 37.83 and 32.37 dB for the mentioned precision. This comparison implies that the performance of the hardware implementation is very close to its floating point simulated counterpart in MATLAB, when appropriate number of bits is used. As another result, IMAT realization using WHT presents better recovery performance.

The implementation results of the two designs (WHT and ADCT) are shown in Table IV. As it can be seen, the recovery time is identical for these two cases and for hardware resource utilization, there is a nearly linear relationship. The results also demonstrate that using WHT for IMAT implementation is more efficient than using ADCT in terms of hardware resources. However, ADCT has a lower power consumption than WHT. The results are inferred from the difference between the fast implementation of WHT and ADCT. As shown in Fig. 5, in the implementation of WHT, the geometry is identical for each stage. Therefore, by implementation of one stage in hardware, all stages can be calculated sequentially. On the other hand, due to Hadamard matrix symmetry, the inverse WHT can be calculated using the same structure. Consequently, the WHT and its inverse are implemented with eight adders and eight subtractors without any additional hardware. As shown in Fig. 6, the ADCT is also implemented with eight adders and eight subtractors but since

Authorized licensed use limited to: Universidad Federal de Pernambuco. Downloaded on June 01,2025 at 22:41:25 UTC from IEEE Xplore. Restrictions apply.

Fig. 12. IMAT recovery performance for hardware implementation and its floating point simulated counterpart in MATLAB compared to the original image. (a) Original image. (b) Random sampled image with a sampling rate of 0.25. Recovered image in MATLAB using (c) exact DCT, (d) WHT, and (e) approximate DCT. Recovered image in hardware using (f) WHT (32 bit), (g) approximate DCT (32 bit), (h) WHT (16 bit), and (i) approximate DCT (16 bit).

TABLE IV
IMPLEMENTATION RESULTS FOR XILINX VIRTEX6 FPGA (XC6VLX240T)

Architecture	IMAT-WHT		IMAT-ADCT	
Precision	16-bit	32-bit	16-bit	32-bit
Max Frequency[MHz]	230	230	230	230
Recovery Time (ms)	95	95	95	95
Occupied Slices	451 (1%)	899 (2%)	484 (1%)	1032 (2%)
DSP Cores	0	0	0	0
Block RAMs[a]	3 (1%)	3 (1%)	3 (1%)	3 (1%)
Dynamic Power[mw][b]	82	174	79	170

[a] The type of RAM blocks are RAMB18E1 and RAMB 36E1 for 16-bit and 32-bit, respectively.
[b] Power consumption is measured using Xilinx XPower Analyzer after Place and Route.

in each stage the input and output nodes do not appear in normal order, some multiplexers should be inserted between the stages. So, the ADCT implementation requires more hardware resource. From the other point of view, the ADCT requires less operations than WHT. According to Fig. 6, only in the first stage all operations (eight additions and eight subtractions) are performed and in the other stages the operations are decreased sequentially. Therefore, ADCT consumes a lower dynamic power than WHT. However, due to power consumption by multiplexers, the power reduction is less than expected.

In the following, we also compare our IMAT recovery design with some existing sparse recovery FPGA implementations. Since other implantations are proposed for 1-D signals,

TABLE V
COMPARISON WITH EXISTING RECOVERY IMPLEMENTATIONS

	Proposed	[7]	[7]	[8]
Algorithm	IMAT	OMP	AMP	OMP
Transform	WHT	Wavelet	Wavelet	N/A
Precision	18-bit	18-bit	18-bit	18-bit
Input Size	(32×32)	1024	1024	1024
Platform	Virtex6	Virtex6	Virtex6	Virtex6
Max Frequency [MHz]	240	100	165	119.96
Recovery Time (μs)	185	622	633	340
Occupied Slices	661	32010	12113	6208
DSP Cores	0	258	256	589
Block RAMs	3[b]	261	258	576
Dynamic Power [mw]	123	N/A	N/A	3233
RSNR (dB)[a]	22.84	23.5	21.4	N/A[c]

[a] RSNR is measured for 256×256 Lena image.
[b] The type of RAM block is RAMB18E1.
[c] The recovery performance for image signals is unknown.

an $N \times N$ image block should be vectorized into a 1-D signal with length of N^2. To have similar conditions for comparison, the block size of 32×32 correspond to 1-D problems with size of 1024 is considered and IMAT architecture is adapted to deal with subimages of this dimension. Since other works are implemented with 18-bit precision, the proposed architecture is also implemented with the similar precision. Table V shows the comparison in terms of hardware resources, power consumption, and recovery time for a block of an image. The recovery performance is also reported using the recovery signal-to-noise ratio (RSNR), defined as $\text{RSNR} = 10\log(\|x\|_2^2/\|x-\tilde{x}\|_2^2)$, where x is the original image and $\tilde{x}$ is the recovered image. As shown in Table V, IMAT architecture is the preferred approach for hardware implementation in terms of required resources, throughput, and power consumption.

VI. CONCLUSION

High-performance sparse recovery algorithms, especially for 2-D sparse signals such as images and videos, require significant computational effort. For real-time applications, the enormous amount of computations prevents applying complex algorithms to recover the sampled signals in a time efficient manner. It is also a serious limitation for portable platforms which are constrained by cost, energy, and computational power. Therefore, simple algorithms such as IMAT are a preferred approach to achieve acceptable performance with affordable computational cost. In this algorithm, by exchanging projections between information and sparsity domains, alongside the adaptive decrease of the threshold, the unknown sparse coefficients are gradually picked up after suitable number of iterations. Since the main operation of IMAT algorithm is the DT, it can take advantage of fast implementations. In this paper, we use two multiplierless transforms for efficient implementation of the IMAT algorithm. These transforms are implemented without multiplication operation and this makes them the preferred candidates for hardware implementation. The postplace and route analysis shows that IMAT implementation using WHT requires 7% and 15% less hardware resources for 16- and 32-bit precisions, respectively, compared to implementation using ADCT. On the other hand, implementation using ADCT consumes 4% and 2% less dynamic

Authorized licensed use limited to: Universidad Federal de Pernambuco. Downloaded on June 01,2025 at 22:41:25 UTC from IEEE Xplore. Restrictions apply.

power for the mentioned precisions. Compared with the OMP and AMP recovery implementations using wavelet transform, the proposed IMAT implementation using WHT consumes significantly lower hardware resources and provides good performance. As a study case, for a 256×256 *Lena* image, the IMAT implementation gives 7% more and 3% less PSNR compared to the AMP and OMP implementations, respectively. Also, the proposed design requires approximately 3 times less reconstruction time compared to the mentioned implementations.

REFERENCES

[1] D. L. Donoho, "Compressed sensing," *IEEE Trans. Inf. Theory*, vol. 52, no. 4, pp. 1289–1306, Apr. 2006.

[2] E. J. Candès, J. Romberg, and T. Tao, "Robust uncertainty principles: Exact signal reconstruction from highly incomplete frequency information," *IEEE Trans. Inf. Theory*, vol. 52, no. 2, pp. 489–509, Feb. 2006.

[3] S. Boyd and L. Vandenberghe, *Convex Optimization*. New York, NY, USA: Cambridge Univ. Press, 2004.

[4] J. A. Tropp and A. C. Gilbert, "Signal recovery from random measurements via orthogonal matching pursuit," *IEEE Trans. Inf. Theory*, vol. 53, no. 12, pp. 4655–4666, Dec. 2007.

[5] T. Blumensath and M. E. Davies, "Iterative hard thresholding for compressed sensing," *Appl. Comput. Harmon. Anal.*, vol. 27, no. 3, pp. 265–274, Nov. 2009.

[6] D. L. Donoho, A. Maleki, and A. Montanari, "Message-passing algorithms for compressed sensing," *Proc. Nat. Acad. Sci. USA*, vol. 106, no. 45, pp. 18914–18919, 2009.

[7] L. Bai, P. Maechler, M. Muehlberghuber, and H. Kaeslin, "High-speed compressed sensing reconstruction on FPGA using OMP and AMP," in *Proc. 19th IEEE Int. Conf. Electron., Circuits Syst. (ICECS)*, Dec. 2012, pp. 53–56.

[8] H. Rabah, A. Amira, B. K. Mohanty, S. Almaadeed, and P. K. Meher, "FPGA implementation of orthogonal matching pursuit for compressive sensing reconstruction," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 23, no. 10, pp. 2209–2220, Oct. 2015.

[9] A. Kulkarni, J. L. V. M. Stanislaus, and T. Mohsenin, "Parallel heterogeneous architectures for efficient OMP compressive sensing reconstruction," *Proc. SPIE*, vol. 9109, p. 91090G, May 2014.

[10] J. L. V. M. Stanislaus and T. Mohsenin, "High performance compressive sensing reconstruction hardware with QRD process," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2012, pp. 29–32.

[11] F. Ren and D. Marković, "A configurable 12–237 kS/s 12.8 mW sparse-approximation engine for mobile data aggregation of compressively sampled physiological signals," *IEEE J. Solid-State Circuits*, vol. 51, no. 1, pp. 68–78, Jan. 2016.

[12] A. M. Kulkarni, H. Homayoun, and T. Mohsenin, "A parallel and reconfigurable architecture for efficient OMP compressive sensing reconstruction," in *Proc. 24th Ed. Great Lakes Symp. VLSI*, New York, NY, USA, 2014, pp. 299–304.

[13] G. Huang and L. Wang, "High-speed signal reconstruction for compressive sensing applications," *J. Signal Process. Syst.*, vol. 81, no. 3, pp. 333–344, Dec. 2015.

[14] A. Kulkarni and T. Mohsenin, "Low overhead architectures for OMP compressive sensing reconstruction algorithm," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 64, no. 6, pp. 1468–1480, Jun. 2017.

[15] P. Maechler *et al.*, "VLSI design of approximate message passing for signal restoration and compressive sensing," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 2, no. 3, pp. 579–590, Sep. 2012.

[16] F. Marvasti *et al.*, "A unified approach to sparse signal processing," *EURASIP J. Adv. Signal Process.*, vol. 2012, p. 44, Feb. 2012.

[17] F. Marvasti and M. B. Mashadi, "Wideband analog to digital conversion by random or level crossing sampling," U.S. Patent 9 729 160, Aug. 8, 2017.

[18] *Dataset of Standard 512 × 512 Grayscale Test Images*. Accessed: Jul. 24, 2017. [Online]. Available: http://decsai.ugr.es/cvg/CG/base.htm

[19] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: From error visibility to structural similarity," *IEEE Trans. Image Process.*, vol. 13, no. 4, pp. 600–612, Apr. 2004.

[20] T. L. T. da Silveira, R. S. Oliveira, F. M. Bayer, R. J. Cintra, and A. Madanayake, "Multiplierless 16-point DCT approximation for low-complexity image and video coding," *Signal Image Video Process.*, vol. 11, no. 2, pp. 227–233, Feb. 2017.

[21] T. L. T. da Silveira, F. M. Bayer, R. J. Cintra, S. Kulasekera, A. Madanayake, and A. J. Kozakevicius, "An orthogonal 16-point approximate DCT for image and video compression," *Multidimens. Syst. Signal Process.*, vol. 27, no. 1, pp. 87–104, Jan. 2016.

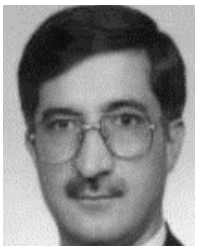
[22] F. M. Bayer, R. J. Cintra, A. Edirisuriya, and A. Madanayake, "A digital hardware fast algorithm and FPGA-based prototype for a novel 16-point approximate DCT for image compression applications," *Meas. Sci. Technol.*, vol. 23, no. 11, p. 114010, 2012.

[23] M. Jridi, A. Alfalou, and P. K. Meher, "A generalized algorithm and reconfigurable architecture for efficient and scalable orthogonal approximation of DCT," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 62, no. 2, pp. 449–457, Feb. 2015.

[24] S. Bouguezel, M. O. Ahmad, and M. N. S. Swamy, "Binary discrete cosine and Hartley transforms," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 60, no. 4, pp. 989–1002, Apr. 2013.

Mohammad Fardad received the B.Sc. degree in electrical engineering from the University of Guilan, Rasht, Iran, in 2007 and the M.Sc. degree in electrical engineering from the University of Tabriz, Tabriz, Iran, 2010. He is currently working toward the Ph.D. degree in electrical engineering at the Isfahan University of Technology, Isfahan, Iran.

His current research interests include design and implementation of signal processing algorithms and low-power hardware designs.



Sayed Masoud Sayedi received the B.Sc. and M.Sc. degrees in electrical engineering from the Isfahan University of Technology (IUT), Isfahan, Iran, in 1986 and 1988, respectively, and the Ph.D. degree in electronics from Concordia University, Montreal, QC, Canada, in 1996.

From 1988 to 1992, and then since 1997, he has been with IUT, where he is currently a Full Professor at the Department of Electrical and Computer Engineering. His current research interests include VLSI fabrication processes, image sensors, low-power VLSI circuits, and data converters.

Ehsan Yazdian received the B.Sc. degree in electrical engineering from the Isfahan University of Technology (IUT), Isfahan, Iran, in 2004 and the M.Sc. and Ph.D. degrees in electrical engineering from the Sharif University of Technology, Tehran, Iran, in 2006 and 2012, respectively.

Since 2013, he has been with the Electrical Engineering Department, IUT. His current research interests include statistical array signal processing, wireless communications, digital communication systems, software defined radio, and design and implementation of signal processing algorithms on FPGA.

Authorized licensed use limited to: Universidad Federal de Pernambuco. Downloaded on June 01,2025 at 22:41:25 UTC from IEEE Xplore. Restrictions apply.