Article

Small-Size Algorithms for the Type-I Discrete Cosine Transform with Reduced Complexity

Miłosz Kolenderski and Aleksandr Cariow *

Faculty of Computer Science and Information Technology, West Pomeranian University of Technology, Żołnierska 52, 71-210 Szczecin, Poland; km46548@zut.edu.pl
* Correspondence: acariow@wi.zut.edu.pl

Abstract: Discrete cosine transforms (DCTs) are widely used in intelligent electronic systems for data storage, processing, and transmission. The popularity of using these transformations, on the one hand, is explained by their unique properties and, on the other hand, by the availability of fast algorithms that minimize the computational and hardware complexity of their implementation. The type-I DCT has so far been perhaps the least popular, and there have been practically no publications on fast algorithms for its implementation. However, at present the situation has changed; therefore, the development of effective methods for implementing this type of DCT becomes an urgent task. This article proposes several algorithmic solutions for implementing type-I DCTs. A set of type-I DCT algorithms for small lengths $N = 2, 3, 4, 5, 6, 7, 8$ is presented. The effectiveness of the proposed solutions is due to the possibility of fortunate factorization of the small-size DCT-I matrices, which reduces the complexity of implementing transformations of this type.

Keywords: digital signal processing; type-I discrete cosine transform; fast algorithms

Citation: Kolenderski, M.; Cariow, A. Small-Size Algorithms for the Type-I Discrete Cosine Transform with Reduced Complexity. *Electronics* **2022**, *11*, 2411. https://doi.org/10.3390/electronics11152411

Academic Editors: Constantin Paleologu and Chiper Doru Florin

Received: 22 June 2022
Accepted: 26 July 2022
Published: 2 August 2022

1. Introduction

Discrete cosine transform (DCT) [1–6] is widely used in many radio-electronic and telecommunication systems for data processing and transmission, including digital signal and image processing [7–9], radar imaging [10], digital watermarking [11,12], analysis of hyperspectral data [13,14], video compression [15–22], etc. In fact, there are eight different types of DCTs [4–6]. In the DCT arsenal, the Type I Discrete Cosine Transform (DCT-I) is one of the less popular ones. However, recently it has been increasingly used in wireless communication systems in order to modernize the multicarrier modulation and channel estimation techniques for Long Term Evolution (LTE) [23–27]. Since, like other types of orthogonal transformations, the implementation of the DCT-I transformation requires a lot of time, the search for algorithmic solutions that can reduce this time is an urgent task. The reduction of the number of arithmetic operations is provided by the so-called fast algorithms. Unfortunately, there are undeservedly few articles devoted to fast algorithms for calculating DCT-I. With rare exceptions, most publications known to the authors mainly deal with fast algorithms for other types of DCT. It should be noted that, as in the case of other discrete orthogonal transformations [28–30], the DCT-I algorithms for short sequences are also of particular interest. In the case of hardware or software implementation of digital signal processing methods, small-sized DCT-I implementation cores can serve as building blocks for the synthesis of larger-size algorithms [4,7,31–33]. Despite this, there is practically no information about DCT-I algorithms for short-length sequences in the publications available to the authors. To eliminate these shortcomings, fast DCT-I algorithms for input sequences of length $N = 2, 3, 4, 5, 6, 7, 8$ are described in detail. This article continues the series of publications related to the development of small-sized algorithms for fast orthogonal transforms [28–30].

2. Preliminary Remarks

The DCT-I transform is given by the following equation [3–5]:

$$c_k = \sqrt{\frac{2}{N-1}} \sum_{n=0}^{N-1} x_n \varepsilon_n \varepsilon_k \cos\left(\frac{\pi nk}{N-1}\right), \tag{1}$$

$$\varepsilon_n, \varepsilon_k = \begin{cases} \frac{1}{\sqrt{2}} & n,k = 0, \\ \frac{1}{\sqrt{2}} & n,k = N-1, \\ 1 & \text{otherwise}, \end{cases}$$

$$k, n = 0, 1, \ldots, N-1,$$

where $\{x_n\}$ is an input data sequence and $\{c_n\}$ is a sequence of DCT coefficients. In matrix-vector notation the pair of FTCT/IDCT transforms can be represented as:

$$\mathbf{Y}_{N\times 1} = \mathbf{C}_N \mathbf{X}_{N\times 1}, \mathbf{X}_{N\times 1} = \mathbf{C}_N^{\mathrm{T}} \mathbf{Y}_{N\times 1}, \tag{2}$$

where $\mathbf{C}_N = \|c_{k,n}\|$ is $(N \times N)$ discrete cosine transform matrix, $\mathbf{X}_{N\times 1} = [x_0, x_1, \ldots, x_{N-1}]^{\mathrm{T}}$ and $\mathbf{Y}_{N\times 1} = [y_0, y_1, \ldots, y_{N-1}]^{\mathrm{T}}$ are input and output data vectors, respectively. Symbol "T" denotes the matrix transpose operation, and

$$c_{k,n} = \sqrt{\frac{2}{N-1}} \varepsilon_n \varepsilon_k \cos\left(\frac{\pi nk}{N-1}\right). \tag{3}$$

In the case of DCT-I $\mathbf{C}_N = \mathbf{C}_N^{\mathrm{T}}$. Based on that general considerations, we can describe the entries of the DCT matrix in the following way:

$$\mathbf{C}_N = \begin{bmatrix} c_{0,0} & c_{0,1} & \cdots & c_{0,N-1} \\ c_{1,0} & c_{1,1} & \cdots & c_{1,N-1} \\ \vdots & \vdots & \ddots & \vdots \\ c_{N-1,0} & c_{N-1,1} & \cdots & c_{N-1,N-1}. \end{bmatrix}.$$

The entries of this matrix are real numbers and their values depend on both the indexes k, n and the number N. However, it will be more convenient for us to denote the numerical values of the matrix $\mathbf{C}_N$ entries by means of the letters of the ordinary Latin alphabet $\{a_N, b_N, c_N, \ldots, z_N\}$. In this case, the subscript N will indicate the size of the DCT matrix. This will simplify the identification of structural features of the matrix and the presence in it of compositions of the same values of the entries.

3. Small-Size Algorithms for the DCT-I

3.1. Algorithm for the 2-Point DCT-I

Let $\mathbf{X}_{2\times 1} = [x_0, x_1]^{\mathrm{T}}$ and $\mathbf{Y}_{2\times 1} = [y_0, y_1]^{\mathrm{T}}$ be 2-element input and output data vectors. The problem is to calculate the product:

$$\mathbf{Y}_{2\times 1} = \mathbf{C}_2 \mathbf{X}_{2\times 1}, \tag{4}$$

where

$$\mathbf{C}_2 = \begin{bmatrix} a_2 & a_2 \\ a_2 & -a_2 \end{bmatrix}, a_2 = \frac{\sqrt{2}}{2}.$$

Direct computation of (4) requires four multiplications and two additions. Because every vector element needs to be multiplied by the same factor it is possible to perform the additions first and then perform the multiplications.

Knowing that, the rationalized computational procedure for computing the 2-point DCT-I can be described in the following form:

$$\mathbf{Y}_{2\times 1} = \mathbf{D}_2\mathbf{H}_2\mathbf{X}_{2\times 1}, \tag{5}$$

where

$$\mathbf{H}_2 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}, \mathbf{D}_2 = \mathrm{diag}(s_0^{(2)}, s_1^{(2)}), s_0^{(2)} = s_1^{(2)} = a_2.$$

As shown in (5), the 2-point DCT-I can be calculated using only two multiplications and two additions.

The same algorithm is represented as a data flow graph in Figure 1. In this paper all of the data flow graphs represent data flow from left to right. Straight lines denote operations of data transfer (data paths). Multiplications are shown as circles with a number inside denoting the factor by which the data should be multiplied. Points where multiple lines end denote summation nodes. Additionally dashed lines visualise data paths changing the sign of a number (these data paths multiply a number by a factor of -1). We use the usual lines without arrows on purpose, so as not to clutter the graphs.

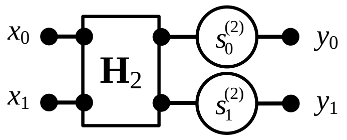

Figure 1. Signal flow graph of the algorithm for computing the 2-point DCT-I.

3.2. *Algorithm for the 3-Point DCT-I*

Let $\mathbf{X}_{3\times 1} = [x_0, x_1, x_2]^{\mathrm{T}}$ and $\mathbf{Y}_{3\times 1} = [y_0, y_1, y_2]^{\mathrm{T}}$ be 3-element input and output data vectors. The 3-point DCT-I can be represented as:

$$\mathbf{Y}_{3\times 1} = \mathbf{C}_3\mathbf{X}_{3\times 1}, \tag{6}$$

where

$$\mathbf{C}_3 = \begin{bmatrix} a_3 & b_3 & a_3 \\ b_3 & 0 & -b_3 \\ a_3 & -b_3 & a_3 \end{bmatrix}, a_3 = \frac{1}{2}, b_3 = \frac{\sqrt{2}}{2}.$$

The $\mathbf{C}_3$ matrix can be described as a sum of two matrices:

$$\begin{bmatrix} a_3 & b_3 & a_3 \\ b_3 & 0 & -b_3 \\ a_3 & -b_3 & a_3 \end{bmatrix} = \mathbf{C}_3^{(1)} + \mathbf{C}_3^{(2)},$$

where

$$\mathbf{C}_3^{(1)} = \begin{bmatrix} a_3 & 0 & a_3 \\ 0 & 0 & 0 \\ a_3 & 0 & a_3 \end{bmatrix}, \mathbf{C}_3^{(2)} = \begin{bmatrix} 0 & b_3 & 0 \\ b_3 & 0 & -b_3 \\ 0 & -b_3 & 0 \end{bmatrix}.$$

The $\mathbf{C}_3^{(1)}$ matrix can be reduced to a 2×2 matrix.

$$\mathbf{C}_3^{(1)} \rightarrow \mathbf{C}_2^{(1)} = \begin{bmatrix} a_3 & a_3 \\ a_3 & a_3. \end{bmatrix}$$

Multiplication by this matrix can be preformed using only one multiplication and one addition using the following formula:

$$\mathbf{Y}_{2\times 1}^{(1)} = \mathbf{1}_{2\times 1} a_3 \mathbf{1}_{1\times 2} \mathbf{X}_{2\times 1}^{(1)}$$

where

$$\mathbf{X}_{2\times 1}^{(1)} = [x_0, x_2]^{\mathrm{T}}, \mathbf{1}_{1\times 2} = [1, 1], \mathbf{1}_{2\times 1} = [1, 1]^{\mathrm{T}}, \mathbf{Y}_{2\times 1}^{(1)} = [y_0, y_2]^{\mathrm{T}}.$$

It is also possible to reduce the number of multiplications in the $\mathbf{C}_3^{(2)}$ matrix. In this case the addition from the second row can be performed before multiplication. So, x_1 can be multiplied by b_3 first and then it can be replicated while changing the sign of the replicated number to reproduce row numbers 1 and 3.

Taking into account the transformations made, the rationalized computational procedure for the 3-point DCT-I can be written in the following form:

$$\mathbf{Y}_{3\times 1} = \mathbf{W}_3^{(2)}\mathbf{D}_3\mathbf{W}_3^{(1)}\mathbf{X}_{3\times 1}, \tag{7}$$

where

$$\mathbf{W}_3^{(1)} = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & -1 \end{bmatrix}, \mathbf{D}_3 = \mathrm{diag}(s_0^{(3)}, s_1^{(3)}, s_2^{(3)}), \mathbf{W}_3^{(2)} = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & -1 & 0 \end{bmatrix}, s_0^{(3)} = a_3, s_1^{(3)} = s_2^{(3)} = b_3.$$

As you can see, in this and some other cases, the developed algorithms contain multiplications by $\frac{1}{2}$. This operation is reduced to the usual shift to the right by one position. Due to the ease of implementation, these operations are usually not taken into account when estimating computational complexity. Therefore, the 3-point DCT-I can be calculated using only two multiplications and four additions. Figure 2 represents this algorithm in the form of a data flow graph.

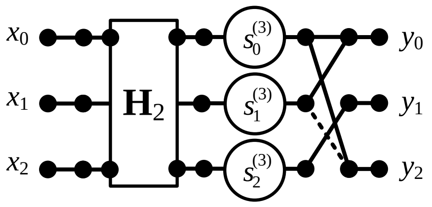

Figure 2. Signal flow graph of the algorithm for computing the 3-point DCT-I.

3.3. Algorithm for the 4-Point DCT-I

Let $\mathbf{X}_{4\times 1} = [x_0, x_1, x_2, x_3]^{\mathrm{T}}$ and $\mathbf{Y}_{4\times 1} = [y_0, y_1, y_2, y_3]^{\mathrm{T}}$ be 2-element input and output data vectors. The 4-point DCT-I can be represented as:

$$\mathbf{Y}_{4\times 1} = \mathbf{C}_4\mathbf{X}_{4\times 1}, \tag{8}$$

where

$$\mathbf{C}_4 = \begin{bmatrix} a_4 & b_4 & b_4 & a_4 \\ b_4 & a_4 & -a_4 & -b_4 \\ b_4 & -a_4 & -a_4 & b_4 \\ a_4 & -b_4 & b_4 & -a_4 \end{bmatrix}, a_4 = \frac{\sqrt{6}}{6}, b_4 = \frac{\sqrt{3}}{3}.$$

In the $\mathbf{C}_4$ the optimized version of the algorithm is not visible at a first glance. What we can do is change the order of columns and rows of the matrix while also permuting the corresponding elements in the input and output vectors. We chose to swap rows with number 1 and number 3 and also columns with numbers 1 and 3. As a result, we get the following matrix:

$$\tilde{\mathbf{C}}_4 = \left[\begin{array}{cc:cc} a_4 & a_4 & b_4 & b_4 \\ a_4 & -a_4 & b_4 & -b_4 \\ \hdashline b_4 & b_4 & -a_4 & -a_4 \\ b_4 & -b_4 & -a_4 & a_4 \end{array}\right] = \left[\begin{array}{c:c} \mathbf{A}_2^{(1)} & \mathbf{B}_2^{(1)} \\ \hdashline \mathbf{B}_2^{(1)} & -\mathbf{A}_2^{(1)}. \end{array}\right]$$

Because of the structure it can be computed using the following procedure [34]:

$$\tilde{\mathbf{C}}_4 = (\mathbf{T}_{2\times 3}^{(1)} \otimes \mathbf{I}_2)[(\mathbf{A}_2^{(1)} - \mathbf{B}_2^{(1)}) \oplus (-\mathbf{A}_2^{(1)} - \mathbf{B}_2^{(1)}) \oplus \mathbf{B}_2^{(1)}](\mathbf{T}_{3\times 2}^{(1)} \otimes \mathbf{I}_2),$$

where

$$\mathbf{T}_{3\times2}=\begin{bmatrix}1&0\\0&1\\1&1\end{bmatrix},\mathbf{T}_{2\times3}=\begin{bmatrix}1&0&1\\0&1&1\end{bmatrix}.$$

In this article the "$\otimes$" and "$\oplus$" symbols are used to represent the Kronecker product and direct sum of two matrices respectively [35,36]. Such factorization allows us to reduce the number of the multiplications by a factor of $\frac{3}{4}$. Both $\mathbf{A}_2^{(1)}$ and $\mathbf{B}_2^{(1)}$ share similar structures, which is a Hadamard matrix of order two multiplied by a scalar. Because of that it is possible to further reduce the number of multiplications times two by first using Hadamard matrix and later multiplying by proper scalars.

Knowing all of that, the rationalized computational procedure for computing the 4-point DCT-I can be described in the following form:

$$\mathbf{Y}_{4\times1}=\mathbf{P}_4\mathbf{A}_{4\times6}\mathbf{D}_6\mathbf{W}_6\mathbf{A}_{6\times4}\mathbf{P}_4\mathbf{X}_{4\times1} \tag{9}$$

where

$$\mathbf{P}_4=\begin{bmatrix}1&0&0&0\\0&0&0&1\\0&0&1&0\\0&1&0&0\end{bmatrix},\mathbf{A}_{6\times4}=\begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&0&1\\1&0&1&0\\0&1&0&1\end{bmatrix},\mathbf{W}_6=\mathbf{I}_3\otimes\mathbf{H}_2,\mathbf{D}_6=\mathrm{diag}\,(s_0^{(4)},s_1^{(4)},s_2^{(4)},\ldots,s_5^{(4)}),$$

$$s_0^{(4)}=s_1^{(4)}=a_4-b_4,s_2^{(4)}=s_3^{(4)}=-a_4-b_4,s_4^{(4)}=s_5^{(4)}=b_4,\mathbf{A}_{4\times6}=\begin{bmatrix}1&0&0&0&1&0\\0&1&0&0&0&1\\0&0&1&0&1&0\\0&0&0&1&0&1.\end{bmatrix}$$

As shown in Figure 3, the 4-point DCT-I can be computed using only six multiplications and 12 additions.

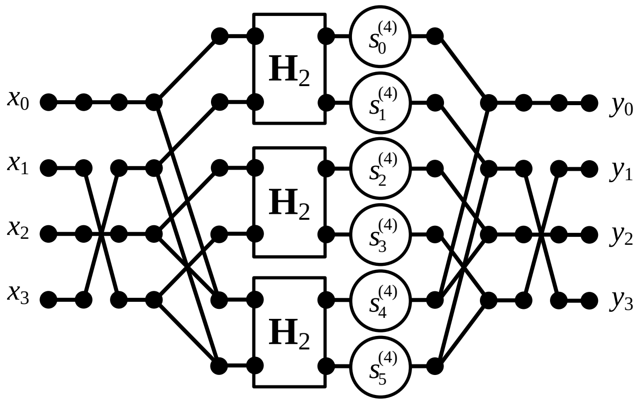

Figure 3. Signal flow graph of the algorithm for computing the 4-point DCT-I.

3.4. Algorithm for the 5-Point DCT-I

Let $\mathbf{X}_{5\times1}=[x_0,x_1,x_2,x_3,x_4]^{\mathrm{T}}$ and $\mathbf{Y}_{5\times1}=[y_0,y_1,y_2,y_3,y_4]^{\mathrm{T}}$ be 5-element input and output data vectors. The 5-point DCT-I can be represented as:

$$\mathbf{Y}_{5\times1}=\mathbf{C}_5\mathbf{X}_{5\times1}, \tag{10}$$

where

$$\mathbf{C}_5 = \begin{bmatrix} a_5 & b_5 & b_5 & b_5 & a_5 \\ b_5 & b_5 & 0 & -b_5 & -b_5 \\ b_5 & 0 & c_5 & 0 & b_5 \\ b_5 & -b_5 & 0 & b_5 & -b_5 \\ a_5 & -b_5 & b_5 & -b_5 & a_5 \end{bmatrix}, a_5 = \frac{\sqrt{2}}{4}, b_5 = \frac{1}{2}, c_5 = -\frac{\sqrt{2}}{2}.$$

In this matrix it is also worth changing the order of columns and rows and reordering them according to vector elements. It is also easier to fit one of the patterns after changing some of the signs. After swapping column 2 with column 5, row 1 with row 4 and inverting signs of x_3 and y_3 the matrix looks the following way:

$$\mathbf{C}_5 = \left[\begin{array}{cc:c:cc} -b_5 & b_5 & 0 & b_5 & b_5 \\ b_5 & -b_5 & 0 & b_5 & b_5 \\ \hdashline b_5 & b_5 & c_5 & 0 & 0 \\ a_5 & a_5 & b_5 & -b_5 & b_5 \\ a_5 & a_5 & b_5 & b_5 & -b_5. \end{array}\right]$$

This matrix can be split into three matrices for applying corresponding rationalized procedures:

$$\mathbf{A}_2^{(2)} = \begin{bmatrix} -b_5 & b_5 \\ b_5 & -b_5 \end{bmatrix}, \mathbf{B}_2^{(2)} = \begin{bmatrix} b_5 & b_5 \\ b_5 & b_5 \end{bmatrix}, \mathbf{A}_{3\times 5} = \left[\begin{array}{cc:c:cc} b_5 & b_5 & c_5 & 0 & 0 \\ \cdashline{4-5} a_5 & a_5 & b_5 & -b_5 & b_5 \\ a_5 & a_5 & b_5 & b_5 & -b_5 \end{array}\right].$$

For the $\mathbf{A}_2^{(2)}$ and $\mathbf{A}_2^{(2)}$ matrices there are already optimised formulas. The $\mathbf{A}_{3\times 5}$ requires an individual approach. The left part of this matrix can be reduced to a single addition and two multiplications. The first step is to add x_1 and x_2. Then the same value can be used twice and multiplied by b_5 and a_5 and added to corresponding rows. In this matrix it is worth calculating the multiplications from column 3 separately. The right part containing $\mathbf{A}_2^{(2)}$ matrix can also be computed using already existing procedures.

After applying all of this, the rationalized computational procedure for computing the 5-point DCT-I can be described in the following form:

$$\mathbf{Y}_{5\times 1} = \mathbf{P}_5^{(2)}\mathbf{W}_5^{(2)}\mathbf{A}_{5\times 7}\mathbf{D}_7\mathbf{M}_{7\times 5}\mathbf{W}_5^{(1)}\mathbf{P}_5^{(1)}\mathbf{X}_{5\times 1}, \tag{11}$$

where

$$\mathbf{P}_5^{(1)} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix}, \mathbf{W}_5^{(1)} = \begin{bmatrix} 1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 1 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \end{bmatrix}, \mathbf{M}_{7\times 5} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix},$$

$$\mathbf{D}_7 = \mathrm{diag}(s_0^{(5)}, s_1^{(5)}, s_2^{(5)}, \ldots, s_6^{(5)}), s_0^{(5)} = s_1^{(5)} = s_2^{(5)} = s_5^{(5)} = s_6^{(5)} = b_5 = \frac{1}{2}, s_3 = a_5, s_4 = c_5,$$

$$\mathbf{A}_{5\times 7} = \begin{bmatrix} 1 & -1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}, \mathbf{W}_5^{(2)} = \mathbf{I}_3 \oplus \mathbf{H}_2, \mathbf{P}_5^{(2)} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

As shown in (11) the 5-point DCT-I can be calculated using only two multiplications and 10 additions. Figure 4 represents this algorithm in the form of a data flow graph.

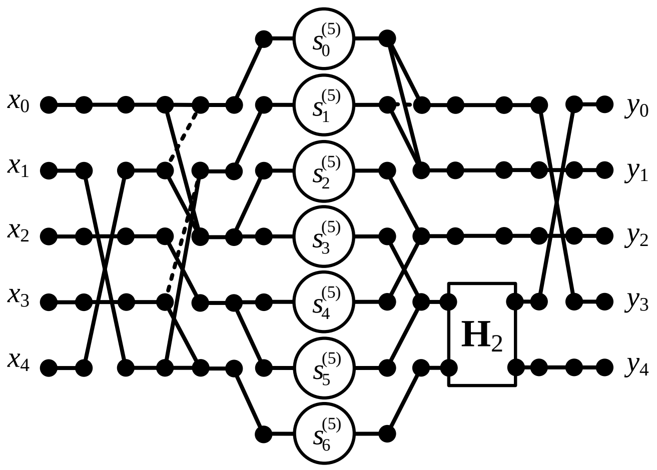

Figure 4. Signal flow graph of the algorithm for computing the 5-point DCT-I.

3.5. Algorithm for the 6-Point DCT-I

Let $\mathbf{X}_{6\times1} = [x_0, x_1, x_2, x_3, x_4, x_5]^{\mathrm{T}}$ and $\mathbf{Y}_{6\times1} = [y_0, y_1, y_2, y_3, y_4, y_5]^{\mathrm{T}}$ be 6-element input and output data vectors. The 6-point DCT-I can be represented as:

$$\mathbf{Y}_{6\times1} = \mathbf{C}_6\mathbf{X}_{6\times1}, \tag{12}$$

where

$$\mathbf{C}_6 = \begin{bmatrix} a_6 & b_6 & b_6 & b_6 & b_6 & a_6 \\ b_6 & c_6 & d_6 & -d_6 & -c_6 & -b_6 \\ b_6 & d_6 & -c_6 & -c_6 & d_6 & b_6 \\ b_6 & -d_6 & -c_6 & c_6 & d_6 & -b_6 \\ b_6 & -c_6 & d_6 & d_6 & -c_6 & b_6 \\ a_6 & -b_6 & b_6 & -b_6 & b_6 & -a_6 \end{bmatrix}, a_6 = \frac{\sqrt{10}}{10}, b_6 = \frac{\sqrt{5}}{5},$$

$$c_6 = \frac{\sqrt{10}}{5}\cos\frac{\pi}{5} \approx 0.51167, d_6 = \frac{\sqrt{10}}{5}\cos\frac{2\pi}{5} \approx 0.19544.$$

Before trying to find any way to optimize, it is worth changing the order of columns and rows. At first, we begin by swapping columns 2 with 6, 4 with 5, and rows 2 with 6 and 4 with 5. After this operation the matrix looks the following way:

$$\mathbf{C}_6^{(1)} = \left[\begin{array}{cc:cccc} a_6 & a_6 & b_6 & b_6 & b_6 & b_6 \\ a_6 & -a_6 & b_6 & b_6 & -b_6 & -b_6 \\ \hdashline b_6 & b_6 & -c_6 & d_6 & -c_6 & d_6 \\ b_6 & b_6 & d_6 & -c_6 & d_6 & -c_6 \\ b_6 & -b_6 & -c_6 & d_6 & c_6 & -d_6 \\ b_6 & -b_6 & d_6 & -c_6 & -d_6 & c_6 \end{array}\right] = \left[\begin{array}{c:c} \mathbf{A}_2^{(3)} & \mathbf{A}_{2\times4} \\ \hdashline \mathbf{A}_{4\times2} & \mathbf{A}_4^{(1)} \end{array}\right].$$

Similar to algorithms for $N = 2, 3, 4, 5$, the $\mathbf{A}_2^{(3)}$ matrix can be calculated by first performing the additions (multiplying by an order 2 Hadamard matrix) and later performing only two multiplications. The $\mathbf{A}_{2\times4}$ matrix can be optimized by first calculating $x_2 + x_3$ and $x_4 + x_5$, multiplying both expressions by b_6, and as the matrix pattern suggests an order 2 Hadamard matrix to put everything together. The $\mathbf{A}_{4\times2}$ matrix consists of two parts. Both of these halves can be reduced to singular multiplications by first computing $x_1 + x_2$ and $x_1 - x_2$ respectively and multiplying the results by b_6. These additions are not required to be calculated, because the additions from $\mathbf{A}_2^{(3)}$ can be reused. The $\mathbf{A}_4^{(1)}$ is more complex than previous matrices. In this case, the matrix has the following structure:

$$\mathbf{A}_4^{(1)} = \left[\begin{array}{cc:cc} -c_6 & d_6 & -c_6 & d_6 \\ d_6 & -c_6 & d_6 & -c_6 \\ \hdashline -c_6 & d_6 & c_6 & -d_6 \\ d_6 & -c_6 & -d_6 & c_6 \end{array}\right], = \left[\begin{array}{c:c} \mathbf{A}_2^{(4)} & \mathbf{A}_2^{(4)} \\ \hdashline \mathbf{A}_2^{(4)} & -\mathbf{A}_2^{(4)} \end{array}\right].$$

It is noticeable that this structure makes it possible to reduce the number of multiplications at least by a factor of 2. The multiplications can be performed for a single vertical half of matrix and these values can be reused in the second half. The right half requires the invertion of the signs of the results when reusing the results. In conclusion, only two matrix multiplications by $\mathbf{A}_2^{(4)}$ are required instead of four. It is also possible to reduce the number of multiplications in a single $\mathbf{A}_2^{(4)}$ matrix. To do so we can apply one of the templates of the matrix structures [34]. In this case the procedure for $\mathbf{A}_2^{(4)}$ would have the following form:

$$\mathbf{Y}_{2\times1} = \mathbf{H}_2 \frac{1}{2}\,\mathrm{diag}(-c_6 + d_6, -c_6 - d_6)\mathbf{H}_2\mathbf{X}_{2\times1}. \tag{13}$$

Knowing all of that, the rationalized computational procedure for computing the 6-point DCT-I can be described in the following form:

$$\mathbf{Y}_{6\times1} = \mathbf{P}_6\mathbf{A}_{6\times10}\mathbf{W}_{10}^{(2)}\mathbf{W}_{10}^{(1)}\mathbf{D}_{10}\mathbf{M}_{10\times6}\mathbf{W}_6^{(1)}\mathbf{P}_6\mathbf{X}_{6\times1}, \tag{14}$$

where

$$\mathbf{P}_6 = \begin{bmatrix} 1&0&0&0&0&0\\0&0&0&0&0&1\\0&0&1&0&0&0\\0&0&0&0&1&0\\0&0&0&1&0&0\\0&1&0&0&0&0 \end{bmatrix}, \mathbf{W}_6^{(1)} = \mathbf{I}_3 \otimes \mathbf{H}_2, \mathbf{M}_{10\times6} = \begin{bmatrix} 1&0&0&0&0&0\\1&0&0&0&0&0\\0&1&0&0&0&0\\0&1&0&0&0&0\\0&0&1&0&0&0\\0&0&0&0&1&0\\0&0&1&0&0&0\\0&0&0&1&0&0\\0&0&0&0&1&0\\0&0&0&0&0&1 \end{bmatrix}$$

$$\mathbf{D}_{10} = \mathrm{diag}(s_0, s_1, s_2, \ldots, s_9), s_0 = s_2 = a_6, s_1 = s_3 = s_4 = s_5 = b_6,$$

$$s_6 = s_8 = \frac{-c_6 + d_6}{2}, s_7 = s_9 = \frac{-c_6 - d_6}{2}, \mathbf{W}_{10}^{(1)} = \mathbf{I}_4 \oplus (\mathbf{I}_3 \otimes \mathbf{H}_2), \mathbf{W}_{10}^{(2)} = \mathbf{I}_6 \oplus (\mathbf{H}_2 \otimes \mathbf{I}_2),$$

$$\mathbf{A}_{6\times10} = \begin{bmatrix} 1&0&0&0&1&0&0&0&0&0\\0&0&1&0&0&1&0&0&0&0\\0&1&0&0&0&0&1&0&0&0\\0&1&0&0&0&0&0&1&0&0\\0&0&0&1&0&0&0&0&1&0\\0&0&0&1&0&0&0&0&0&1 \end{bmatrix}.$$

As shown in (14), the 6-point DCT-I can be calculated using only 10 multiplications and 22 additions. Figure 5 represents this algorithm in the form of a data flow graph.

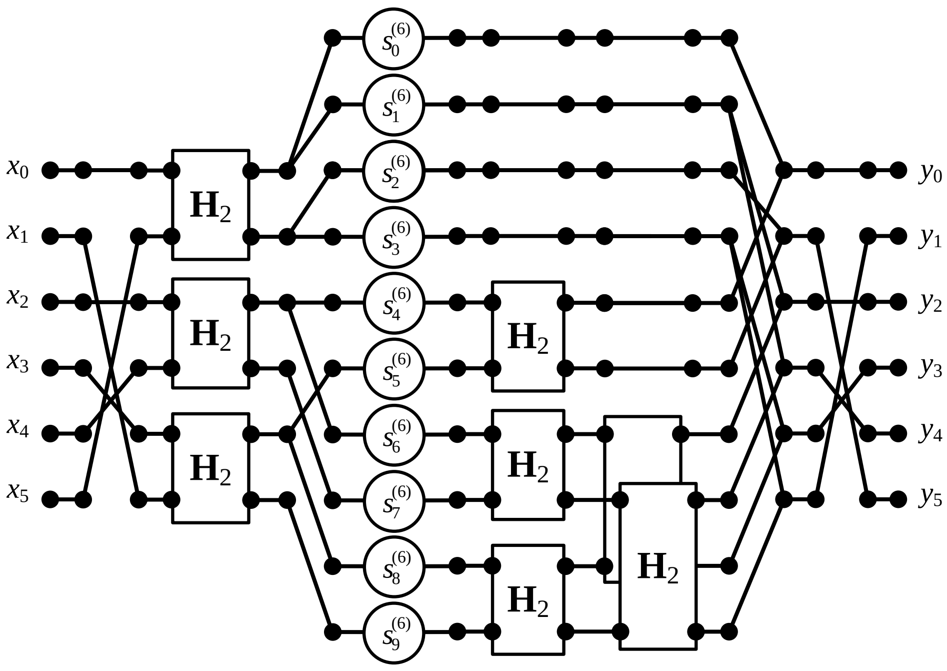

Figure 5. Signal flow graph of the algorithm for computing the 6-point DCT-I.

3.6. Algorithm for the 7-Point DCT-I

Let $\mathbf{X}_{7\times1} = [x_0, x_1, x_2, x_3, x_4, x_5, x_6]^{\mathrm{T}}$ and $\mathbf{Y}_{7\times1} = [y_0, y_1, y_2, y_3, y_4, y_5, y_6]^{\mathrm{T}}$ be 7-element input and output data vectors. The 7-point DCT-I can be represented as:

$$\mathbf{Y}_{7\times1} = \mathbf{C}_7\mathbf{X}_{7\times1}, \tag{15}$$

where

$$\mathbf{C}_7 = \begin{bmatrix} a_7 & b_7 & b_7 & b_7 & b_7 & b_7 & a_7 \\ b_7 & c_7 & a_7 & 0 & -a_7 & -c_7 & -b_7 \\ b_7 & a_7 & -a_7 & -d_7 & -a_7 & a_7 & b_7 \\ b_7 & 0 & -d_7 & 0 & d_7 & 0 & -b_7 \\ b_7 & -a_7 & -a_7 & d_7 & -a_7 & -a_7 & b_7 \\ b_7 & -c_7 & a_7 & 0 & -a_7 & c_7 & -b_7 \\ a_7 & -b_7 & b_7 & -b_7 & b_7 & -b_7 & a_7 \end{bmatrix}, a_7 = \frac{\sqrt{3}}{6}, b_7 = \frac{\sqrt{6}}{6}, c_7 = \frac{1}{2}, d_7 = -\frac{\sqrt{3}}{3}.$$

For better clarity we begin with changing the order of columns and rows in the $\mathbf{C}_7$ matrix. In this case it is worth swapping columns 2 with 7, 4 with 5 and rows 2 with 7 and 4 with 5. This leaves us with the following matrix:

$$\mathbf{C}_7^{(1)} = \left[\begin{array}{cccc:ccc} a_7 & a_7 & b_7 & b_7 & b_7 & b_7 & b_7 \\ a_7 & a_7 & b_7 & b_7 & -b_7 & -b_7 & -b_7 \\ b_7 & b_7 & -a_7 & -a_7 & -d_7 & a_7 & a_7 \\ b_7 & b_7 & -a_7 & -a_7 & d_7 & -a_7 & -a_7 \\ \hdashline b_7 & -b_7 & -d_7 & d_7 & 0 & 0 & 0 \\ b_7 & -b_7 & a_7 & -a_7 & 0 & c_7 & -c_7 \\ b_7 & -b_7 & a_7 & -a_7 & 0 & -c_7 & c_7 \end{array}\right] = \left[\begin{array}{c:c} \mathbf{A}_4 & \mathbf{A}_{4\times3}^{(1)} \\ \hdashline \mathbf{A}_{3\times4}^{(1)} & \mathbf{A}_3^{(1)} \end{array}\right].$$

Because the $\mathbf{A}_4$ matrix has a following structure:

$$\mathbf{A}_4 = \left[\begin{array}{cc:cc} a_7 & a_7 & b_7 & b_7 \\ a_7 & a_7 & b_7 & b_7 \\ \hdashline b_7 & b_7 & -a_7 & -a_7 \\ b_7 & b_7 & -a_7 & -a_7 \end{array}\right] = \left[\begin{array}{c:c} \mathbf{A}_2^{(5)} & \mathbf{B}_2 \\ \hdashline \mathbf{B}_2 & \mathbf{A}_2^{(5)}. \end{array}\right].$$

it is possible to apply the following procedure:

$$\mathbf{Y}_{4\times1} = (\mathbf{T}_{2\times3} \otimes \mathbf{I}_2)[(\mathbf{A}_2^{(5)} - \mathbf{B}_2) \oplus (-\mathbf{A}_2^{(5)} - \mathbf{B}_2) \oplus \mathbf{B}_2](\mathbf{T}_{3\times2} \otimes \mathbf{I}_2)\mathbf{X}_{4\times1}$$

where

$$\mathbf{T}_{3\times2} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix}, \mathbf{T}_{2\times3} = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}.$$

This way, the number of multiplications in $\mathbf{A}_4$ is reduced to only $\frac{3}{4}$ of the original multiplications. Additionally, matrices $(\mathbf{A}_2^{(5)} - \mathbf{B}_2)$, $(-\mathbf{A}_2^{(5)} - \mathbf{B}_2)$ and $\mathbf{B}_2$ have identical structures and these matrices require only a single multiplication as shown in one of the previous procedures. This means that multiplication by the $\mathbf{A}_4$ matrix can be reduced to only three multiplications.

The $\mathbf{A}_{4\times3}^{(1)}$ matrix can be split in the following way:

$$\mathbf{A}_{4\times3}^{(1)} = \left[\begin{array}{c:cc} b_7 & b_7 & b_7 \\ -b_7 & -b_7 & -b_7 \\ \hdashline -d_7 & a_7 & a_7 \\ d_7 & -a_7 & -a_7. \end{array}\right].$$

The upper half of this matrix can be calculated by adding all three of the input values and multiplying them once. The same result can be used for both of the rows by inverting the sign. The bottom half can be computed in a similar way, but the left part requires additional multiplication by d_7. Because this part requires the addition of the second and third input elements and the upper part requires the addition of all three arguments it is worth separating additions in two steps. Therefore, the number of multiplications in $\mathbf{A}_{4\times3}$ can also be reduced to three multiplications.

The $\mathbf{A}_3^{(1)}$ matrix contains only zeros in the first column and the first row and can be reduced to a 2×2 matrix:

$$\mathbf{A}_3^{(1)} \rightarrow \mathbf{A}_2^{(6)} = \begin{bmatrix} c_7 & -c_7 \\ -c_7 & c_7 \end{bmatrix}.$$

Number of multiplications in this matrix can be reduced to a single multiplication by performing the additions first and by knowing that both rows of this matrix are the same, but with an inverted sign.

The last part of the $\mathbf{C}_7$ matrix has the following structure:

$$\mathbf{A}_{3\times4} = \left[\begin{array}{cc:cc} b_7 & -b_7 & -d_7 & d_7 \\ b_7 & -b_7 & a_7 & -a_7 \\ b_7 & -b_7 & a_7 & -a_7. \end{array}\right].$$

The first step in this case is to calculate $x_1 - x - 2$ and $x_3 - x_4$. The left part of $\mathbf{A}_{3\times4}$ contains three identical rows so only a single multiplication of the first addition is required. In the right side it is important to note that $d_7 = 2a_7$ and it is possible to calculate this part by multiplying $x_3 - x_4$ only by a_7. To calculate the first row we can use the same result, reverse the sign and use a bitwise shift.

Knowing all of that, the rationalized computational procedure for computing the 7-point DCT-I can be described in the following form:

$$\mathbf{Y}_{7\times1} = \mathbf{P}_7\mathbf{A}_7\mathbf{A}_{7\times10}\mathbf{M}_{10\times9}\mathbf{D}_9\mathbf{A}_9\mathbf{W}_9\mathbf{A}_{9\times7}\mathbf{P}_7\mathbf{X}_{7\times1}, \tag{16}$$

where

$$\mathbf{P}_7 = \begin{bmatrix} 1&0&0&0&0&0&0\\ 0&0&0&0&0&0&1\\ 0&0&1&0&0&0&0\\ 0&0&0&0&1&0&0\\ 0&0&0&1&0&0&0\\ 0&0&0&0&0&1&0\\ 0&1&0&0&0&0&0 \end{bmatrix}, \mathbf{A}_{9\times7} = \begin{bmatrix} 1&0&0&0&0&0&0\\ 0&1&0&0&0&0&0\\ 0&0&1&0&0&0&0\\ 0&0&0&1&0&0&0\\ 1&0&1&0&0&0&0\\ 0&1&0&1&0&0&0\\ 0&0&0&0&1&0&0\\ 0&0&0&0&0&1&0\\ 0&0&0&0&0&0&1 \end{bmatrix},$$

$$\mathbf{W}_9 = \begin{bmatrix} 1&1&0&0&0&0&0&0&0\\ 0&0&1&1&0&0&0&0&0\\ 1&-1&0&0&0&0&0&0&0\\ 0&0&1&-1&0&0&0&0&0\\ 0&0&0&0&1&0&0&0&0\\ 0&0&0&0&0&1&0&0&0\\ 0&0&0&0&0&0&1&0&0\\ 0&0&0&0&0&0&0&1&1\\ 0&0&0&0&0&0&0&1&-1 \end{bmatrix}, \mathbf{A}_9 = \begin{bmatrix} 1&0&0&0&0&0&0&0&0\\ 0&1&0&0&0&0&0&0&0\\ 0&0&0&0&1&1&0&0&0\\ 0&0&0&0&0&0&1&1&0\\ 0&0&0&0&0&0&1&0&0\\ 0&0&0&0&0&0&0&1&0\\ 0&0&0&0&0&0&0&0&1\\ 0&0&1&0&0&0&0&0&0\\ 0&0&0&1&0&0&0&0&0 \end{bmatrix},$$

$$\mathbf{D}_9 = \mathrm{diag}(s_0, s_1, s_2, \ldots, s_8), s_0 = a_7 - b_7, s_1 = -a_7 - b_7, s_2 = s_3 = s_7 = b_7, s_4 = d_7,$$

$$s_5 = s_8 = a_7, s_6 = c_7, s_9 = 2, \mathbf{M}_{10\times9} = \mathbf{I}_8 \oplus \begin{bmatrix} 1\\ 1 \end{bmatrix},$$

$$\mathbf{A}_{7\times10} = \begin{bmatrix} 1&0&1&0&0&0&0&0&0&0\\ 0&1&1&0&0&0&0&0&0&0\\ 0&0&0&1&0&0&0&0&0&0\\ 0&0&0&0&1&-1&0&0&0&0\\ 0&0&0&0&0&0&1&0&0&0\\ 0&0&0&0&0&0&0&1&0&-1\\ 0&0&0&0&0&0&0&1&1&0 \end{bmatrix}, \mathbf{A}_7 = \begin{bmatrix} 1&0&1&0&0&0&0\\ 1&0&-1&0&0&0&0\\ 0&1&0&-1&0&0&0\\ 0&1&0&1&0&0&0\\ 0&0&0&0&0&1&0\\ 0&0&0&0&1&0&1\\ 0&0&0&0&-1&0&1 \end{bmatrix}.$$

As shown in (16) the 7-point DCT-I can be calculated using only nine multiplications, 21 additions and a single bitwise shift. Figure 6 represents this algorithm in the form of a data flow graph.

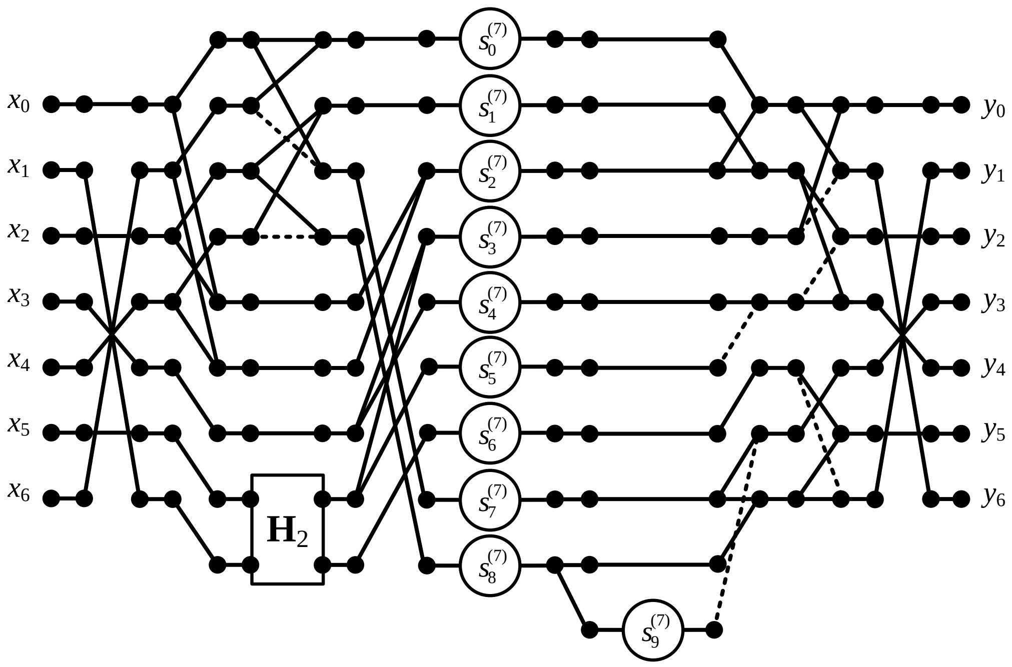

Figure 6. Signal flow graph of the algorithm for computing the 7-point DCT-I.

3.7. Algorithm for the 8-Point DCT-I

Let $\mathbf{X}_{8\times1} = [x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7]^{\mathrm{T}}$ and $\mathbf{Y}_{8\times1} = [y_0, y_1, y_2, y_3, y_4, y_5, y_6, y_7]^{\mathrm{T}}$ be 8-element input and output data vectors. The 8-point DCT-I can be represented as:

$$\mathbf{Y}_{8\times1} = \mathbf{C}_8\mathbf{X}_{8\times1}, \tag{17}$$

where

$$\mathbf{C}_8 = \begin{bmatrix} a_8 & b_8 & b_8 & b_8 & b_8 & b_8 & b_8 & a_8 \\ b_8 & c_8 & d_8 & e_8 & -e_8 & -d_8 & -c_8 & -b_8 \\ b_8 & d_8 & -e_8 & -c_8 & -c_8 & -e_8 & d_8 & b_8 \\ b_8 & e_8 & -c_8 & -d_8 & d_8 & c_8 & -e_8 & -b_8 \\ b_8 & -e_8 & -c_8 & d_8 & d_8 & -c_8 & -e_8 & b_8 \\ b_8 & -d_8 & -e_8 & c_8 & -c_8 & e_8 & d_8 & -b_8 \\ b_8 & -c_8 & d_8 & -e_8 & -e_8 & d_8 & -c_8 & b_8 \\ a_8 & -b_8 & b_8 & -b_8 & b_8 & -b_8 & b_8 & -a_8 \end{bmatrix}, a_8 = \frac{\sqrt{14}}{14}, b_8 = \frac{\sqrt{7}}{7},$$

$$c_8 = \sqrt{\frac{2}{7}}\cos\frac{\pi}{7} \approx 0.481588, d_8 = \sqrt{\frac{2}{7}}\cos\frac{2\pi}{7} \approx 0.333269, e_8 = \sqrt{\frac{2}{7}}\cos\frac{3\pi}{7} \approx 0.118942,$$

The first step for finding the algorithm is to split the $\mathbf{C}_8$ matrix in the following way:

$$\mathbf{C}_8 = \left[\begin{array}{c:cccccc:c} a_8 & b_8 & b_8 & b_8 & b_8 & b_8 & b_8 & a_8 \\ \hdashline b_8 & c_8 & d_8 & e_8 & -e_8 & -d_8 & -c_8 & -b_8 \\ b_8 & d_8 & -e_8 & -c_8 & -c_8 & -e_8 & d_8 & b_8 \\ b_8 & e_8 & -c_8 & -d_8 & d_8 & c_8 & -e_8 & -b_8 \\ b_8 & -e_8 & -c_8 & d_8 & d_8 & -c_8 & -e_8 & b_8 \\ b_8 & -d_8 & -e_8 & c_8 & -c_8 & e_8 & d_8 & -b_8 \\ b_8 & -c_8 & d_8 & -e_8 & -e_8 & d_8 & -c_8 & b_8 \\ \hdashline a_8 & -b_8 & b_8 & -b_8 & b_8 & -b_8 & b_8 & -a_8 \end{array}\right] = \mathbf{A}_8^{(1)} + \mathbf{A}_8^{(2)},$$

where

$$\mathbf{A}_8^{(1)} = \begin{bmatrix} a_8 & b_8 & b_8 & b_8 & b_8 & b_8 & b_8 & a_8 \\ b_8 & 0 & 0 & 0 & 0 & 0 & 0 & -b_8 \\ b_8 & 0 & 0 & 0 & 0 & 0 & 0 & b_8 \\ b_8 & 0 & 0 & 0 & 0 & 0 & 0 & -b_8 \\ b_8 & 0 & 0 & 0 & 0 & 0 & 0 & b_8 \\ b_8 & 0 & 0 & 0 & 0 & 0 & 0 & -b_8 \\ b_8 & 0 & 0 & 0 & 0 & 0 & 0 & b_8 \\ a_8 & -b_8 & b_8 & -b_8 & b_8 & -b_8 & b_8 & -a_8 \end{bmatrix},$$

$$\mathbf{A}_8^{(2)} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & c_8 & d_8 & e_8 & -e_8 & -d_8 & -c_8 & 0 \\ 0 & d_8 & -e_8 & -c_8 & -c_8 & -e_8 & d_8 & 0 \\ 0 & e_8 & -c_8 & -d_8 & d_8 & c_8 & -e_8 & 0 \\ 0 & -e_8 & -c_8 & d_8 & d_8 & -c_8 & -e_8 & 0 \\ 0 & -d_8 & -e_8 & c_8 & -c_8 & e_8 & d_8 & 0 \\ 0 & -c_8 & d_8 & -e_8 & -e_8 & d_8 & -c_8 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0. \end{bmatrix}.$$

Multiplication by $\mathbf{A}_8^{(1)}$ can be optimised by separating the multiplications from additions and performing them before multiplications. Simple techniques like factoring out parentheses provide good results in this case and it is possible to reduce a multiplication by $\mathbf{A}_8^{(1)}$ to only six multiplications.

The $\mathbf{A}_8^{(2)}$ matrix contains only zeros on its borders and can be reduced to a 6×6 matrix:

$$\mathbf{A}_8^{(2)} \to \mathbf{A}_6 = \begin{bmatrix} c_8 & d_8 & e_8 & -e_8 & -d_8 & -c_8 \\ d_8 & -e_8 & -c_8 & -c_8 & -e_8 & d_8 \\ e_8 & -c_8 & -d_8 & d_8 & c_8 & -e_8 \\ -e_8 & -c_8 & d_8 & d_8 & -c_8 & -e_8 \\ -d_8 & -e_8 & c_8 & -c_8 & e_8 & d_8 \\ -c_8 & d_8 & -e_8 & -e_8 & d_8 & -c_8 \end{bmatrix}.$$

To find a way to reduce the number of multiplications in this matrix the first step is to change the order of columns in this matrix to: 1, 5, 3, 6, 2, 4 and the order of rows to: 3, 5, 1, 4, 2, 6 and invert the signs of the three last columns in the resulting matrix. After this operation, the matrix looks as follows:

$$\mathbf{A}_6^{(2)} = \begin{bmatrix} e_8 & c_8 & -d_8 & e_8 & c_8 & -d_8 \\ -d_8 & e_8 & c_8 & -d_8 & e_8 & c_8 \\ c_8 & -d_8 & e_8 & c_8 & -d_8 & e_8 \\ -e_8 & -c_8 & d_8 & e_8 & c_8 & -d_8 \\ d_8 & -e_8 & -c_8 & -d_8 & e_8 & c_8 \\ -c_8 & d_8 & -e_8 & c_8 & -d_8 & e_8 \end{bmatrix}.$$

The structure of this matrix can be described in the following way:

$$\mathbf{A}_6^{(2)} = \left[\begin{array}{ccc:ccc} e_8 & c_8 & -d_8 & e_8 & c_8 & -d_8 \\ -d_8 & e_8 & c_8 & -d_8 & e_8 & c_8 \\ c_8 & -d_8 & e_8 & c_8 & -d_8 & e_8 \\ \hdashline -e_8 & -c_8 & d_8 & e_8 & c_8 & -d_8 \\ d_8 & -e_8 & -c_8 & -d_8 & e_8 & c_8 \\ -c_8 & d_8 & -e_8 & c_8 & -d_8 & e_8 \end{array}\right] = \left[\begin{array}{c:c} \mathbf{A}_3^{(2)} & \mathbf{A}_3^{(2)} \\ \hdashline \mathbf{B}_3 & -\mathbf{B}_3 \; . \end{array}\right].$$

Because of that it is possible to apply the following formula [34]:

$$\mathbf{Y}_{6\times 1} = (\mathbf{A}_3^{(2)} \oplus \mathbf{B}_3)(\mathbf{H}_2 \otimes \mathbf{I}_3)\mathbf{X}_{6\times 1}.$$

This already reduces the number of multiplication by a factor of 2. The input vector is multiplied by two 3×3 matrices instead of a single 6×6 matrix. These smaller matrices share the same pattern and only all of the signs are inverted relative to the other matrix. This means that we can take the same approach for both of these matrices.

Because of the characteristic structure of $\mathbf{A}_3^{(2)}$, multiplication by this matrix can be calculated using a three-point circular convolution [37] which has the following form for $\mathbf{A}_3^{(2)}$:

$$\mathbf{A}_3^{(2)} = \mathbf{A}_3^{(4)}\mathbf{A}_{3\times4}\mathbf{D}_4^{(1)}\mathbf{A}_{4\times3}^{(2)}\mathbf{A}_3^{(3)},$$

where

$$\mathbf{A}_3^{(3)} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & -1 \\ 0 & 1 & -1 \end{bmatrix}, \mathbf{A}_{4\times3}^{(2)} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}, \mathbf{D}_4^{(1)} = \mathrm{diag}(s_0, s_1, s_2, s_3), s_0 = \frac{c-d+e}{3},$$

$$s_1 = -c+e, s_2 = -c-d, s_3 = \frac{-2c-d+e}{3}, \mathbf{A}_{3\times4} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & -1 \\ 0 & 0 & 1 & -1 \end{bmatrix}, \mathbf{A}_3^{(4)} = \begin{bmatrix} 1 & 1 & 0 \\ 1 & -1 & -1 \\ 1 & 0 & 1 \end{bmatrix}.$$

Knowing all of that, the rationalized computational procedure for computing the 8-point DCT-I can be described in the following form:

$$\mathbf{Y}_{8\times1} = \mathbf{A}_{8\times10}\mathbf{A}_{10}\mathbf{A}_{10\times14}\mathbf{D}_{14}\mathbf{M}_{14\times10}\mathbf{W}_{10}\mathbf{W}_{10\times14}\mathbf{M}_{14\times8}\mathbf{X}_{8\times1}, \tag{18}$$

where

$$\mathbf{M}_{14\times8} = \begin{bmatrix} \mathbf{P}_{6\times8} \\ \hdashline \mathbf{I}_8 \end{bmatrix} \mathbf{P}_{6\times8} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \end{bmatrix},$$

$$\mathbf{W}_{10\times14} = \mathbf{W}_6^{(2)} \oplus \mathbf{W}_{4\times8}, \mathbf{W}_{4\times8} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \end{bmatrix},$$

$$\mathbf{W}_6^{(2)} = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & -1 & 0 & 0 \\ 0 & 1 & 0 & 0 & -1 & 0 \\ 0 & 0 & 1 & 0 & 0 & -1 \end{bmatrix}, \mathbf{W}_{10} = \mathbf{I}_2 \otimes \mathbf{A}_3^{(3)} \oplus \mathbf{I}_4, \mathbf{A}_3^{(3)} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & -1 \\ 0 & 1 & -1 \end{bmatrix},$$

$$\mathbf{M}_{14\times10} = \mathbf{I}_2 \otimes \mathbf{A}_{4\times3}^{(2)} \oplus \mathbf{M}_{6\times4}, \mathbf{M}_{6\times4} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \mathbf{A}_{4\times3}^{(2)} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix},$$

$$\mathbf{D}_{14} = \mathrm{diag}(s_0^{(8)}, s_1^{(8)}, s_2^{(8)}, \ldots, s_{13}^{(8)}), s_0^{(8)} = \frac{c-d+e}{3}, s_1^{(8)} = -c+e, s_2^{(8)} = -c-d,$$

$$s_3^{(8)} = \frac{-2c-d+e}{3}, s_4^{(8)} = -s_0^{(8)}, s_5^{(8)} = -s_1^{(8)}, s_6^{(8)} = -s_2^{(8)}, s_7^{(8)} = -s_3^{(8)}, s_8^{(8)} = s_{11}^{(8)} = a_8,$$

$$s_9^{(8)} = s_{10}^{(8)} = s_{12}^{(8)} = s_{13}^{(8)} = b_8, \mathbf{A}_{10\times 14} = \mathbf{I}_2 \otimes \mathbf{A}_{3\times 4}^{(2)} \oplus \mathbf{A}_{4\times 6}, \mathbf{A}_{3\times 4}^{(2)} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & -1 \\ 0 & 0 & 1 & -1 \end{bmatrix},$$

$$\mathbf{A}_{4\times 6} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & -1 & 1 \end{bmatrix}, \mathbf{A}_{10} = \mathbf{I}_2 \otimes \mathbf{A}_3^{(4)} \oplus \mathbf{I}_4, \mathbf{A}_3^{(4)} = \begin{bmatrix} 1 & 1 & 0 \\ 1 & -1 & -1 \\ 1 & 0 & 1 \end{bmatrix}$$

$$\mathbf{A}_{8\times 10} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1. \end{bmatrix}$$

As shown in (18) the 8-point DCT-I can be calculated using only 14 multiplications and 43 additions. Figure 7 represents this algorithm in the form of a data flow graph.

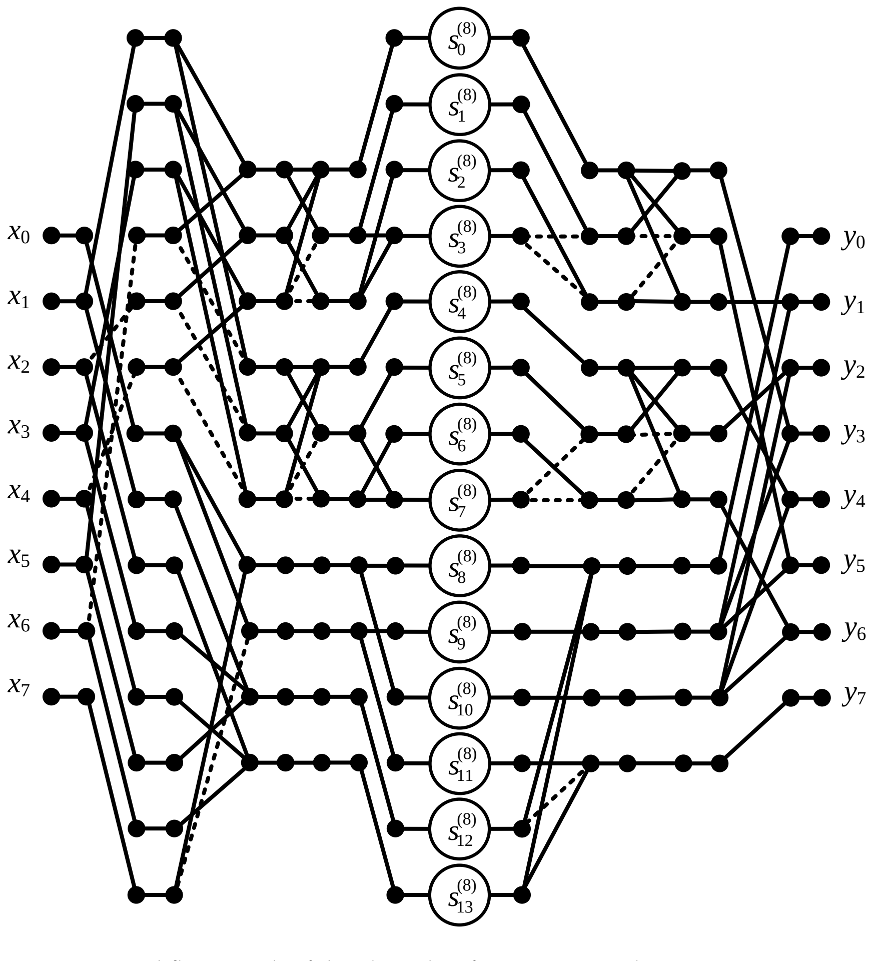

Figure 7. Signal flow graph of the algorithm for computing the 8-point DCT-I.

4. Computation Complexity

Despite the fact that we noted the number of arithmetic operations spent during the implementation for each algorithm separately, in this section we provide a summary table. Table 1 shows estimates of the number of arithmetic operations for short length DCT-I algorithms. The penultimate column of Table 1 shows the percentage reduction in the number of multiplications, while the last column shows the percentage reduction in the number of additions.

Table 1. Arithmetical complexities of naïve implementation and proposed solutions.

Length N	**Numbers of Arithmetic Operations in DCT-I Algorithms**					
	Naïve Implementation		**Proposed Solutions**		**Percentage Estimate**	
	"×"	"+"	"×"	"+"	"×"	"+"
2	4	2	2	2	50%	0%
3	9	6	2	4	78%	33%
4	16	12	6	12	63%	0%
5	25	20	2	10	92%	50%
6	36	30	10	22	72%	27%
7	49	42	9	21	82%	50%
8	64	56	14	43	78%	23%

5. Conclusions

This article presents a set of small-size type I discrete cosine transform algorithms with a reduced number of multiplications. This fact suggests that with the correct hardware implementation of the developed algorithms in the form of full-fledged ASIC modules, these modules will take up less space and consume less energy. As a result, the entire system in which these modules will be used as building blocks will have minimal dimensions and low power consumption. This approach is especially important when dealing with battery-powered devices. While modern stationary data processing systems have sufficient processing power due to the parallelization of calculations, the process of designing battery-powered mobile airborne systems contains many conflicting factors that prevent maximum performance. The parallelization of computing, traditionally used to achieve high data processing speed, leads to an increase in hardware costs and, as a result, to an increase in the size, weight, and power consumption of the entire system. Therefore, we need solutions that, on the one hand, maximize the use of parallel computing, and on the other hand, minimize the hardware implementation costs. With proper implementation, the algorithms proposed in the article can provide high technical characteristics. In the future, we plan to expand the set of proposed algorithmic solutions, as well as implement and present the algorithms in the form of IP cores. These issues will be consistently reflected in the authors' subsequent publications.

Author Contributions: Conceptualization, M.K. and A.C.; methodology, A.C.; software, M.K.; validation, M.K. and A.C.; formal analysis, M.K. and A.C.; investigation, M.K. and A.C.; resources, A.C.; data curation, M.K.; writing—original draft preparation, M.K.; writing—review and editing, A.C.; visualization, M.K.; supervision, A.C.; project administration, M.K.; funding acquisition, M.K. and A.C. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Acknowledgments: This research was supported by ZUT Highfliers School/Szkoła Orłów ZUT/ project co-ordinated by Piotr Sulikowski, within the framework of the program of the Minister of Education and Science, Poland/Grant No. MNiSW/2019/391/DIR/KH, POWR.03.01.00-00-P015/18/, co-financed by the European Social Fund, the amount of financing PLN 1,704,201,66.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Ahmed, N.; Natarajan, T.; Rao, K. Discrete Cosine Transform. *IEEE Trans. Comput.* **1974**, *C-23*, 90–93. [CrossRef]
2. Ahmed, N.; Rao, K.R. *Orthogonal Transforms for Digital Signal Processing*; Springer: Berlin/Heidelberg, Germany, 1975. [CrossRef]
3. Rao, K.; Yip, P. *Discrete Cosine Transform: Algorithms, Advantages, Applications*; Academic Press: Cambridge, MA, USA, 1990. [CrossRef]
4. Britanak, V.; Yip, P.; Rao, K. *Discrete Cosine and Sine Transforms: General Properties, Fast Algorithms and Integer Approximations*; Academic Press: Cambridge, MA, USA, 2007. [CrossRef]
5. Ochoa-Domínguez, H.; Rao, K.R. *Discrete Cosine Transform*; CRC Press: Boca Raton, FL, USA, 2019. [CrossRef]
6. Elliott, D.F.; Rao, K.R. *Fast Transforms: Algorithms, Analyses, Applications*; Academic Press: Cambridge, MA, USA, 1983.
7. Chitprasert, B.; Rao, K.R. Discrete Cosine Transform Filtering. *Signal Process.* **1990**, *19*, 235–245. [CrossRef]
8. Armas Vega, E.A.; Sandoval Orozco, A.L.; García Villalba, L.J.; Hernandez-Castro, J. Digital Images Authentication Technique Based on DWT, DCT and Local Binary Patterns. *Sensors* **2018**, *18*, 3372. [CrossRef] [PubMed]
9. Krikor, L.; Baba, S.; Alnasiri, T.; Shaaban, Z. Image Encryption Using DCT and Stream Cipher. *Eur. J. Sci. Res.* **2009**, *32*, 45–57.
10. Yang, J.; Jin, T.; Xiao, C.; Huang, X. Compressed Sensing Radar Imaging: Fundamentals, Challenges, and Advances. *Sensors* **2019**, *19*, 3100. [CrossRef]
11. Lee, C.F.; Shen, J.J.; Chen, Z.R.; Agrawal, S. Self-Embedding Authentication Watermarking with Effective Tampered Location Detection and High-Quality Image Recovery. *Sensors* **2019**, *19*, 2267. [CrossRef] [PubMed]
12. Lu, W.; Chen, Z.; Li, L.; Cao, X.; Wei, J.; Xiong, N.; Li, J.; Dang, J. Watermarking Based on Compressive Sensing for Digital Speech Detection and Recovery. *Sensors* **2018**, *18*, 2390. [CrossRef] [PubMed]
13. Boukhechba, K.; Wu, H.; Bazine, R. DCT-Based Preprocessing Approach for ICA in Hyperspectral Data Analysis. *Sensors* **2018**, *18*, 1138. [CrossRef]
14. Xu, P.; Chen, B.; Xue, L.; Zhang, J.; Zhu, L. A Prediction-Based Spatial-Spectral Adaptive Hyperspectral Compressive Sensing Algorithm. *Sensors* **2018**, *18*, 3289. [CrossRef]
15. Agostini, L.; Silva, I.; Bampi, S. Pipelined fast 2D DCT architecture for JPEG image compression. In Proceedings of the Symposium on Integrated Circuits and Systems Design, Pirenopolis, Brazil, 10–15 September 2001; pp. 226–231. [CrossRef]
16. Dhandapani, V.; Seshasayanan, R. Area and power efficient DCT architecture for image compression. *J. Adv. Signal Process.* **2014**, *2014*, 1–9. [CrossRef]
17. Budagavi, M.; Fuldseth, A.; Bjøntegaard, G.; Sze, V.; Sadafale, M. Core Transform Design in the High Efficiency Video Coding (HEVC) Standard. *IEEE J. Sel. Top. Signal Process.* **2013**, *7*, 1029–1041. [CrossRef]
18. Yip, P.C.; Rao, K.R. High Efficiency Video Coding. ITU-T Rec. H.265 and ISO/IEC 23008-2 (HEVC). Standard, ITU-T and ISO/IEC. 2013.
19. Meher, P.; Park, S.Y.; Mohanty, B.; Lim, K.; Yeo, C. Efficient integer DCT architectures for HEVC. *IEEE Trans. Circuits Syst. Video Technol.* **2014**, *24*, 168–178. [CrossRef]
20. Kalali, E.; Mert, A.C.; Hamzaoglu, I. A computation and energy reduction technique for HEVC Discrete Cosine Transform. *IEEE Trans. Consum. Electron.* **2016**, *62*, 166–174. [CrossRef]
21. Pastuszak, G. Hardware architectures for the H.265/HEVC discrete cosine transform. *IET Image Process.* **2015**, *9*, 468–477. [CrossRef]
22. Pourazad, M.T.; Doutre, C.; Azimi, M.; Nasiopoulos, P. HEVC: The New Gold Standard for Video Compression: How Does HEVC Compare with H.264/AVC? *IEEE Consum. Electron. Mag.* **2012**, *1*, 36–46. [CrossRef]
23. Zhou, M.; Jiang, B.; Li, T.; Zhong, W.; Gao, X. DCT-based channel estimation techniques for LTE uplink. In Proceedings of the 2009 IEEE 20th International Symposium on Personal, Indoor and Mobile Radio Communications, Tokyo, Japan, 13–16 September 2009; pp. 1034–1038. [CrossRef]
24. Ali, M.; Islam, M.; Memon, M.; Asif, D.M.; Lin, F. Optimum DCT type-I based transceiver model and effective channel estimation for uplink NB-IoT system. *Phys. Commun.* **2021**, *48*, 101431. [CrossRef]
25. Domínguez-Jiménez, M.E.; Luengo, D.; Sansigre-Vidal, G. Estimation of Symmetric Channels for Discrete Cosine Transform Type-I Multicarrier Systems: A Compressed Sensing Approach. *Sci. World J.* **2015**, *2015*, 1–10. [CrossRef] [PubMed]
26. Domínguez-Jiménez, M.E.; Luengo, D.; Sansigre-Vidal, G.; Cruz-Roldán, F. A novel channel estimation scheme for multicarrier communications with the Type-I even discrete cosine transform. In Proceedings of the 2017 25th European Signal Processing Conference (EUSIPCO), Kos Island, Greece, 28 August–2 September 2017; pp. 2239–2243. [CrossRef]
27. Domínguez-Jiménez, M.E.; Luengo, D.; Sansigre-Vidal, G.; Cruz-Roldán, F. A Novel Scheme of Multicarrier Modulation With the Discrete Cosine Transform. *IEEE Trans. Wirel. Commun.* **2021**, *20*, 7992–8005. [CrossRef]
28. Cariow, A.; Makowska, M.; Strzelec, P. Small-Size FDCT/IDCT Algorithms with Reduced Multiplicative Complexity. *Radioelectron. Commun. Syst.* **2019**, *62*, 559–576. [CrossRef]
29. Cariow, A.; Lesiecki, Ł. Small-Size Algorithms for Type-IV Discrete Cosine Transform with Reduced Multiplicative Complexity. *Radioelectron. Commun. Syst.* **2020**, *63*, 465–487. [CrossRef]
30. Cariow, A.; Papliński, J.; Majorkowska-Mech, D. Some Structures of Parallel VLSI-Oriented Processing Units for Implementation of Small Size Discrete Fractional Fourier Transforms. *Electronics* **2019**, *8*, 509. [CrossRef]
31. Britanak, V. New universal rotation-based fast computational structures for an efficient implementation of the DCT-IV/DST-IV and analysis/synthesis MDCT/MDST filter banks. *Signal Process.* **2009**, *89*, 2213–2232. [CrossRef]

32. Britanak, V. New Recursive Fast Radix-2 Algorithm for the Modulated Complex Lapped Transform. *IEEE Trans. Signal Process.* **2012**, *60*, 6703–6708. [CrossRef]
33. Britanak, V.; Rao, R. Two-dimensional DCT/DST universal computational structure for 2*m* × 2*n* block sizes. *IEEE Trans. Signal Process.* **2000**, *48*, 3250–3255. [CrossRef]
34. Cariow, A. Strategies for the Synthesis of Fast Algorithms for the Computation of the Matrix-vector Products. *J. Signal Process. Theory Appl.* **2014**, *3*, 1–19. [CrossRef]
35. Regalia, P.A.; Sanjit, M.K. Kronecker Products, Unitary Matrices and Signal Processing Applications. *SIAM Rev.* **1989**, *31*, 586–613. [CrossRef]
36. Granata, J.; Conner, M.; Tolimieri, R. The tensor product: A mathematical programming language for FFTs and other fast DSP operations. *IEEE Signal Process. Mag.* **1992**, *9*, 40–48. [CrossRef]
37. Cariow, A.; Paplinski, J.P. Algorithmic Structures for Realizing Short-Length Circular Convolutions with Reduced Complexity. *Electronics* **2021**, *10*, 2800. [CrossRef]