Int. J. Appl. Math. Comput. Sci., 2026, Vol. 36, No. 1, 91–112
DOI: 10.61822/amcs-2026-0008

amcs

ALGORITHMS FOR A SMALL–SIZED TYPE II DISCRETE FOURIER TRANSFORM

JANUSZ PAPLIŃSKI [a,*], MARINA POLYAKOVA [b], ALEKSANDR CARIOW [a]

[a]Faculty of Computer Science and Information Technology
West Pomeranian University of Technology in Szczecin
Żołnierska 49, 71-210 Szczecin, Poland
e-mail: janusz.paplinski@zut.edu.pl

[b]Institute of Computer Systems
Odesa Polytechnic National University
1 Shevchenko Ave., Odesa, 65044, Ukraine

The paper is devoted to the development of algorithms for the odd-time discrete Fourier transform (type II DFT, or simply DFT-II). It presents efficient computational solutions related to the implementation of a small-sized DFT-II for input sequences of lengths 3, 4, 5, 6, 7, and 8. The derivation of each algorithm is described in detail, and computational complexity estimates are provided. The developed algorithms are implemented on field-programmable gate array (FPGA) platforms such as Spartan 3 and Spartan 6, demonstrating the advantages of their implementation in hardware environments where performance and resource utilization are critical. Key performance indicators such as the number of multiplications and additions as well as FPGA resource utilization are evaluated. The values of the maximum operating frequencies achieved are given. The results show a significant improvement in computational performance compared to the direct matrix-vector product, which proves the effectiveness of the obtained solutions.

Keywords: digital signal processing, DFT-II, resource-efficient algorithms.

1. Introduction

In digital signal processing, the discrete Fourier transform (DFT) is one of the most frequently used tools to find the spectrum of a time-domain signal (Bi and Zeng, 2012; Silva-García *et al.*, 2020). It is possible to shift DFT sampling in the time and/or frequency domain by some real shifts n_0 and k_0, respectively. As a result of such shifting, a generalized DFT (GDFT) is obtained (Yaroslavsky, 2014; Spałek, 2018). Most often, shifts of 0.5 (half a sample) are used. While the traditional DFT corresponds to a periodic signal in both time and frequency domains, $n_0 = 0.5$ produces a signal that is anti-periodic in the frequency domain and vice versa for $k_0 = 0.5$. If $n_0 = k_0 = 0.5$, then we obtain an odd-time odd-frequency DFT (Yaroslavsky, 2014; Oraintara, 2002).

In a one-dimensional transform, the amplitude spectrum of the DFT and GDFT is the same, whereas the phase spectrum differs. The phase spectrum is widely used in engineering applications (Dattoli *et al.*, 2017). The half-sample shift can be viewed as a modification within the GDFT family, where non-integer sampling intervals provide improved spectral resolution, particularly for signals with anti-periodic properties. These properties make the GDFT valuable in applications like fast computations (Mansour, 2006; Dai *et al.*, 2019), communication, code division multiple access (CDMA) and orthogonal frequency division multiplexing (OFDM) systems (Elshirkasi *et al.*, 2014; Li *et al.*, 2014; Almenar *et al.*, 2011; Patil *et al.*, 2012), filter bank design for subband processing systems (Wilbur *et al.*, 2004; Otunniyi and Myburgh, 2021), wave propagation (Tsitsas, 2010), audio systems (Dun and Liu, 2015; Belega *et al.*, 2016), or encoding (Mazrooei *et al.*, 2018). Hence, this paper is devoted to a DFT with a half-sample shift in time, which is also named an odd-time DFT or DFT-II, because the processed signal is expanded into complex exponentials with frequencies that are multiples of $\pi i/N$,

*Corresponding author

where N is the length of the input sequence, and i is an odd positive integer.

The number of arithmetic operations required for the direct computation of the GDFT is proportional to N. That is why there is a need to reduce the implementation cost and computational complexity of the GDFT and odd-time DFT in particular (Bi and Zeng, 2012). It is necessary to notice that the length of the input sequence significantly influences the design of fast algorithms for an odd-time DFT and GDFT in general. The case of large lengths of input data sequences is considered in the few existing papers related to designing fast GDFT algorithms (Bi and Zeng, 2012; Pei and Luo, 1996; Britanak and Rao, 1999; Bi and Chen, 1998; Zheng, 1996). However, small-sized GDFT algorithms are of special interest, since they can be regarded as typical modules in synthesizing more complex algorithms (Bi and Zeng, 2012; Saleh *et al.*, 2003). Then, designing fast odd-time DFT algorithms for small-sized input sequences is a relevant problem. Further, to select the approach for developing such fast algorithms, related papers are analyzed.

1.1. State-of-art of the problem. For most applications, efficient computation of the GDFT is crucial. However, directly calculating an N-point GDFT requires N^2 complex multiplications and $N(N-1)$ complex additions. This extremely slows down the speed of digital signal processing, especially in real-time applications.

To reduce the computational complexity of the GDFT for arbitrary time and frequency origins, in the works of Bi and Zeng (2012) as well as Bi and Chen (1998) the split-radix GDFT algorithm is exploited. The odd-time DFT is then decomposed into an odd-time DFT of length $N/2$ and an odd-time odd-frequency DFT of length $N/2$. The latter, in turn, can be decomposed into two $N/2$-point discrete cosine transforms (DCTs).

Britanak and Rao (1999) decompose the DFT-II into an $N/2$-point odd-time DCT and an $N/2$-point odd-time discrete sine transform, but the matrix of the latter exhibitis the reverse order in both rows and columns.

In the work of Zheng (1996), an odd-length GDFT is converted into a DFT using only permutation and sign changes of the input or/and output sequences. The same approach is also used to convert an odd-length GDFT to the discrete Hartley transform. Then the DFT or a discrete Hartley transform can be computed by fast algorithms (Bi and Zeng, 2012). By merging the obtained results, a unified algorithm for a sinusoid-class transform is proposed.

The papers by Saleh *et al.* (2003) and Potipantong *et al.* (2009) are devoted to the implementation of GDFT algorithms on FPGAs. Saleh *et al.* (2003) proposed to calculate the $N = 2^m$-point DFT in terms of DFT-II blocks of shorter length. For example, for $N = 16$, a fast DFT algorithm is obtained using a DFT-II of length $N/2$, DFT-II of length $N/4$, and DFT-II of length $N/8$. The proposed algorithms are used to develop fast Fourier transform algorithms with a significant improvement in hardware resources for implementation on FPGAs. In the work of Potipantong *et al.* (2009), an FPGA implementation of highly modular discrete trigonometric transforms including the GDFT is realized. Two practical applications, such as the joint photographic experts group (JPEG) co-processor and a powerline communication subsystem, also demonstrated the flexibility and modularity of the proposed FPGA implementation.

A brief review of the existing fast GDFT algorithms enabled the identification of key limitations inherent in the radix-based design approach. Furthermore, it facilitated the formulation of unresolved aspects of the broader problem of computational complexity reduction.

1.2. Main contributions. Radix-type GDFT algorithms (Bi and Zeng, 2012; Pei and Luo, 1996; Britanak and Rao, 1999; Bi and Chen, 1998; Zheng, 1996) for an input data sequence of length 2^m, where m is a positive integer, differ from other fast algorithms by a relatively simple regular computational structure and reasonable computational complexity. They often allow recursive computations for many different sizes of input sequences (Bi and Zeng, 2012).

Despite the advantages of radix-type algorithms for a size-2^m GDFT, the length of the input sequence may differ from that adopted in the algorithm under study. To apply a fast algorithm to a certain-size GDFT, the zero-padding technique is often used to increase the length of the input sequence, which usually leads to redundant computations. In addition, in many papers, authors present fast algorithms without providing data flow graphs, which makes them difficult to understand and implement (Bi and Zeng, 2012; Pei and Luo, 1996; Britanak and Rao, 1999; Bi and Chen, 1998; Zheng, 1996; Saleh *et al.*, 2003; Potipantong *et al.*, 2009). In electrical and systems engineering, control theory, theoretical computer science, etc., signal flow graphs are often used as a modeling tool for implementing a system as an electronic device connecting system components (Perera, 2018).

To avoid these drawbacks of the existing GDFT algorithms, it is reasonable to use the structural approach to matrix factorization proposed by Cariow (2014). Based on this method, the block structure of the transform matrix can be reduced to one of the matrix patterns given by Cariow (2014). Then, the extracted matrix patterns are used to obtain advantageous factorizations of the original matrices. As a result of these factorizations, the number of multiplications and additions required to calculate the product of the factored matrix and a vector is reduced

compared to the direct calculation of the product of the matrix and a vector. Based on these factorizations, it is quite easy to construct signal flow graphs by directly mapping the structures of the matrix factors to the graph geometry. In addition, unlike in traditional methods, in the structural approach the length of the original sequence is not limited to a power of two or three (Cariow and Paplinski, 2021; Polyakova *et al.*, 2024; Polyakova and Cariow, 2024).

This study aims to develop low-complexity DFT-II algorithms based on the structural approach for input sequences of lengths $N = 3, 4, 5, 6, 7, 8$.

2. Preliminary remarks

The n-point DFT-II is defined by the formula (Bi and Zeng, 2012)

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{-j\frac{2\pi}{N}(n+0.5)k}, \quad k = 0, 1, \ldots, N-1, \tag{1}$$

where $X(k)$ is the k-th DFT-coefficient, $x(n)$ is the n-th signal sample, N is the number of signal samples, $n+0.5$ is a half-sample shift in the time domain.

The DFT-II (1) in matrix notation takes the form

$$\boldsymbol{Y}_{N\times 1} = \boldsymbol{F}_N \boldsymbol{X}_{N\times 1}, \tag{2}$$

where

$$\boldsymbol{F}_N = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ e^{-j\frac{2\pi}{N}(0.5)} & e^{-j\frac{2\pi}{N}(1.5)} & \cdots & e^{-j\frac{2\pi}{N}(N-0.5)} \\ \vdots & \vdots & \ddots & \vdots \\ e^{-j\frac{2\pi}{N}(0.5)(N-1)} & e^{-j\frac{2\pi}{N}(1.5)(N-1)} & \cdots & e^{-j\frac{2\pi}{N}(N-0.5)(N-1)} \end{bmatrix} \tag{3}$$

and

$$\begin{aligned} \boldsymbol{Y}_{N\times 1} &= [y_0, y_1, \ldots, y_{N-1}]^{\mathrm{T}}, \\ \boldsymbol{X}_{N\times 1} &= [x_0, x_1, \ldots, x_{N-1}]^{\mathrm{T}}. \end{aligned} \tag{4}$$

Section 3 presents the derivations of computationally efficient algorithms for calculating the DFT-II for $N = 3, 4, \ldots, 8$.

3. Algorithms for a small-sized DFT-II

In general, the construction of fast algorithms for small-sized DFT-II calculations based on the structural approach (Cariow, 2014) involves the following main stages:

1. *Permutation of rows and/or columns of the initial transform matrix.* Permutation matrices are constructed to rearrange the rows and/or columns of the original matrix.

2. *Extraction of structurally redundant entries from the matrix.* From the current matrix, entries with identical absolute values (typically ± 1 or $\pm j$) are extracted. The matrix is split into two components:

   - the first matrix comprises entries with trivial values, which do not necessitate additional reduction of arithmetic complexity;
   - the second matrix includes the remaining entries and is subject to further minimization of arithmetic operations.

3. *Extraction and factorization of structural templates.* Submatrices matching predefined structural templates are extracted from the second matrix. Each submatrix is then factorized according to the corresponding template, as described by Cariow (2014).

4. *Construction of the factorization of the initial transform matrix.* If necessary, stages 1–3 are repeated for submatrices. Based on the obtained factorizations, the overall factorization of the initial transform matrix is assembled, incorporating the permutation matrices from stage 1.

5. *Improving the resulting factorization to minimize the number of additions.* The final factorization is refined to reduce the computational cost, particularly the number of addition operations.

3.1. Algorithm for three-point DFT-II calculation.

Let $\boldsymbol{X}_{3\times 1} = [x_0, x_1, x_2]^{\mathrm{T}}$ and $\boldsymbol{Y}_{3\times 1} = [y_0, y_1, y_2]^{\mathrm{T}}$ be 3-dimensional input and output data vectors, respectively. The problem is to calculate a product:

$$\boldsymbol{Y}_{3\times 1} = \boldsymbol{F}_3 \boldsymbol{X}_{3\times 1}, \tag{5}$$

with the DFT matrix

$$\boldsymbol{F}_3 = \begin{bmatrix} 1 & 1 & 1 \\ a^{(3)} & -1 & b^{(3)} \\ -b^{(3)} & 1 & -a^{(3)} \end{bmatrix} \tag{6}$$

and

$$a^{(3)} = e^{-j\pi/3}, \quad b^{(3)} = e^{-j5\pi/3}. \tag{7}$$

Calculating (5) directly requires four multiplications and six additions.

Stage 1. To decrease computational complexity, we interchange the first and second columns in the matrix $\boldsymbol{F}_3$, multiplying it simultaneously with -1. In that case,

we can decompose this matrix into the new matrix $\boldsymbol{F}_3^{(1)}$ multiplied by the permutation matrix $\boldsymbol{P}_3$:

$$\boldsymbol{F}_3 = \boldsymbol{F}_3^{(1)} \boldsymbol{P}_3^{(1)}, \tag{8}$$

where

$$\begin{aligned} \boldsymbol{F}_3^{(1)} &= \left[\begin{array}{ccc} 1 & -1 & 1 \\ a^{(3)} & -b^{(3)} & -1 \\ -b^{(3)} & a^{(3)} & 1 \end{array}\right], \\ \boldsymbol{P}_3 &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix}. \end{aligned} \tag{9}$$

Stage 2. We can decompose the matrix $\boldsymbol{F}_3^{(1)}$ into two submatrices:

$$\boldsymbol{F}_3^{(2)} = \begin{bmatrix} 1 & -1 & 1 \\ 0 & 0 & -1 \\ 0 & 0 & 1 \end{bmatrix}, \boldsymbol{F}_2 = \begin{bmatrix} a^{(3)} & -b^{(3)} \\ -b^{(3)} & a^{(3)} \end{bmatrix}. \tag{10}$$

Stage 3. Due to its internal structure, the matrix $\boldsymbol{F}_2$ can be represented as a product of the matrices:

$$\boldsymbol{F}_2 = \boldsymbol{H}_2 \boldsymbol{D}_2 \boldsymbol{H}_2, \tag{11}$$

where

$$\boldsymbol{H}_2 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}, \quad \boldsymbol{D}_2 = \operatorname{diag}\left(s_0^{(3)}, \quad s_1^{(3)}\right), \tag{12}$$

and

$$s_0^{(3)} = \frac{a^{(3)} - b^{(3)}}{2} = -j \sin(\pi/3), \tag{13}$$

$$s_1^{(3)} = \frac{a^{(3)} + b^{(3)}}{2} = \cos(\pi/3) = 0.5. \tag{14}$$

Stage 4. Consequently, when the $\boldsymbol{F}_3^{(2)}$ and $\boldsymbol{F}_2$ submatrices are folded, the $\boldsymbol{F}_3^{(1)}$ matrix can be represented as

$$\boldsymbol{F}_3^{(1)} = \boldsymbol{T}_{3\times4}^{(0)} \boldsymbol{D}_4 \boldsymbol{T}_{4\times3}^{(0)}, \tag{15}$$

$$\begin{aligned} \boldsymbol{T}_{3\times4}^{(0)} &= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 1 & 1 \\ 0 & 1 & 1 & -1 \end{bmatrix}, \\ \boldsymbol{T}_{4\times3}^{(0)} &= \begin{bmatrix} 1 & -1 & 1 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & -1 & 0 \end{bmatrix}, \end{aligned} \tag{16}$$

$$\boldsymbol{D}_4 = \operatorname{diag}\left(1, 1, s_0^{(3)}, 0.5\right). \tag{17}$$

Stage 5. In the above relations, the number of additions can be reduced and the permutation matrix $\boldsymbol{P}_3$ can be combined with the $\boldsymbol{T}_{4\times3}^{(0)}$ matrix, thus allowing the third-order DFT-II to be represented in its final form as

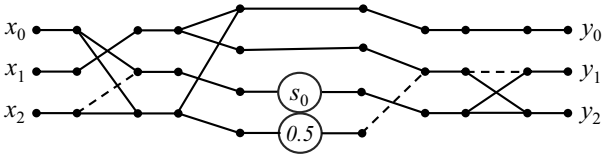

Fig. 1. Data flow graph of the three-point DFT-II.

$$\boldsymbol{Y}_3 = \boldsymbol{T}_3^{(0)} \boldsymbol{T}_{3\times4}^{(1)} \boldsymbol{D}_4 \boldsymbol{T}_{4\times3}^{(1)} \boldsymbol{T}_3^{(1)} \boldsymbol{X}_3, \tag{18}$$

where

$$\begin{aligned} \boldsymbol{T}_3^{(0)} &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 1 \\ 0 & 1 & 1 \end{bmatrix}, \\ \boldsymbol{T}_{3\times4}^{(1)} &= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & -1 \\ 0 & 0 & 1 & 0 \end{bmatrix}, \end{aligned} \tag{19}$$

$$\begin{aligned} \boldsymbol{T}_{4\times3}^{(1)} &= \begin{bmatrix} 1 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \\ \boldsymbol{T}_3^{(1)} &= \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & -1 \\ 1 & 0 & 1 \end{bmatrix}. \end{aligned} \tag{20}$$

Taking into account that the division by two can be done with a bitwiss shift, we obtain an algorithm for the third-order DFT-II that requires only one multiplication and six additions. Thus, three multiplications were reduced while the number of additions remained the same.

Figure 1 shows the data flow graph of the proposed algorithm (18). The paper presents data flow graphs in a left-to-right orientation, with the straight lines within the figures representing data transfer operations. The circles in these graphs represent multiplication operations, with the numerical factors inscribed inside. For simplicity, superscripts on multiplicators have been omitted in all figures, as it is self-evident which variable is referenced in each case. Points of convergence, marked with a bold dot, indicate summation. In addition, the dashed lines indicate data transfer operations with a simultaneous sign change. To maintain visual clarity, standard lines without arrows are used.

3.2. Algorithm for four-point DFT-II calculation. Let $\boldsymbol{X}_{4\times1} = [x_0, x_1, x_2, x_3]^{\mathrm{T}}$ be a four-dimensional data vector and $\boldsymbol{Y}_{4\times1} = [y_0, y_1, y_2, y_3]^{\mathrm{T}}$ be an output vector. The problem is to calculate a vector-matrix product:

$$\boldsymbol{Y}_{4\times1} = \boldsymbol{F}_4 \boldsymbol{X}_{4\times1}, \tag{21}$$

with the DFT-II matrix

$$\boldsymbol{F}_4 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ a^{(4)} - ja^{(4)} & -a^{(4)} - ja^{(4)} & -a^{(4)} + ja^{(4)} & a^{(4)} + ja^{(4)} \\ -j & j & -j & j \\ -a^{(4)} - ja^{(4)} & a^{(4)} - ja^{(4)} & a^{(4)} + ja^{(4)} & -a^{(4)} + ja^{(4)} \end{bmatrix} \tag{22}$$

and $a^{(4)} = \cos(\pi/4)$.

The direct calculation of Eqn. (21) requires eight complex multiplications and 12 additions. It is easy to see that the matrix $\boldsymbol{F}_4$ has a specific structure. Considering this, reducing the number of multiplications in calculating the four-point DFT-II is possible.

The structure of the matrix $\boldsymbol{F}_4$ enables immediate extraction of submatrices for factorization, thereby omitting the first and second stages of constructing a fast DFT-II algorithm based on the structural approach.

Stage 3. The $\boldsymbol{F}_4$ matrix, due to the occurrence of entries with the same value in the rows, can be represented as a product:

$$\boldsymbol{F}_4 = \boldsymbol{F}_4^{(0)} \boldsymbol{T}_4^{(0)}, \tag{23}$$

where

$$\boldsymbol{F}_4^{(0)} = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & a^{(4)} - ja^{(4)} & -a^{(4)} - ja^{(4)} \\ -j & j & 0 & 0 \\ 0 & 0 & -a^{(4)} - ja^{(4)} & a^{(4)} - ja^{(4)} \end{bmatrix}, \tag{24}$$

$$\boldsymbol{T}_4^{(0)} = \boldsymbol{H}_2 \otimes \boldsymbol{I}_2. \tag{25}$$

Stage 4. Considering only the non-zero entries of the $\boldsymbol{F}_4^{(0)}$ matrix, two submatrices can be extracted:

$$\begin{aligned} \boldsymbol{F}_2^{(0)} &= \begin{bmatrix} 1 & 1 \\ -j & j \end{bmatrix}, \\ \boldsymbol{F}_2^{(1)} &= \begin{bmatrix} a^{(4)} - ja^{(4)} & -a^{(4)} - ja^{(4)} \\ -a^{(4)} - ja^{(4)} & a^{(4)} - ja^{(4)} \end{bmatrix}. \end{aligned} \tag{26}$$

The structure of $\boldsymbol{F}_2^{(1)}$ can be simplified by expressing it as a product of three matrices:

$$\boldsymbol{F}_2^{(1)} = \boldsymbol{H}_2 \boldsymbol{F}_2^{(2)} \boldsymbol{H}_2, \tag{27}$$

where

$$\boldsymbol{F}_2^{(2)} = \begin{bmatrix} -ja^{(4)} & 0 \\ 0 & a^{(4)} \end{bmatrix}. \tag{28}$$

Consequently, the above relationships allow us to define a computationally efficient procedure for calculating the four-point DFT-II in the form

$$\boldsymbol{Y}_{4\times 1} = \boldsymbol{T}_4^{(2)} \boldsymbol{D}_4^{(0)} \boldsymbol{T}_4^{(1)} \boldsymbol{T}_4^{(0)} \boldsymbol{X}_{4\times 1}, \tag{29}$$

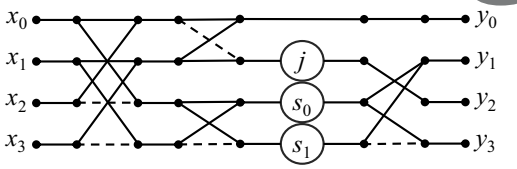

Fig. 2. Data flow graph of the four-point DFT-II.

where

$$\begin{aligned} \boldsymbol{T}_4^{(2)} &= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -1 \end{bmatrix}, \\ \boldsymbol{T}_4^{(1)} &= \begin{bmatrix} 1 & 1 & 0 & 0 \\ -1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & -1 \end{bmatrix}, \end{aligned} \tag{30}$$

and

$$\boldsymbol{D}_4^{(0)} = \operatorname{diag}\left(1, j, s_0^{(4)}, s_1^{(4)}\right),\ s_0^{(4)} = -ja^{(4)},\ s_1^{(4)} = a^{(4)}. \tag{31}$$

It takes two multiplications and 10 additions to calculate a four-point DFT-II. The proposed algorithm saves six multiplications and two additions compared to the direct matrix-vector product (DMVP) of determining the DFT-II (21).

As no further improvement of the factorization (31) is required, Stage 5 is omitted. Figure 2 shows a data flow graph of the four-point DFT-II.

3.3. Algorithm for five-point DFT-II calculation. Let $\boldsymbol{X}_{5\times 1} = [x_0, x_1, x_2, x_3, x_4]^{\mathrm{T}}$ be a five-dimensional data vector and $\boldsymbol{Y}_{5\times 1} = [y_0, y_1, y_2, y_3, y_4]^{\mathrm{T}}$ be an output vector. The problem is to calculate a vector-matrix product:

$$\boldsymbol{Y}_{5\times 1} = \boldsymbol{F}_5 \boldsymbol{X}_{5\times 1}, \tag{32}$$

with the DFT-II matrix

$$\boldsymbol{F}_5 = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ a^{(5)} - jb^{(5)} & -c^{(5)} - jd^{(5)} & -c^{(5)} + jd^{(5)} & -a^{(5)} + jb^{(5)} & c^{(5)} - jd^{(5)} \\ c^{(5)} - jd^{(5)} & -a^{(5)} + jb^{(5)} & a^{(5)} - jb^{(5)} & -c^{(5)} + jd^{(5)} & -a^{(5)} - jb^{(5)} \\ -c^{(5)} - jd^{(5)} & a^{(5)} + jb^{(5)} & -a^{(5)} - jb^{(5)} & c^{(5)} + jd^{(5)} & -c^{(5)} + jd^{(5)} \\ -a^{(5)} - jb^{(5)} & c^{(5)} - jd^{(5)} & -c^{(5)} + jd^{(5)} & a^{(5)} + jb^{(5)} & -a^{(5)} + jb^{(5)} \end{bmatrix} \tag{33}$$

and

$$a^{(5)} = \cos(\pi/5), \qquad b^{(5)} = \sin(\pi/5), \tag{34}$$

$$c^{(5)} = \cos(2\pi/5), \qquad d^{(5)} = \sin(2\pi/5). \tag{35}$$

The direct calculation of Eqn. (32) requires 16 complex multiplications and 20 additions.

Stage 1. To factorize the $\boldsymbol{F}_5$ matrix, we can swap the order of the rows and columns and multiply some of them by -1. Using the appropriate permutation matrices, we will obtain a new $\boldsymbol{F}_5^{(0)}$ matrix satisfying the relation

$$\boldsymbol{F}_5 = \boldsymbol{P}_5^{(1)} \boldsymbol{F}_5^{(0)} \boldsymbol{P}_5^{(0)}, \tag{36}$$

where

$$\boldsymbol{F}_5^{(0)} = \left[\begin{array}{ccccc} 1 & 1 & -1 & 1 & -1 \\ -1 & a^{(5)}-jb^{(5)} & c^{(5)}+jd^{(5)} & a^{(5)}+jb^{(5)} & c^{(5)}-jd^{(5)} \\ 1 & c^{(5)}-jd^{(5)} & a^{(5)}-jb^{(5)} & c^{(5)}+jd^{(5)} & a^{(5)}+jb^{(5)} \\ -1 & a^{(5)}+jb^{(5)} & c^{(5)}-jd^{(5)} & a^{(5)}-jb^{(5)} & c^{(5)}+jb^{(5)} \\ 1 & c^{(5)}+jd^{(5)} & a^{(5)}+jb^{(5)} & c^{(5)}-jd^{(5)} & a^{(5)}-jb^{(5)} \end{array}\right], \tag{37}$$

$$\boldsymbol{P}_5^{(0)} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & -1 & 0 \end{bmatrix}, \quad \boldsymbol{P}_5^{(1)} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & -1 & 0 \end{bmatrix}. \tag{38}$$

Stage 2. Two submatrices can be extracted from the matrix $\boldsymbol{F}_5^{(0)}$:

$$\boldsymbol{F}_5^{(1)} = \begin{bmatrix} 1 & 1 & -1 & 1 & -1 \\ -1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix}, \tag{39}$$

$$\boldsymbol{F}_4^{(1)} = \left[\begin{array}{cc:cc} a^{(5)}-jb^{(5)} & c^{(5)}+jd^{(5)} & a^{(5)}+jb^{(5)} & c^{(5)}-jd^{(5)} \\ c^{(5)}-jd^{(5)} & a^{(5)}-jb^{(5)} & c^{(5)}+jd^{(5)} & a^{(5)}+jb^{(5)} \\ \hdashline a^{(5)}+jb^{(5)} & c^{(5)}-jd^{(5)} & a^{(5)}-jb^{(5)} & c^{(5)}+jd^{(5)} \\ c^{(5)}+jd^{(5)} & a^{(5)}+jb^{(5)} & c^{(5)}-jd^{(5)} & a^{(5)}-jb^{(5)} \end{array}\right]. \tag{40}$$

Stage 3. The matrix $\boldsymbol{F}_4^{(1)}$ has a specific structure,

$$\boldsymbol{F}_4^{(1)} = \begin{bmatrix} \boldsymbol{F}_2^{(3)} & \boldsymbol{F}_2^{(4)} \\ \boldsymbol{F}_2^{(4)} & \boldsymbol{F}_2^{(3)} \end{bmatrix}, \tag{41}$$

and can be replaced by (Cariow, 2014)

$$\boldsymbol{F}_4^{(1)} = \boldsymbol{T}_4^{(0)} \Big[\frac{1}{2}\left(\boldsymbol{F}_2^{(3)} + \boldsymbol{F}_2^{(4)}\right) \oplus \frac{1}{2}\left(\boldsymbol{F}_2^{(3)} - \boldsymbol{F}_2^{(4)}\right) \Big] \boldsymbol{T}_4^{(0)}, \tag{42}$$

where

$$\boldsymbol{F}_2^{(3)} = \begin{bmatrix} a^{(5)}-jb^{(5)} & c^{(5)}+jd^{(5)} \\ c^{(5)}-jd^{(5)} & a^{(5)}-jb^{(5)} \end{bmatrix}, \tag{43}$$

$$\boldsymbol{F}_2^{(4)} = \begin{bmatrix} a^{(5)}+jb^{(5)} & c^{(5)}-jd^{(5)} \\ c^{(5)}+jd^{(5)} & a^{(5)}+jb^{(5)} \end{bmatrix}. \tag{44}$$

The symbols $\otimes$ and $\oplus$ denote a tensor product and a direct sum of two matrices, respectively (Regalia and Sanjit, 1989).

New matrices can be defined that are the sum and difference of matrices $\boldsymbol{F}_2^{(3)}$ and $\boldsymbol{F}_2^{(4)}$:

$$\boldsymbol{F}_2^{(5)} = \frac{1}{2}\left(\boldsymbol{F}_2^{(3)} + \boldsymbol{F}_2^{(4)}\right) = \begin{bmatrix} a^{(5)} & c^{(5)} \\ c^{(5)} & a^{(5)} \end{bmatrix}, \tag{45}$$

$$\boldsymbol{F}_2^{(6)} = \frac{1}{2}\left(\boldsymbol{F}_2^{(3)} - \boldsymbol{F}_2^{(4)}\right) = \begin{bmatrix} -jb^{(5)} & jd^{(5)} \\ -jd^{(5)} & -jb^{(5)} \end{bmatrix}. \tag{46}$$

The $\boldsymbol{F}_2^{(5)}$ matrix has a similar structure to the $\boldsymbol{F}_4^{(1)}$ matrix and can be factorized using identities (Cariow, 2014):

$$\boldsymbol{F}_2^{(5)} = \boldsymbol{H}_2 \left[\frac{1}{2}\left(a^{(5)} + c^{(5)}\right) \oplus \frac{1}{2}\left(a^{(5)} - c^{(5)}\right) \right] \boldsymbol{H}_2. \tag{47}$$

The $\boldsymbol{F}_2^{(6)}$ can be factorized using

$$\boldsymbol{F}_2^{(6)} = \boldsymbol{T}_{2\times 3} \Big[j\left(b^{(5)} - d^{(5)}\right) \oplus j\left(b^{(5)} + d^{(5)}\right) \oplus \left(-jb^{(5)}\right) \Big] \boldsymbol{T}_{3\times 2}, \tag{48}$$

where

$$\boldsymbol{T}_{2\times 3} = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix}, \quad \boldsymbol{T}_{3\times 2} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix}. \tag{49}$$

The $\boldsymbol{F}_5^{(1)}$ matrix can be represented as the expression

$$\boldsymbol{F}_5^{(1)} = \boldsymbol{T}_{5\times 2} \boldsymbol{T}_{2\times 5}, \tag{50}$$

where

$$\boldsymbol{T}_{5\times 2} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 1 & -1 & 1 \end{bmatrix}^{\mathrm{T}}, \quad \boldsymbol{T}_{5\times 2} = \begin{bmatrix} 1 & 1 & -1 & 1 & -1 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix}. \tag{51}$$

Stage 4. Given the relations (42), (47), (48), the matrix $\boldsymbol{F}_5^{(0)}$ can be represented as

$$\boldsymbol{F}_5^{(0)} = \boldsymbol{T}_{5\times 6} \boldsymbol{T}_{6\times 7} \boldsymbol{D}_7 \boldsymbol{T}_{7\times 6} \boldsymbol{T}_{6\times 5}, \tag{52}$$

where

$$\boldsymbol{T}_{5\times 6}=\begin{bmatrix}1&0&0&0&0&0\\0&-1&1&0&1&0\\0&1&0&1&0&1\\0&-1&1&0&-1&0\\0&1&0&1&0&-1\end{bmatrix}, \tag{53}$$

$$\boldsymbol{T}_{6\times 7}=\boldsymbol{I}_2\oplus\boldsymbol{H}_2\oplus\boldsymbol{T}_{2\times 3},\quad \boldsymbol{T}_{7\times 6}=\boldsymbol{I}_2\oplus\boldsymbol{H}_2\oplus\boldsymbol{T}_{3\times 2}, \tag{54}$$

$$\boldsymbol{T}_{6\times 5}=\begin{bmatrix}1&1&-1&1&-1\\1&0&0&0&0\\0&1&0&1&0\\0&0&1&0&1\\0&1&0&-1&0\\0&0&1&0&-1\end{bmatrix}, \tag{55}$$

$$\boldsymbol{D}_7=\operatorname{diag}\left(1,1,s_0^{(5)},s_1^{(5)},s_2^{(5)},s_3^{(5)},s_4^{(5)}\right), \tag{56}$$

$$\begin{aligned}s_0^{(5)}&=\frac{1}{2}\left(a^{(5)}+c^{(5)}\right),\\ s_1^{(5)}&=\frac{1}{2}\left(a^{(5)}-c^{(5)}\right),\end{aligned} \tag{57}$$

$$\begin{aligned}s_2^{(5)}&=j(b^{(5)}-d^{(5)}),\\ s_3^{(5)}&=j(b^{(5)}+d^{(5)}),\\ s_4^{(5)}&=-jb^{(5)}.\end{aligned} \tag{58}$$

Stage 5. The matrix $\boldsymbol{T}_{6\times 5}$ can be further decomposed into two matrices, which reduces the number of additions required for computing the DFT-II:

$$\boldsymbol{T}_{6\times 5}=\boldsymbol{T}_{6\times 5}^{(0)}\boldsymbol{T}_5^{(0)}, \tag{59}$$

$$\boldsymbol{T}_{6\times 5}^{(0)}=\begin{bmatrix}1&1&-1&0&0\\1&0&0&0&0\\0&1&0&0&0\\0&0&1&0&0\\0&0&0&1&0\\0&0&0&0&1\end{bmatrix}, \tag{60}$$

$$\boldsymbol{T}_5^{(0)}=\begin{bmatrix}1&0&0&0&0\\0&1&0&1&0\\0&0&1&0&1\\0&1&0&-1&0\\0&0&1&0&-1\end{bmatrix}.$$

The $\boldsymbol{T}_{5\times 6}$ matrix can be split similarly:

$$\boldsymbol{T}_{5\times 6}=\boldsymbol{T}_5^{(1)}\boldsymbol{T}_{5\times 6}^{(0)}, \tag{61}$$

$$\boldsymbol{T}_5^{(1)}=\begin{bmatrix}1&0&0&0&0\\0&1&1&0&0\\0&0&0&1&1\\0&1&-1&0&0\\0&0&0&1&-1\end{bmatrix}, \tag{62}$$

$$\boldsymbol{T}_{5\times 6}^{(0)}=\begin{bmatrix}1&0&0&0&0&0\\0&-1&1&0&0&0\\0&0&0&0&1&0\\0&1&0&1&0&0\\0&0&0&0&0&1\end{bmatrix}.$$

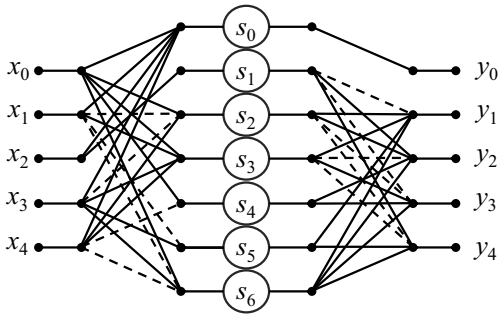

Fig. 3. Signal flow graph for the computation of the five-point DFT-II (63).

Taking into account the above modifications, the final form of determining the five-point DFT-II transform, reducing the number of multiplications, can be presented as

$$\boldsymbol{Y}_{5\times 1}=\boldsymbol{T}_5^{(3)}\boldsymbol{T}_{5\times 6}^{(0)}\boldsymbol{T}_{6\times 7}\boldsymbol{D}_7\boldsymbol{T}_{7\times 6}\boldsymbol{T}_{6\times 5}^{(0)}\boldsymbol{T}_5^{(2)}\boldsymbol{X}_{5\times 1}, \tag{63}$$

where

$$\boldsymbol{T}_5^{(3)}=\boldsymbol{P}_5^{(1)}\boldsymbol{T}_5^{(1)},\quad \boldsymbol{T}_5^{(2)}=\boldsymbol{T}_5^{(0)}\boldsymbol{P}_5^{(0)}. \tag{64}$$

The calculation of a five-point DFT-II by (63) requires five multiplications and 18 additions. The proposed algorithm saves 15 multiplications and two additions compared to the DMVP for determining the DFT-II (32).

Figure 3 shows the signal flow graph of the proposed algorithm to implement the five-point DFT-II.

3.4. Algorithm for six-point DFT-II calculation. Let $\boldsymbol{X}_{6\times 1}=[x_0,x_1,x_2,x_3,x_4,x_5]^{\mathrm{T}}$ be a six-dimensional data vector and $\boldsymbol{Y}_{6\times 1}=[y_0,y_1,y_2,y_3,y_4,y_5]^{\mathrm{T}}$ be an output vector. The problem is to calculate a vector-matrix product:

$$\boldsymbol{Y}_{6\times 1}=\boldsymbol{F}_6\boldsymbol{X}_{6\times 1}, \tag{65}$$

with the DFT-II matrix

$$\boldsymbol{F}_6=\begin{bmatrix}1&1&1\\a^{(6)}-jb^{(6)}&-1&-a^{(6)}-jb^{(6)}\\b^{(6)}-ja^{(6)}&-1&b^{(6)}+ja^{(6)}\\-j&j&-j\\-b^{(6)}-ja^{(6)}&1&-b^{(6)}+ja^{(6)}\\-a^{(6)}-jb^{(6)}&-j&a^{(6)}-jb^{(6)}\end{bmatrix}$$
$$\begin{bmatrix}1&1&1\\-a^{(6)}+jb^{(6)}&j&a^{(6)}+jb^{(6)}\\b^{(6)}-ja^{(6)}&-1&b^{(6)}+ja^{(6)}\\j&-j&j\\-b^{(6)}-ja^{(6)}&1&-b^{(6)}+ja^{(6)}\\a^{(6)}+jb^{(6)}&j&-a^{(6)}+jb^{(6)}\end{bmatrix} \tag{66}$$

and

$$a^{(6)} = \cos(\pi/6) = \sqrt{3}/2, \quad b^{(6)} = \cos(\pi/3) = 1/2. \tag{67}$$

The direct calculation of Eqn. (65) requires 16 complex multiplications and 30 additions.

In this case, the permutation of rows and columns of the original matrix is not utilized and Stage 1 of constructing the algorithm is not applied.

Stage 2. In the matrix $\boldsymbol{F}_6$ (66), we can notice the presence of identical values in the rows, which allows us to reduce the complexity of calculating the matrix-vector product. Thus, the computational procedure describing the computationally efficient algorithm for calculating DFT-II takes the following form:

$$\boldsymbol{Y}_{6\times1} = \boldsymbol{T}_{6\times10}\boldsymbol{F}_{10\times8}\boldsymbol{T}_{8\times6}\boldsymbol{X}_{6\times1}, \tag{68}$$

where

$$\boldsymbol{T}_{8\times6} = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & -1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & -1 \end{bmatrix}, \tag{69}$$

$$\boldsymbol{F}_{10\times8} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & j & 0 & 0 \\ -j & 0 & 0 & 0 & 0 & 0 \\ 0 & j & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & b^{(6)} - ja^{(6)} & b^{(6)} + ja^{(6)} \\ 0 & 0 & 0 & 0 & b^{(6)} + ja^{(6)} & b^{(6)} - ja^{(6)} \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$
$$\begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ a^{(6)} - jb^{(6)} & -a^{(6)} - jb^{(6)} \\ -a^{(6)} - jb^{(6)} & a^{(6)} - jb^{(6)} \end{bmatrix}, \tag{70}$$

$$\boldsymbol{T}_{6\times10} = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}. \tag{71}$$

Stage 3. If two submatrices are extracted from the matrix $\boldsymbol{F}_{10\times8}$, namely,

$$\boldsymbol{F}_2^{(7)} = \begin{bmatrix} b^{(6)} - ja^{(6)} & b^{(6)} + ja^{(6)} \\ b^{(6)} + ja^{(6)} & b^{(6)} - ja^{(6)} \end{bmatrix}, \tag{72}$$

$$\boldsymbol{F}_2^{(8)} = \begin{bmatrix} a^{(6)} - jb^{(6)} & -a^{(6)} - jb^{(6)} \\ -a^{(6)} - jb^{(6)} & a^{(6)} - jb^{(6)} \end{bmatrix}, \tag{73}$$

it is easy to observe that both matrices exhibit a structure very similar to that of matrix (45). Thus, applying the formula (47) to them, we obtain

$$\boldsymbol{F}_2^{(7)} = \boldsymbol{H}_2\boldsymbol{D}_2^{(6)}\boldsymbol{H}_2, \; \boldsymbol{F}_2^{(8)} = \boldsymbol{H}_2\boldsymbol{D}_2^{(7)}\boldsymbol{H}_2, \tag{74}$$

where

$$\boldsymbol{D}_2^{(6)} = \left[\frac{1}{2}\left(b^{(6)} - ja^{(6)} + b^{(6)} + ja^{(6)}\right) \oplus \frac{1}{2}\left(b^{(6)} - ja^{(6)} - (b^{(6)} + ja^{(6)})\right)\right], \tag{75}$$

$$\boldsymbol{D}_2^{(7)} = \left[\frac{1}{2}\left(a^{(6)} - jb^{(6)} - a^{(6)} - jb^{(6)}\right) \oplus \frac{1}{2}\left(a^{(6)} - jb^{(6)} - (-a^{(6)} - jb^{(6)})\right)\right]. \tag{76}$$

Stage 4. By applying the above transformations to the expression (68), a computationally efficient algorithm for computing the six-point DFT-II can be derived:

$$\boldsymbol{Y}_{6\times1} = \boldsymbol{T}_{6\times10}\boldsymbol{T}_{10}\boldsymbol{D}_{10}\boldsymbol{T}_{10\times8}^{(0)}\boldsymbol{T}_{8\times6}\boldsymbol{X}_{6\times1}, \tag{77}$$

where

$$\boldsymbol{T}_{10} = \boldsymbol{I}_6 \oplus \boldsymbol{H}_2 \oplus \boldsymbol{H}_2, \tag{78}$$

$$\boldsymbol{T}_{10\times8}^{(0)} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 0 \end{bmatrix} \otimes \boldsymbol{I}_2 \oplus \boldsymbol{H}_2 \oplus \boldsymbol{H}_2, \tag{79}$$

$$\boldsymbol{D}_{10} = \operatorname{diag}\left(1, 1, 1, j, -j, j, 0.5, s_0^{(6)}, s_1^{(6)}, s_2^{(6)}\right), \tag{80}$$

and

$$s_0^{(6)} = -j\sqrt{3}/2, \quad s_1^{(6)} = -j0.5, \quad s_2^{(6)} = \sqrt{3}/2. \tag{81}$$

The calculation of a six-point DFT-II by (77) requires only two multiplications and 24 additions. The proposed algorithm saves 14 multiplications and six additions compared to the DMVP for calculating the DFT-II (65).

Improving the obtained factorization (77) is not necessary and Stage 5 is omitted. Figure 4 shows the signal flow graph of the proposed algorithm to implement the six-point DFT-II.

3.5. Algorithm for seven-point DFT-II calculation. Let $\boldsymbol{X}_{7\times1} = [x_0, x_1, x_2, x_3, x_4, x_5, x_6]^{\mathrm{T}}$ be a seven-dimensional data vector being transformed by the DFT-II and $\boldsymbol{Y}_{7\times1} = [y_0, y_1, y_2, y_3, y_4, y_5, y_6]^{\mathrm{T}}$ be an output vector. The problem is to calculate a vector-matrix product:

$$\boldsymbol{Y}_{7\times1} = \boldsymbol{F}_7 \boldsymbol{X}_{7\times1}, \tag{82}$$

$$\boldsymbol{F}_7 = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ a^{(7)} & b^{(7)} & -c^{(7)} & -1 & -d^{(7)} & f^{(7)} & g^{(7)} \\ d^{(7)} & -g^{(7)} & -b^{(7)} & 1 & -f^{(7)} & -a^{(7)} & c^{(7)} \\ b^{(7)} & -d^{(7)} & a^{(7)} & -1 & g^{(7)} & -c^{(7)} & f^{(7)} \\ -f^{(7)} & c^{(7)} & -g^{(7)} & 1 & -a^{(7)} & d^{(7)} & -b^{(7)} \\ -c^{(7)} & a^{(7)} & f^{(7)} & -1 & b^{(7)} & g^{(7)} & -d^{(7)} \\ -g^{(7)} & -f^{(7)} & d^{(7)} & 1 & c^{(7)} & -b^{(7)} & -a^{(7)} \end{bmatrix}, \tag{83}$$

and

$$a^{(7)} = e^{-j\pi/7},\ b^{(7)} = e^{-j\pi3/7},\ c^{(7)} = -e^{-j\pi5/7}, \tag{84}$$

$$d^{(7)} = e^{-j\pi2/7},\ f^{(7)} = -e^{-j\pi4/7},\ g^{(7)} = -e^{-j\pi6/7}. \tag{85}$$

The direct calculation of Eqn. (82) requires 36 complex multiplications and 42 additions.

Stage 1. To factorize the matrix $\boldsymbol{F}_7$, we rearrange its rows and columns and multiply selected ones by -1. By applying appropriate permutation matrices, we obtain a new matrix $\boldsymbol{F}_7^{(0)}$ that satisfies the following relation:

$$\boldsymbol{F}_7 = \boldsymbol{P}_7^{(1)} \boldsymbol{F}_7^{(0)} \boldsymbol{P}_7^{(0)}, \tag{86}$$

where

$$\boldsymbol{F}_7^{(0)} = \left[\begin{array}{ccc:c:ccc} 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ a^{(7)} & b^{(7)} & -c^{(7)} & -1 & -g^{(7)} & -f^{(7)} & d^{(7)} \\ -c^{(7)} & a^{(7)} & f^{(7)} & -1 & d^{(7)} & -g^{(7)} & -b^{(7)} \\ b^{(7)} & -d^{(7)} & a^{(7)} & -1 & -f^{(7)} & c^{(7)} & -g^{(7)} \\ \hdashline -g^{(7)} & -f^{(7)} & d^{(7)} & 1 & a^{(7)} & b^{(7)} & -c^{(7)} \\ d^{(7)} & -g^{(7)} & -b^{(7)} & 1 & -c^{(7)} & a^{(7)} & f^{(7)} \\ -f^{(7)} & c^{(7)} & -g^{(7)} & 1 & b^{(7)} & -d^{(7)} & a^{(7)} \end{array}\right], \tag{87}$$

$$\boldsymbol{P}_7^{(0)} = \boldsymbol{I}_4 \oplus \begin{bmatrix} 0 & 0 & -1 \\ 0 & -1 & 0 \\ -1 & 0 & 0 \end{bmatrix}, \tag{88}$$

$$\boldsymbol{P}_7^{(1)} = \begin{bmatrix} 1&0&0&0&0&0&0 \\ 0&1&0&0&0&0&0 \\ 0&0&0&0&0&1&0 \\ 0&0&0&1&0&0&0 \\ 0&0&0&0&0&0&1 \\ 0&0&1&0&0&0&0 \\ 0&0&0&0&1&0&0 \end{bmatrix}. \tag{89}$$

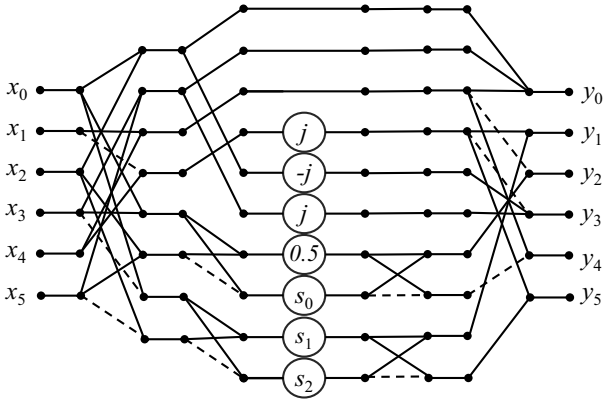

Fig. 4. Signal flow graph for the computation of the six-point DFT-II (77).

Stage 2. From the matrix $\boldsymbol{F}_7^{(0)}$, two submatrices can be extracted:

$$\boldsymbol{F}_7^{(1)} = \begin{bmatrix} 1&1&1&1&1&1&1 \\ 0&0&0&-1&0&0&0 \\ 0&0&0&-1&0&0&0 \\ 0&0&0&-1&0&0&0 \\ 0&0&0&1&0&0&0 \\ 0&0&0&1&0&0&0 \\ 0&0&0&1&0&0&0 \end{bmatrix}, \tag{90}$$

$$\boldsymbol{F}_6^{(0)} = \left[\begin{array}{ccc:ccc} a^{(7)} & b^{(7)} & -c^{(7)} & -g^{(7)} & -f^{(7)} & d^{(7)} \\ -c^{(7)} & a^{(7)} & f^{(7)} & d^{(7)} & -g^{(7)} & -b^{(7)} \\ b^{(7)} & -d^{(7)} & a^{(7)} & -f^{(7)} & c^{(7)} & -g^{(7)} \\ \hdashline -g^{(7)} & -f^{(7)} & d^{(7)} & a^{(7)} & b^{(7)} & -c^{(7)} \\ d^{(7)} & -g^{(7)} & -b^{(7)} & -c^{(7)} & a^{(7)} & f^{(7)} \\ -f^{(7)} & c^{(7)} & -g^{(7)} & b^{(7)} & -d^{(7)} & a^{(7)} \end{array}\right]. \tag{91}$$

Stage 3. The matrix $\boldsymbol{F}_6^{(0)}$ has a specific structure and can alternatively be expressed as proposed by Cariow (2014):

$$\boldsymbol{F}_6^{(0)} = \begin{bmatrix} \boldsymbol{F}_3^{(3)} & \boldsymbol{F}_3^{(4)} \\ \boldsymbol{F}_3^{(4)} & \boldsymbol{F}_3^{(3)} \end{bmatrix} = \boldsymbol{T}_6 \boldsymbol{F}_6^{(1)} \boldsymbol{T}_6, \tag{92}$$

where

$$\boldsymbol{T}_6 = \boldsymbol{H}_2 \otimes \boldsymbol{I}_3, \quad \boldsymbol{F}_6^{(1)} = \left[\boldsymbol{F}_3^{(5)} \oplus \boldsymbol{F}_3^{(6)}\right], \tag{93}$$

and

$$\begin{aligned} \boldsymbol{F}_3^{(5)} &= \frac{1}{2}\left(\boldsymbol{F}_3^{(3)} + \boldsymbol{F}_3^{(4)}\right) \\ &= \frac{1}{2}\begin{bmatrix} a^{(7)} - g^{(7)} & b^{(7)} - f^{(7)} & d^{(7)} - c^{(7)} \\ d^{(7)} - c^{(7)} & a^{(7)} - g^{(7)} & f^{(7)} - b^{(7)} \\ b^{(7)} - f^{(7)} & c^{(7)} - d^{(7)} & a^{(7)} - g^{(7)} \end{bmatrix}, \end{aligned} \tag{94}$$

$$\boldsymbol{F}_3^{(6)} = \frac{1}{2}\left(\boldsymbol{F}_3^{(3)} - \boldsymbol{F}_3^{(4)}\right) = \frac{1}{2}\begin{bmatrix} a^{(7)}+g^{(7)} & b^{(7)}+f^{(7)} & -c^{(7)}-d^{(7)} \\ -c^{(7)}-d^{(7)} & a^{(7)}+g^{(7)} & b^{(7)}+f^{(7)} \\ b^{(7)}+f^{(7)} & -c^{(7)}-d^{(7)} & a^{(7)}+g^{(7)} \end{bmatrix}. \tag{95}$$

In the $\boldsymbol{F}_3^{(5)}$ matrix, we can multiply the first row and column by -1. Using the appropriate matrices, we will obtain a new $\boldsymbol{F}_3^{(7)}$ matrix:

$$\boldsymbol{F}_3^{(7)} = \boldsymbol{P}_3^{(0)}\boldsymbol{F}_3^{(5)}\boldsymbol{P}_3^{(0)}, \tag{96}$$

where

$$\boldsymbol{F}_3^{(7)} = \frac{1}{2}\begin{bmatrix} a^{(7)}-g^{(7)} & f^{(7)}-b^{(7)} & c^{(7)}-d^{(7)} \\ c^{(7)}-d^{(7)} & a^{(7)}-g^{(7)} & f^{(7)}-b^{(7)} \\ f^{(7)}-b^{(7)} & c^{(7)}-d^{(7)} & a^{(7)}-g^{(7)} \end{bmatrix}, \tag{97}$$

$$\boldsymbol{P}_3^{(0)} = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{98}$$

The matrices $\boldsymbol{F}_3^{(6)}$ and $\boldsymbol{F}_3^{(7)}$ are circular matrices for which the following algorithm can be applied (Cariow and Paplinski, 2021; Polyakova *et al.*, 2024):

$$\boldsymbol{H}_3 = \boldsymbol{A}_{3\times 4}\boldsymbol{D}_4^{(H_3)}\boldsymbol{A}_{4\times 3}, \tag{99}$$

where the matrix $\boldsymbol{H}_3$ is some circulant matrix:

$$\boldsymbol{H}_3 = \begin{bmatrix} h_1 & h_0 & h_2 \\ h_2 & h_1 & h_0 \\ h_0 & h_2 & h_1 \end{bmatrix}, \tag{100}$$

$$\boldsymbol{D}_4^{(H_3)} = \operatorname{diag}\left(s_0^{(H_3)},\ s_1^{(H_3)},\ s_2^{(H_3)},\ s_3^{(H_3)}\right), \tag{101}$$

$$\begin{aligned} s_0^{(H_3)} &= (h_0+h_1+h_2)/3, \\ s_1^{(H_3)} &= (2h_1-h_0-h_2)/3, \end{aligned} \tag{102}$$

$$\begin{aligned} s_2^{(H_3)} &= (h_1-2h_0+h_2)/3, \\ s_3^{(H_3)} &= (h_0+h_1-2h_2)/3, \end{aligned} \tag{103}$$

$$\boldsymbol{A}_{4\times 3} = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & -1 \\ 1 & -1 & 0 \\ 1 & 0 & -1 \end{bmatrix}, \quad \boldsymbol{A}_{3\times 4} = \begin{bmatrix} 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & -1 \\ 1 & -1 & -1 & 0 \end{bmatrix}, \tag{104}$$

and h_0, h_1, h_2 are some values present in the circular matrix.

For the matrix $\boldsymbol{F}_3^{(5)}$, the following relation is obtained:

$$\boldsymbol{F}_3^{(5)} = \boldsymbol{P}_3^{(0)}\boldsymbol{A}_{3\times 4}\boldsymbol{D}_4^{(\boldsymbol{F}_3^{(5)})}\boldsymbol{A}_{4\times 3}\boldsymbol{P}_3^{(0)}, \tag{105}$$

where

$$\boldsymbol{D}_4^{(\boldsymbol{F}_3^{(5)})} = \operatorname{diag}\left(s_0^{(7)},\ s_1^{(7)},\ s_2^{(7)},\ s_3^{(7)}\right), \tag{106}$$

$$\begin{aligned} s_0^{(7)} &= \frac{1}{6}\left(a^{(7)}-b^{(7)}+c^{(7)}-d^{(7)}+f^{(7)}-g^{(7)}\right) \approx 0.018, \\ s_1^{(7)} &= \frac{1}{6}\left(2a^{(7)}+b^{(7)}-c^{(7)}+d^{(7)}-f^{(7)}-2g^{(7)}\right) \approx 0.883, \\ s_2^{(7)} &= \frac{1}{6}\left(a^{(7)}+2b^{(7)}+c^{(7)}-d^{(7)}-2f^{(7)}-g^{(7)}\right) \approx 0.241, \\ s_3^{(7)} &= \frac{1}{6}\left(a^{(7)}-b^{(7)}-2c^{(7)}+2d^{(7)}+f^{(7)}-g^{(7)}\right) \approx 0.642. \end{aligned} \tag{107}$$

Similarly, for the matrix $\boldsymbol{F}_3^{(6)}$, the following relation holds:

$$\boldsymbol{F}_3^{(6)} = \boldsymbol{A}_{3\times 4}\boldsymbol{D}_4^{(\boldsymbol{F}_3^{(6)})}\boldsymbol{A}_{4\times 3}, \tag{108}$$

where

$$\boldsymbol{D}_4^{(\boldsymbol{F}_3^{(6)})} = \operatorname{diag}\left(s_4^{(7)},\ s_5^{(7)},\ s_6^{(7)},\ s_7^{(7)}\right), \tag{109}$$

$$\begin{aligned} s_4^{(7)} &= \frac{1}{6}\left(a^{(7)}+b^{(7)}-c^{(7)}-d^{(7)}+f^{(7)}+g^{(7)}\right) \approx \mathrm{j}0.209, \\ s_5^{(7)} &= \frac{1}{6}\left(2a^{(7)}-b^{(7)}+c^{(7)}+d^{(7)}-f^{(7)}+2g^{(7)}\right) \approx \mathrm{j}0.225, \\ s_6^{(7)} &= \frac{1}{6}\left(a^{(7)}-2b^{(7)}-c^{(7)}-d^{(7)}-2f^{(7)}+g^{(7)}\right) \approx \mathrm{j}0.766, \\ s_7^{(7)} &= \frac{1}{6}\left(a^{(7)}+b^{(7)}+2c^{(7)}+2d^{(7)}+f^{(7)}+g^{(7)}\right) \approx \mathrm{j}0.991. \end{aligned} \tag{110}$$

Stage 4. Taking the above modifications into account, the seven-point DFT-II transform matrix can be represented as

$$\boldsymbol{F}_7 = \boldsymbol{P}_7^{(1)}\boldsymbol{T}_{7\times 8}\boldsymbol{T}_{8\times 10}^{(0)}\boldsymbol{D}_{10}\boldsymbol{T}_{10\times 8}^{(1)}\boldsymbol{T}_{8\times 7}\boldsymbol{P}_7^{(0)}, \tag{111}$$

where

$$\boldsymbol{T}_{8\times 7} = \begin{bmatrix} \mathbf{1}_{1\times 3} & 0 & -\mathbf{1}_{1\times 3} \\ \mathbf{0}_{1\times 3} & 1 & \mathbf{0}_{1\times 3} \\ \boldsymbol{I}_3 & \mathbf{0}_{3\times 1} & \boldsymbol{I}_3 \\ \boldsymbol{I}_3 & \mathbf{0}_{3\times 1} & -\boldsymbol{I}_3 \end{bmatrix}, \tag{112}$$

$$\boldsymbol{T}_{10\times 8}^{(1)} = \begin{bmatrix} 1&0&0&0&0&0&0&0\\ 0&1&0&0&0&0&0&0\\ 0&0&-1&1&1&0&0&0\\ 0&0&0&1&-1&0&0&0\\ 0&0&-1&-1&0&0&0&0\\ 0&0&-1&0&-1&0&0&0\\ 0&0&0&0&0&1&1&1\\ 0&0&0&0&0&0&1&-1\\ 0&0&0&0&0&1&-1&0\\ 0&0&0&0&0&1&0&-1 \end{bmatrix}, \tag{113}$$

$$\boldsymbol{T}_{8\times 10}^{(0)} = \begin{bmatrix} 1&0&0&0&0&0&0&0&0&0\\ 0&1&0&0&0&0&0&0&0&0\\ 0&0&-1&0&-1&-1&0&0&0&0\\ 0&0&1&1&0&-1&0&0&0&0\\ 0&0&1&-1&-1&0&0&0&0&0\\ 0&0&0&0&0&0&1&0&1&1\\ 0&0&0&0&0&0&1&1&0&-1\\ 0&0&0&0&0&0&1&-1&-1&0 \end{bmatrix}, \tag{114}$$

$$\boldsymbol{T}_{7\times 8} = \begin{bmatrix} 1&1&0&0&0&0&0&0\\ 0&-1&1&0&0&1&0&0\\ 0&-1&0&1&0&0&1&0\\ 0&-1&0&0&1&0&0&1\\ 0&1&1&0&0&-1&0&0\\ 0&1&0&1&0&0&-1&0\\ 0&1&0&0&1&0&0&-1 \end{bmatrix}, \tag{115}$$

$$\boldsymbol{D}_{10} = \operatorname{diag}\left(1, 1, s_0^{(7)}, s_1^{(7)}, \ldots, s_7^{(7)}\right). \tag{116}$$

Stage 5. In the formula (111), which determines the F_7 matrix, an adjustment can be made to reduce the number of additions and matrices.

One such possibility is to decompose each of the $\boldsymbol{T}_{7\times 8}$ and $\boldsymbol{T}_{8\times 7}$ matrices into

$$\boldsymbol{T}_{7\times 8} = \boldsymbol{T}_7 \boldsymbol{T}_{7\times 8}^{(1)}, \quad \boldsymbol{T}_{8\times 7} = \boldsymbol{T}_{8\times 7}^{(1)} \boldsymbol{T}_7^{(0)}, \tag{117}$$

defined as follows:

$$\boldsymbol{T}_{7\times 8}^{(1)} = \begin{bmatrix} 1&1&0&0&0&0&0&0\\ 0&1&0&0&0&-1&0&0\\ 0&0&1&0&0&0&0&0\\ 0&1&0&0&0&0&-1&0\\ 0&0&0&1&0&0&0&0\\ 0&1&0&0&0&0&0&-1\\ 0&0&0&0&1&0&0&0 \end{bmatrix}, \tag{118}$$

$$\boldsymbol{T}_7 = \begin{bmatrix} 1&0&0&0&0&0&0\\ 0&-1&1&0&0&0&0\\ 0&0&0&-1&1&0&0\\ 0&0&0&0&0&-1&1\\ 0&1&1&0&0&0&0\\ 0&0&0&1&1&0&0\\ 0&0&0&0&0&1&1 \end{bmatrix}, \tag{119}$$

$$\boldsymbol{T}_7^{(0)} = \begin{bmatrix} \boldsymbol{I}_3 & \boldsymbol{0}_{3\times 1} & -\boldsymbol{I}_3\\ \boldsymbol{0}_{1\times 3} & 1 & \boldsymbol{0}_{1\times 3}\\ \boldsymbol{I}_3 & \boldsymbol{0}_{3\times 1} & \boldsymbol{0}_3 \end{bmatrix}, \tag{120}$$

$$\boldsymbol{T}_{8\times 7}^{(1)} = \begin{bmatrix} 1&1&1&0&0&0&0\\ 0&0&0&1&0&0&0\\ 0&0&0&0&1&0&0\\ 0&0&0&0&0&1&0\\ 0&0&0&0&0&0&1\\ 1&0&0&0&0&0&0\\ 0&1&0&0&0&0&0\\ 0&0&1&0&0&0&0 \end{bmatrix}. \tag{121}$$

Further reduction in the number of additions can be achieved by applying the following substitutions:

$$\boldsymbol{T}_{10\times 8}^{(1)} \boldsymbol{T}_{8\times 7}^{(1)} = \boldsymbol{T}_{10\times 9} \boldsymbol{T}_{9\times 7}, \tag{122}$$

$$\boldsymbol{T}_{7\times 8}^{(1)} \boldsymbol{T}_{8\times 10}^{(0)} = \boldsymbol{T}_{7\times 8}^{(2)} \boldsymbol{T}_{8\times 10}^{(1)}, \tag{123}$$

where

$$\boldsymbol{T}_{9\times 7} = \begin{bmatrix} 1&1&1&0&0&0&0\\ 0&0&0&1&0&0&0\\ 0&0&0&0&-1&1&1\\ 0&0&0&0&0&1&-1\\ 0&0&0&0&-1&-1&0\\ 0&0&0&0&-1&0&-1\\ 0&1&-1&0&0&0&0\\ 1&-1&0&0&0&0&0\\ 1&0&-1&0&0&0&0 \end{bmatrix}, \tag{124}$$

$$\boldsymbol{T}_{10\times 9} = \begin{bmatrix} 1&0&0&0&0&0&0&0&0\\ 0&1&0&0&0&0&0&0&0\\ 0&0&1&0&0&0&0&0&0\\ 0&0&0&1&0&0&0&0&0\\ 0&0&0&0&1&0&0&0&0\\ 0&0&0&0&0&1&0&0&0\\ 1&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&1&0&0\\ 0&0&0&0&0&0&0&1&0\\ 0&0&0&0&0&0&0&0&1 \end{bmatrix}, \tag{125}$$

$$\boldsymbol{T}_{8\times 10}^{(1)} = \begin{bmatrix} 1&1&0&0&0&0&0&0&0&0\\ 0&1&0&0&0&0&-1&0&0&0\\ 0&0&0&0&0&0&0&0&-1&-1\\ 0&0&-1&0&-1&-1&0&0&0&0\\ 0&0&0&0&0&0&0&-1&0&1\\ 0&0&1&1&0&-1&0&0&0&0\\ 0&0&0&0&0&0&0&1&1&0\\ 0&0&1&-1&-1&0&0&0&0&0 \end{bmatrix}, \tag{126}$$

$$\boldsymbol{T}_{7\times 8}^{(2)} = \begin{bmatrix} 1&0&0&0&0&0&0&0\\ 0&1&1&0&0&0&0&0\\ 0&0&0&1&0&0&0&0\\ 0&1&0&0&1&0&0&0\\ 0&0&0&0&0&1&0&0\\ 0&1&0&0&0&0&1&0\\ 0&0&0&0&0&0&0&1 \end{bmatrix}. \tag{127}$$

In addition, the permutation matrix can be reduced using the following substitutions:

$$\boldsymbol{T}_7^{(1)} = \boldsymbol{T}_7^{(0)}\boldsymbol{P}_7^{(0)}, \quad \boldsymbol{T}_7^{(2)} = \boldsymbol{P}_7^{(1)}\boldsymbol{T}_7. \tag{128}$$

The final procedure for calculating the seven-point DFT-II takes the form

$$\boldsymbol{Y}_{7\times1} = \boldsymbol{T}_7^{(2)}\boldsymbol{T}_{7\times8}^{(2)}\boldsymbol{T}_{8\times10}^{(1)}\boldsymbol{D}_{10}\boldsymbol{T}_{10\times9}\boldsymbol{T}_{9\times7}\boldsymbol{T}_7^{(1)}\boldsymbol{X}_{7\times1}. \tag{129}$$

The calculation of a seven-point DFT-II by (129) requires only eight multiplications and 36 additions. The proposed algorithm saves 28 multiplications and six additions compared to the direct DFT-II determination method (82).

Figure 5 shows the signal flow graph of the proposed algorithm for implementing the seven-point DFT-II.

3.6. Algorithm for eight-point DFT-II calculation. Let $\boldsymbol{X}_{8\times1} = [x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7]^{\mathrm{T}}$ be an eight-dimensional data vector and $\boldsymbol{Y}_{8\times1} = [y_0, y_1, y_2, y_3, y_4, y_5, y_6, y_7]^{\mathrm{T}}$ be an output vector. The problem is to calculate a vector-matrix product:

$$\boldsymbol{Y}_{8\times1} = \boldsymbol{F}_8\boldsymbol{X}_{8\times1}, \tag{130}$$

where

$$\boldsymbol{F}_8 = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ a^{(8)} & c^{(8)} & -d^{(8)} & -g^{(8)} & -a^{(8)} & -c^{(8)} & d^{(8)} & g^{(8)} \\ b^{(8)} & -f^{(8)} & -b^{(8)} & f^{(8)} & b^{(8)} & -f^{(8)} & -b^{(8)} & f^{(8)} \\ c^{(8)} & -a^{(8)} & g^{(8)} & -d^{(8)} & -c^{(8)} & a^{(8)} & -g^{(8)} & d^{(8)} \\ -j & j & -j & j & -j & j & -j & j \\ -d^{(8)} & g^{(8)} & -a^{(8)} & c^{(8)} & d^{(8)} & -g^{(8)} & a^{(8)} & -c^{(8)} \\ -f^{(8)} & b^{(8)} & f^{(8)} & -b^{(8)} & -f^{(8)} & b^{(8)} & f^{(8)} & -b^{(8)} \\ -g^{(8)} & -d^{(8)} & c^{(8)} & a^{(8)} & g^{(8)} & d^{(8)} & -c^{(8)} & -a^{(8)} \end{bmatrix}, \tag{131}$$

and

$$\begin{aligned} a^{(8)} &= e^{-j\pi/8}, & b^{(8)} &= e^{-j\pi/4}, \\ c^{(8)} &= e^{-j\pi3/8}, & d^{(8)} &= -e^{-j\pi5/8}, \\ f^{(8)} &= -e^{-j\pi3/4}, & g^{(8)} &= -e^{-j\pi7/8}. \end{aligned} \tag{132}$$

The direct calculation of Eqn. (130) requires 48 complex multiplications and 56 additions.

Stage 1. In the matrix $\boldsymbol{F}_8$, the order of the rows can be rearranged. By applying appropriate permutation matrices, a new matrix $\boldsymbol{F}_8^{(0)}$ is obtained, which satisfies the following relation:

$$\boldsymbol{F}_8 = \boldsymbol{P}_8\boldsymbol{F}_8^{(0)}, \tag{133}$$

where

$$\boldsymbol{F}_8^{(0)} = \left[\begin{array}{cccc:cccc} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ -j & j & -j & j & -j & j & -j & j \\ b^{(8)} & -f^{(8)} & -b^{(8)} & f^{(8)} & b^{(8)} & -f^{(8)} & -b^{(8)} & f^{(8)} \\ -f^{(8)} & b^{(8)} & f^{(8)} & -b^{(8)} & -f^{(8)} & b^{(8)} & f^{(8)} & -b^{(8)} \\ \hdashline a^{(8)} & c^{(8)} & -d^{(8)} & -g^{(8)} & -a^{(8)} & -c^{(8)} & d^{(8)} & g^{(8)} \\ c^{(8)} & -a^{(8)} & g^{(8)} & -d^{(8)} & -c^{(8)} & a^{(8)} & -g^{(8)} & d^{(8)} \\ -d^{(8)} & g^{(8)} & -a^{(8)} & c^{(8)} & d^{(8)} & -g^{(8)} & a^{(8)} & -c^{(8)} \\ -g^{(8)} & -d^{(8)} & c^{(8)} & a^{(8)} & g^{(8)} & d^{(8)} & -c^{(8)} & -a^{(8)} \end{array}\right], \tag{134}$$

$$\boldsymbol{P}_8 = \begin{bmatrix} 1&0&0&0&0&0&0&0 \\ 0&0&0&0&1&0&0&0 \\ 0&0&1&0&0&0&0&0 \\ 0&0&0&0&0&1&0&0 \\ 0&1&0&0&0&0&0&0 \\ 0&0&0&0&0&0&1&0 \\ 0&0&0&1&0&0&0&0 \\ 0&0&0&0&0&0&0&1 \end{bmatrix}. \tag{135}$$

Stage 3. The structure of the $\boldsymbol{F}_8^{(0)}$ matrix enables immediate extraction of submatrices for factorization, allowing Stage 2 of constructing a fast DFT-II algorithm to be skipped. The matrix $\boldsymbol{F}_8^{(0)}$ exhibits the following structure:

$$\boldsymbol{F}_8^{(0)} = \begin{bmatrix} \boldsymbol{F}_4^{(2)} & \boldsymbol{F}_4^{(2)} \\ \boldsymbol{F}_4^{(3)} & -\boldsymbol{F}_4^{(3)} \end{bmatrix}, \tag{136}$$

where

$$\boldsymbol{F}_4^{(2)} = \left[\begin{array}{cc:cc} 1 & 1 & 1 & 1 \\ -j & j & -j & j \\ \hdashline b^{(8)} & -f^{(8)} & -b^{(8)} & f^{(8)} \\ -f^{(8)} & b^{(8)} & f^{(8)} & -b^{(8)} \end{array}\right], \tag{137}$$

$$\boldsymbol{F}_4^{(3)} = \begin{bmatrix} a^{(8)} & c^{(8)} & -d^{(8)} & -g^{(8)} \\ c^{(8)} & -a^{(8)} & g^{(8)} & -d^{(8)} \\ -d^{(8)} & g^{(8)} & -a^{(8)} & c^{(8)} \\ -g^{(8)} & -d^{(8)} & c^{(8)} & a^{(8)} \end{bmatrix}. \tag{138}$$

Thus, the matrix $\boldsymbol{F}_8^{(0)}$ can be preformed as

$$\boldsymbol{F}_8^{(0)} = \boldsymbol{F}_8^{(1)}\boldsymbol{T}_8, \tag{139}$$

where

$$\boldsymbol{T}_8 = \boldsymbol{H}_2 \otimes \boldsymbol{I}_4, \quad \boldsymbol{F}_8^{(1)} = \left[\boldsymbol{F}_4^{(2)} \oplus \boldsymbol{F}_4^{(3)}\right]. \tag{140}$$

The $\boldsymbol{F}_4^{(2)}$ matrix has a similar structure to $\boldsymbol{F}_8^{(0)}$ (136), i.e.,

$$\boldsymbol{F}_4^{(2)} = \begin{bmatrix} \boldsymbol{F}_2^{(9)} & \boldsymbol{F}_2^{(9)} \\ \boldsymbol{F}_2^{(10)} & -\boldsymbol{F}_2^{(10)} \end{bmatrix}, \tag{141}$$

where

$$\boldsymbol{F}_2^{(9)} = \begin{bmatrix} 1 & 1 \\ -j & j \end{bmatrix}, \quad \boldsymbol{F}_2^{(10)} = \begin{bmatrix} b^{(8)} & -f^{(8)} \\ -f^{(8)} & b^{(8)} \end{bmatrix}. \tag{142}$$

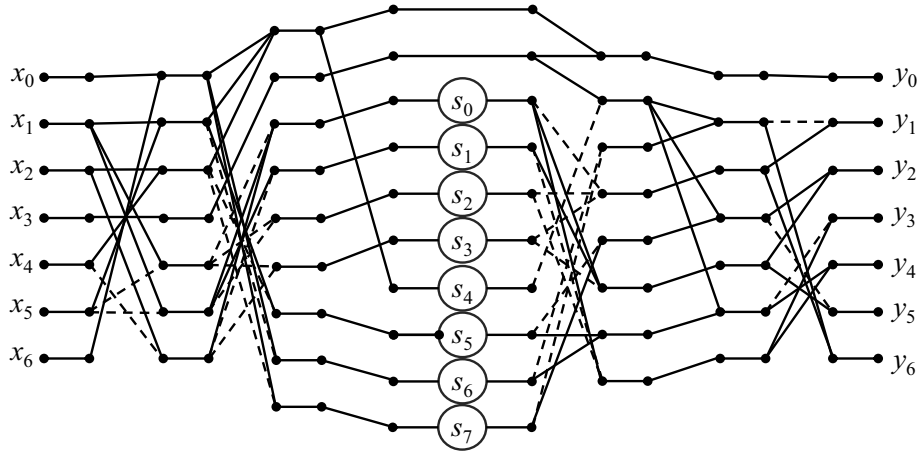

Fig. 5. Signal flow graph for the computation of the seven-point DFT-II (129).

Thus, the matrix $\boldsymbol{F}_4^{(2)}$ can be replaced by the expression

$$\boldsymbol{F}_4^{(2)} = \boldsymbol{F}_4^{(4)}\boldsymbol{T}_4^{(3)}, \quad \boldsymbol{F}_4^{(4)} = \left[\boldsymbol{F}_2^{(9)} \oplus \boldsymbol{F}_2^{(10)}\right]. \quad (143)$$

Similarly, we can perform a permutation of the rows and columns of a matrix $\boldsymbol{F}_4^{(3)}$:

$$\boldsymbol{F}_4^{(3)} = \boldsymbol{P}_4\boldsymbol{F}_4^{(5)}\boldsymbol{P}_4, \quad (144)$$

where

$$\boldsymbol{F}_4^{(5)} = \left[\begin{array}{cc:cc} a^{(8)} & c^{(8)} & -g^{(8)} & -d^{(8)} \\ c^{(8)} & -a^{(8)} & -d^{(8)} & g^{(8)} \\ \hdashline -g^{(8)} & -d^{(8)} & a^{(8)} & c^{(8)} \\ -d^{(8)} & g^{(8)} & c^{(8)} & -a^{(8)} \end{array}\right], \quad (145)$$

$$\boldsymbol{P}_4 = \boldsymbol{I}_2 \oplus \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}. \quad (146)$$

The matrix $\boldsymbol{F}_4^{(5)}$ has a structure similar to the matrix $\boldsymbol{F}_6^{(0)}$ (92):

$$\boldsymbol{F}_4^{(5)} = \begin{bmatrix} \boldsymbol{F}_2^{(11)} & \boldsymbol{F}_2^{(12)} \\ \boldsymbol{F}_2^{(12)} & \boldsymbol{F}_2^{(11)} \end{bmatrix}, \quad (147)$$

where

$$\boldsymbol{F}_2^{(11)} = \begin{bmatrix} a^{(8)} & c^{(8)} \\ c^{(8)} & -a^{(8)} \end{bmatrix}, \ \boldsymbol{F}_2^{(12)} = \begin{bmatrix} -g^{(8)} & -d^{(8)} \\ -d^{(8)} & g^{(8)} \end{bmatrix}, \quad (148)$$

and can be replaced by

$$\boldsymbol{F}_4^{(5)} = \boldsymbol{T}_4^{(3)}\boldsymbol{F}_4^{(6)}\boldsymbol{T}_4^{(3)}, \quad \boldsymbol{F}_4^{(6)} = \left[\boldsymbol{F}_2^{(13)} \oplus \boldsymbol{F}_2^{(14)}\right], \quad (149)$$

where

$$\boldsymbol{F}_2^{(13)} = \frac{1}{2}\left(\boldsymbol{F}_2^{(11)} + \boldsymbol{F}_2^{(12)}\right) = \frac{1}{2}\begin{bmatrix} v^{(0)} & v^{(1)} \\ v^{(1)} & -v^{(0)} \end{bmatrix}, \quad (150)$$

$$\boldsymbol{F}_2^{(14)} = \frac{1}{2}\left(\boldsymbol{F}_2^{(11)} - \boldsymbol{F}_2^{(12)}\right) = \frac{1}{2}\begin{bmatrix} v^{(2)} & v^{(3)} \\ v^{(3)} & -v^{(2)} \end{bmatrix}, \quad (151)$$

$$v^{(0)} = a^{(8)} - g^{(8)}, \quad v^{(1)} = c^{(8)} - d^{(8)}, \quad (152)$$

$$v^{(2)} = a^{(8)} + g^{(8)}, \quad v^{(3)} = c^{(8)} + d^{(8)}. \quad (153)$$

The matrices $\boldsymbol{F}_2^{(13)}$ and $\boldsymbol{F}_2^{(14)}$ share a structure similar to that of the matrix $\boldsymbol{F}_2^{(6)}$ (46) and can be substituted by equations analogous to (48):

$$\begin{aligned}\boldsymbol{F}_2^{(13)} = \boldsymbol{T}_{2\times 3}\frac{1}{2}\Big[&\left(v^{(0)} - v^{(1)}\right) \\ &\oplus \left(-v^{(0)} - v^{(1)}\right) \oplus v^{(1)}\Big]\boldsymbol{T}_{3\times 2},\end{aligned} \quad (154)$$

$$\begin{aligned}\boldsymbol{F}_2^{(14)} = \boldsymbol{T}_{2\times 3}\frac{1}{2}\Big[&\left(v^{(2)} - v^{(3)}\right) \\ &\oplus \left(-v^{(2)} - v^{(3)}\right) \oplus v^{(3)}\Big]\boldsymbol{T}_{3\times 2},\end{aligned} \quad (155)$$

where the matrices $\boldsymbol{T}_{2\times 3}$ and $\boldsymbol{T}_{3\times 2}$ are defined as in (49).

Stage 4. Taking the above modifications into account, the eight-point DFT-II matrix factorization can be represented as

$$\boldsymbol{Y}_{8\times 1} = \boldsymbol{T}_8^{(2)}\boldsymbol{T}_{8\times 10}^{(2)}\boldsymbol{D}_{10}\boldsymbol{T}_{10\times 8}^{(2)}\boldsymbol{T}_8^{(1)}\boldsymbol{T}_8^{(0)}\boldsymbol{X}_{7\times 1}, \quad (156)$$

where

$$\boldsymbol{T}_8^{(0)} = \begin{bmatrix} \boldsymbol{I}_4 & \boldsymbol{I}_4 \\ \boldsymbol{T}_4 & -\boldsymbol{T}_4 \end{bmatrix}, \quad \boldsymbol{T}_4 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}, \quad (157)$$

$$\boldsymbol{T}_8^{(1)} = (\boldsymbol{H}_2 \otimes \boldsymbol{I}_2) \oplus (\boldsymbol{H}_2 \otimes \boldsymbol{I}_2), \quad (158)$$

$$\boldsymbol{T}_{10\times 8}^{(2)} = \boldsymbol{H}_2 \oplus \boldsymbol{H}_2 \oplus \boldsymbol{T}_{3\times 2} \oplus \boldsymbol{T}_{3\times 2}, \quad (159)$$

$$\boldsymbol{T}_{8\times 10}^{(2)} = \boldsymbol{I}_2 \oplus \boldsymbol{H}_2 \oplus \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix} \oplus \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}, \quad (160)$$

$$\boldsymbol{T}_8^{(2)} = \begin{bmatrix} 1&0&0&0&0&0&0&0 \\ 0&0&0&0&1&0&1&0 \\ 0&0&1&0&0&0&0&0 \\ 0&0&0&0&0&1&0&1 \\ 0&1&0&0&0&0&0&0 \\ 0&0&0&0&0&1&0&-1 \\ 0&0&0&1&0&0&0&0 \\ 0&0&0&0&1&0&-1&0 \end{bmatrix}, \tag{161}$$

$$\boldsymbol{D}_{10} = \operatorname{diag}\left(1, -j, s_0^{(7)}, s_1^{(7)}, \ldots, s_7^{(7)}\right), \tag{162}$$

and

$$\begin{aligned} s_0^{(8)} &= \frac{1}{2}\left(b^{(8)} - f^{(8)}\right) = -j\cos(\pi/4), \\ s_1^{(8)} &= \frac{1}{2}\left(b^{(8)} + f^{(8)}\right) = \cos(\pi/4), \\ s_2^{(8)} &= \frac{1}{2}\left(a^{(8)} - c^{(8)} + d^{(8)} - g^{(8)}\right) \\ &= j\sqrt{2}\cos\left(3\pi/8\right), \\ s_3^{(8)} &= \frac{1}{2}\left(-a^{(8)} - c^{(8)} + d^{(8)} + g^{(8)}\right) \\ &= j\sqrt{2}\cos\left(-\pi/8\right), \\ s_4^{(8)} &= \frac{1}{2}\left(c^{(8)} - d^{(8)}\right) = \cos(\pi/8), \\ s_5^{(8)} &= \frac{1}{2}\left(a^{(8)} - c^{(8)} - d^{(8)} + g^{(8)}\right) \\ &= \sqrt{2}\cos\left(3\pi/8\right), \\ s_6^{(8)} &= \frac{1}{2}\left(-a^{(8)} - c^{(8)} - d^{(8)} - g^{(8)}\right) \\ &= -\sqrt{2}\cos\left(-\pi/8\right), \\ s_7^{(8)} &= \frac{1}{2}\left(c^{(8)} + d^{(8)}\right) = \sin(\pi/8). \end{aligned} \tag{163}$$

Further refinement of the obtained factorization (156) is unnecessary, so Stage 5 is omitted.

The calculation of the eight-point DFT-II by (130) requires only eight multiplications and 32 additions. If one were to take into account that one complex multiplication requires at least three multiplications on real numbers and a complex addition requires two additions on real numbers, the proposed algorithm reduces to 136 multiplications and 80 additions on real numbers.

Figure 6 shows the signal flow graph of the proposed algorithm to implement the eight-point DFT-II.

4. FPGA implementation

The algorithms proposed in this paper were implemented on Xilinx Spartan 3 or Spartan 6 FPGAs. The principle of implementing the algorithms on the simplest possible FPGA chip was adopted. Assuming all input and output signals appear in parallel, the number of available inputs/outputs proved to be the most significant constraint in selecting a particular FPGA chip. It was assumed that the inputs and constants were 8 bits and the number of bits in the outputs was appropriate to the order of the DFT-II, i.e., for $N = 3$, the output would be 10 bits, for $N = 4$, the output would be 11 bits, and so on. Algorithms of the DFT-II were implemented on an FPGA using fixed-point arithmetic in the Q1.7 format for both inputs and coefficients. Internal coefficients s_k were pre-scaled to satisfy the condition $|s_k| \leq 1$ in order to avoid saturation.

Bit growth was managed by increasing the bit width by one for each addition and by summing the operand widths for multiplication operations. Final outputs were quantized via bit-shifting to the appropriate bit format, which limits cumulative quantization error and achieves precision comparable to the DMVP, while maintaining efficient hardware utilization.

A direct implementation of the DFT-II algorithm was performed using the definition of matrix-vector multiplication, assuming that all multiplication operations are performed in parallel.

The algorithms implemented in Verilog were synthesised using Xilinx Project Navigator Release Version 14.7. The implementation was carried out with two primary design objectives: power optimisation using "strategy 1" and timing performance using the strategy "performance with IOB packing".

5. Results

5.1. Evaluation of the computational complexity of the proposed fast DFT-II algorithms. Table 1 presents the number of complex multiplication and addition operations required for both the DMVP and the proposed algorithm. In every instance, a substantial decrease in the number of multiplications was achieved, along with some reduction in additions. Reducing the number of multiplications is especially crucial, as multipliers are the most resource-intensive components in terms of both space and energy consumption.

The same table presents the number of multiplications and additions performed on real numbers for both algorithms. For the DMVP, the number of these operations is significantly higher than those listed in Table 1, whereas for the proposed algorithm, these quantities only double due to the complex values of the input signals. This demonstrates that the proposed algorithms enable a much more significant reduction in the operations required for practical implementation.

5.2. Discussion of the quantization effects (fixed-point vs. floating-point) and error analysis. The main sources of errors in the implemented DFT-II algorithms arise from coefficient quantization, input signal quantization, fixed-point arithmetic, and error propagation along the critical paths.

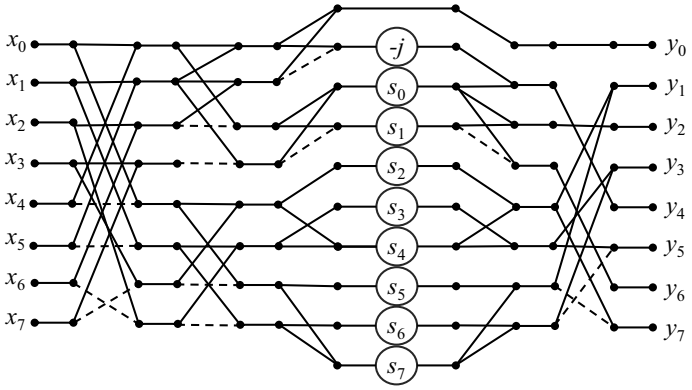

Fig. 6. Data flow graph for the computation of the eight-point DFT-II by a computationally efficient algorithm (130).

Coefficient quantization is addressed by pre-scaling all internal coefficients to ensure they remain within the representable Q1.7 range, thereby preventing saturation and reducing systematic quantization errors that would otherwise occur.

Input signal quantization, inherent to the 8-bit Q1.7 format, introduces a fixed initial error that is independent of the algorithm chosen.

Errors related to fixed-point arithmetic are mitigated through careful bit-growth management: each addition increases the accumulator width by one bit, while multiplication results are stored using the sum of operand bit widths.

Furthermore, the factorized DFT implementation limits error propagation along critical paths, as each path contains only a single multiplication and fewer operations compared to the DMVP.

Final results are quantized to the appropriate output bit format using bit-shifting, ensuring minimal cumulative error while preserving precision and maintaining efficient hardware utilization on an FPGA.

Adopting floating-point arithmetic in FPGA implementations of both the proposed DFT-II algorithms and the DMVP can significantly reduce quantization errors inherent to fixed-point representations. In such implementations, input signals and coefficients benefit from a wider dynamic range and higher precision, eliminating the need for coefficient pre-scaling to prevent saturation. Moreover, the bit-growth management techniques typically employed in fixed-point implementations—such as increasing accumulator width for additions or calculating the combined operand widths for multiplications—are rendered unnecessary, since floating-point units inherently accommodate dynamic range variations.

In the proposed DFT-II algorithms, the critical paths contain fewer multiplications and additions. As a result, a floating-point implementation may achieve lower latency and slightly reduced resource consumption compared to the DMVP, which requires performing all N^2 complex multiplications. Additionally, the structural approach inherent to the proposed algorithms facilitates parallelization on FPGA platforms. This contrasts with the DMVP, which relies on dense matrix operations and would require more general-purpose floating-point units to handle the computational load efficiently.

5.3. Approaches combining the proposed algorithms for a large-sized transforms. The proposed algorithms for a small-sized DFT-II can be extended to compute large-size transforms through several approaches.

Firstly, an algebraic approach can be employed, which combines terms directly from the analytical expression of the DFT-II. This method decomposes a large DFT-II computation into smaller subtasks. For example, as discussed in Section 1.1, Saleh *et al.* (2003) developed an algorithm for computing an $N = 2^m$-point DFT-II, where $m \geq 4$ is a positive integer. This algorithm utilizes computations of small-sized DFT-II's at sizes $N/2$, $N/4$, and $N/8$ to calculate the large-sized DFT-II with an FPGA implementation. However, a limitation of this approach is that it applies only to input data sizes that are powers of two.

Secondly, an approach based on matrix operations, such as the Kronecker product, can be applied to construct large-sized DFT-II algorithms from smaller-sized transform algorithms (Zhang *et al.*, 2015). The Kronecker product combines two matrices into a larger block matrix by multiplying each element of the first matrix by the entire second one. These products preserve matrix orthogonality, which is critical in many signal processing and communication applications.

Nonetheless, the matrix-based approach yields transforms that resemble but do not exactly equate to the direct DFT-II. Moreover, in data-dependent applications, designing or optimizing algorithms using these products can be challenging due to the difficulty in selecting appropriate parameter values at smaller matrix levels.

Additionally, interpreting the results from the resulting transforms is not always straightforward, which may complicate their practical use.

5.4. Resource and power efficiency in FPGA implementations. The implementation on Spartan 3 and Spartan 6 family FPGAs enabled verification of the proposed algorithms effectiveness compared to the DMVP. Utilizing the "power optimisation" strategy facilitated a comparison of implementations that minimise the energy consumption of the chip. This approach is especially crucial for embedded systems, such as Internet of things (IoT) devices.

When analysing the obtained results, it is essential to consider the power requirements and the amount of space occupied by the individual logic circuits.

The registers, or flip-flops, consume a relatively small amount of power. Their power consumption is mainly related to the switching of logic states. They also cover a very small surface area in the FPGA structure, occupying only a small part of the FPGA's resources compared to other elements such as look-up tables (LUTs) or multiplication modules MULT18X18S.

For this reason, it is important to minimise MULT18X18S multiplication circuits, and it is preferable to use more registers than LUTs.

LUTs consume more power than flip-flops because of their combinational function, where they implement logic based on several inputs. The energy consumed by LUTs depends on the number of switching inputs and the complexity of the logic function they implement. LUTs occupy a larger area than registers but can still be considered relatively small.

MULT18X18S multiplication modules consume significantly more power than both registers and LUTs. Multiplication is a logic resource-intensive operation, which translates into higher power consumption, especially at high operating frequencies. At the same time, they take up significantly more space in the FPGA than registers and LUTs because they are large, specialised logic blocks designed to perform arithmetic operations that are much more complex than the functions performed by standard LUTs or registers.

Table 2 shows the use of MULT18X18S. The reduction in the number of the blocks used is significant due to the FPGA chip's space occupancy and power consumption. For each DFT-II order, using the proposed algorithms allows a substantial reduction in MULT18X18S concerning the implementation the DMVP.

Table 2 shows the number of slices used to implement the DMVP and the proposed algorithms, clearly detailing the number of registers and LUTs used. In most cases, the number of employed registers is slightly higher for the proposed algorithms, but that of LUTs is

Table 1. Comparison of the number of complex and real multiplications, as well as complex and real additions between the DMVP and the proposed algorithm.

DFT-II size	Complex multiplications			Real multiplications			Complex additions			Real additions		
	DMVP	Proposed	Decrease	DMVP	Proposed	Decrease	DMVP	Proposed	Decrease	DMVP	Proposed	Decrease
3	4	1	3	16	2	14	6	6	0	14	12	2
4	8	2	6	24	4	20	12	10	2	28	20	8
5	16	5	64	10	54	11	20	18	2	52	36	16
6	16	2	14	64	4	60	30	24	6	62	48	14
7	36	8	28	144	16	128	42	36	6	114	72	42
8	48	8	40	192	16	176	56	32	24	152	64	88

Table 2. Number of MULT18X18S, registers and LUTs used in the DMVP and the proposed algorithm.

DFT-II size	Targed device	Number of MULT18X18S			Number of registers			Number of LUTs		
		DMVP	Proposed	Decrease	DMVP	Proposed	Increase	DMVP	Proposed	Decrease
3	xc3s50-5pq208	4	2	2	140	184	44	268	188	80
4	xc3s200-5ft256	12	4	8	204	225	21	796	285	511
5	xc3s400-5fg456	16	10	6	581	624	43	2064	783	1281
6	xc3s400-5fg456	16	4	12	316	572	256	1364	767	597
7	xc6slx45-2fgg676	56	4	52	879	1103	224	4456	1853	2603
8	xc6slx75-2fgg676	48	16	32	859	692	-167	5134	850	4284

always significantly lower. This reduction in the use of LUTs is beneficial. The registers take up less space in the circuit and consume less power, making the proposed algorithms more efficient regarding resource use and power consumption.

Table 3 summarizes the maximum operating frequency of an FPGA for the DMVP and the proposed algorithms across different DFT-II orders, along with the absolute and percentage improvements. The proposed algorithms consistently outperform the DMVP, achieving higher maximum frequencies for all DFT-II orders. The absolute frequency increase ranges significantly, with the percentage improvement varying from 45% to 178%. Notably, the largest percentage gains are observed at DFT-II orders 5 and 6, with 178% and 173% improvements, respectively, demonstrating the scalability and effectiveness of the proposed algorithms as the DFT-II order increases. Even at the highest order, i.e., (8), the proposed algorithm achieves a substantial improvement of 71%, confirming its robust performance across varying conditions.

5.5. Applying the proposed DFT-II algorithms in the OFDM system. The proposed algorithms and their FPGA implementation were applied to DFT-II calculation for data processing within the OFDM system. In such systems, a high-rate data stream is divided into multiple lower-rate ones, each transmitted simultaneously over different orthogonal subcarriers through the following steps (Khosla *et al.*, 2017):

1. OFDM partitions the available bandwidth into multiple closely spaced subcarriers, with frequencies precisely chosen to ensure mathematical orthogonality.
2. To enable parallel data transmission, the original data stream is divided into parallel ones, each modulating a distinct subcarrier.
3. The frequency-domain symbols assigned to each subcarrier are converted into the time domain using the inverse DFT.
4. A cyclic prefix, which is a copy of the final portion of the OFDM symbol, is appended to each symbol to mitigate intersymbol interference caused by multipath propagation. This converts the linear convolution of the channel into a circular convolution, thereby simplifying equalization.
5. The time-domain OFDM symbols with the cyclic prefix are serialized and transmitted. At the receiver, the cyclic prefix is removed, and the direct DFT converts the signal back into individual subcarrier frequency components to recover the transmitted data symbols.

Table 3. Maximum operating frequency of an FPGA with the DMVP and the proposed algorithms.

DFT-II size	DMVP	Proposed	Increase	Increase %
3	122.593	177.275	54.682	45%
4	77.965	174.005	96.04	123%
5	62.497	173.983	111.486	178%
6	55.435	151.096	95.661	173%
7	109.625	189.952	80.327	73%
8	108.178	184.851	76.673	71%

As mentioned, the DFT-II is better suited for processing bit streams of odd length with a symmetrical structure, in contrast to the standard (even-length) DFT (Britanak and Rao, 1999). In our experiment, we used DFTs of odd lengths (5, 7, and 63 samples) as well as typical even power-of-two lengths (8, 16, and 64 samples). Randomly and uniformly distributed binary signals were generated and modulated using M-ary quadrature amplitude modulation (QAM) with $M = 8$, commonly referred to as 8-QAM. The modulated symbols were then mapped into OFDM subcarriers. The signals were processed using type I–IV inverse DFTs of the selected lengths, a cyclic prefix was added, and the resulting sequences were transmitted through a channel with additive white Gaussian noise (AWGN). On the receiver side, after cyclic prefix removal, the corresponding type I–IV direct DFTs and 8-QAM demodulation were applied to recover the transmitted data symbols.

Channel estimation and equalization were not performed in order to study the efficiency of the different DFT types. The application of these transforms was analyzed in AWGN channels, as well as in channels with phase noise and AWGN. Phase noise is a multiplicative noise resulting from the jitter of the local oscillator in the OFDM system. This noise was introduced into the carrier signal after prepending the cyclic prefix to each symbol. Then, the received signal with the effect of phase noise, $z(t)$, was expressed as $z(t) = y(t)e^{i\phi(t)}$, where $y(t)$ is the received signal and $\phi(t)$ is a random process. The process $\phi(t)$ is described by the following equation (Syrjala *et al.*, 2009): $\phi(t) = cB(t)$, where $c = \sqrt{2\pi\beta T_s}$. Here, β and T_s are the phase noise parameters, specifically the phase noise linewidth and the sampling period, respectively, while $B(t)$ denotes a Wiener process, for which the differences between neighboring samples are i.i.d. Gaussian distributed with zero mean and unit variance.

In our experiments, the number of symbols chosen was 100, $c = 0.0177$, and the number of signals for averaging was also 100. The mean phase noise level density was -93 dBc/Hz.

Finally, the bit-error rate (BER) was calculated for a range of values of E_b/N_0. The ratio E_b/N_0 represents the energy per bit (E_b) relative to the noise power spectral density (N_0), and it is commonly used to evaluate signal quality in the presence of noise (Akkaş, 2017).

Here, E_b denotes the average energy required to transmit one bit of information, which can be interpreted as the signal power divided by the bit rate (in bits per second). The quantity N_0 represents the noise power spectral density, i.e., the noise power present in the channel per unit bandwidth (per Hz). The ratio E_b/N_0 is dimensionless and is typically expressed in decibels (dB).

Figure 7 presents plots of the BER as a function of E_b/N_0 for various signal lengths and cyclic prefix lengths. Let N_F denote the length of the DFT and P_c the length of the cyclic prefix.

Figure 7 shows that the BER initially decreases with increasing E_b/N_0 and subsequently reaches a steady-state value. For the AWGN channel without phase noise, the BER curves corresponding to DFT types I–IV nearly overlap (Fig. 7(a)), and the BER stabilizes close to zero. This behavior indicates that, in the absence of phase noise, the application of these transforms yields comparable data transmission performance.

However, the presence of phase noise in the OFDM system alters this behavior, as the DFT type becomes a significant factor influencing the BER. Figures 7(c)–(h) present the BER versus the E_b/N_0 curves for the AWGN channel with phase noise and without noise compensation. For small N_F, the BER stabilizes not near zero but around a value between 0.04 and 0.08, consistent with previous observations for the type I DFT (Bhatti *et al.*, 2010; Sadinov, 2017). When N_F is odd and symmetric small-sized signals are transmitted, the DFT-II exhibits a lower BER than the other DFT types (Fig. 7(e), (g)), in agreement with prior studies (Britanak and Rao, 1999). Therefore, the proposed DFT-II algorithms and their FPGA implementations are directly applicable to small-sized signals in OFDM systems affected by phase noise. For random (asymmetric) signals or even N_F, alternative DFT types may provide better performance (Fig. 7(b), (d), (f), (h)). As N_F increases, the BER also rises; for instance, with $N_F = 16$, the BER ranges approximately from 0.10 to 0.22 (Fig. 7(b)), whereas for $N_F = 64$, it varies from 0.17 to 0.31. Consequently, the application of DFT types I–IV for large

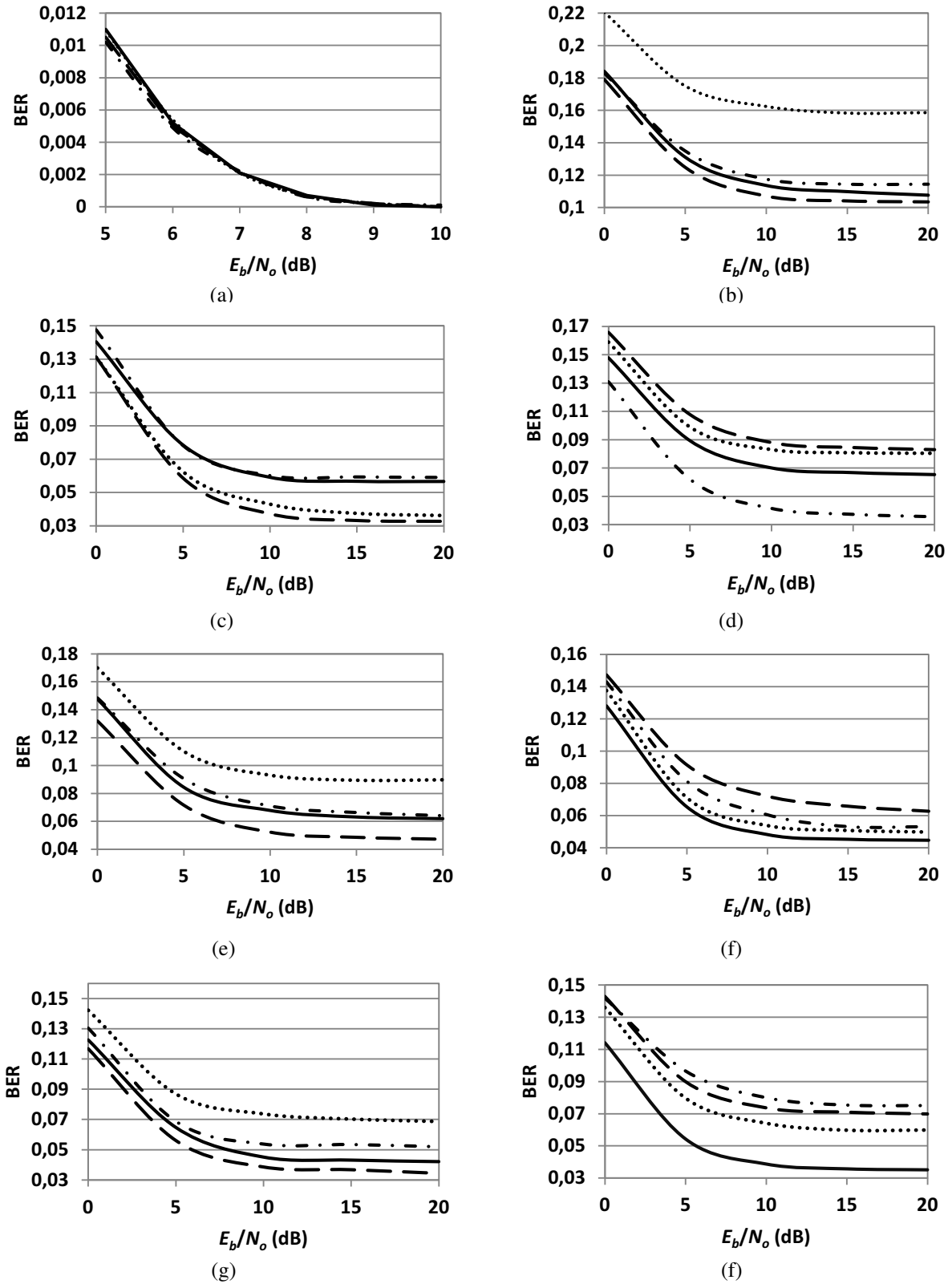

Fig. 7. Plots of the BER versus E_b/N_0 for the type I DFT (solid line), type II DFT (dashed line), type III DFT (dash-dotted line), and type IV DFT (dotted line): $N_F = 7$, $P_c = 4$, random signal (a), $N_F = 16$, $P_c = 8$, symmetrical signal (b), $N_F = 8$, $P_c = 4$, symmetrical signal (c), $N_F = 8$, $P_c = 4$, random signal (d), $N_F = 7$, $P_c = 4$, symmetrical signal (e),x $N_F = 7$, $P_c = 4$, random signal (f), $N_F = 5$, $P_c = 4$, symmetrical signal (g), $N_F = 5$, $P_c = 4$, random signal (h).

N_F is recommended only after phase noise compensation, which can be achieved, for example, by transmitting pilot symbols at known locations to estimate the phase noise, followed by interpolation and filtering to correct the data symbols (Bhatti *et al.*, 2010).

6. Summary

This paper presented odd-time DFT algorithms with a significantly reduced number of multiplications and additions. The developed algorithms were compared in their computational complexity with DMVP computation. In the range of signal lengths from three to eight samples, the proposed DFT-II matrix factorizations reduce the number of complex multiplications by an average of 78% and that of complex additions by an average of 17%. The number of real multiplications is reduced by an average of 88% and that of real additions by an average of 32%.

The proposed fast DFT-II algorithms were implemented on FPGA platforms from Spartan 3 and Spartan 6 families, demonstrating their practical applicability in hardware environments where performance and resource usage are critical. A detailed analysis of the results was focused on the power requirements and the amount of space occupied by the individual logic circuits. The FPGA resource utilization (and the number of consumed logic elements in particular) determines how efficiently the proposed algorithms use the available hardware. So, the number of multiplication modules MULT18X18S is reduced by an average of 65% compared to that of the same blocks required for DMVP implementation. At the same time, the number of logical elements is increased by an average of 23% and that of lookup tables is reduced by an average of 57%. Additionally, the maximum operating frequencies achieved by the FPGA implementations were increased by an average of 110% depending on the DFT-II order. This optimization makes the DFT-II more cost-effective for high-speed signal processing where performance and hardware resource efficiency are critical factors.

The obtained FPGA implementation of small-sized DFT-II algorithms can be applied for training a Fourier series neural network (Halawa, 2008) with low computational complexity. The network mentioned is constructed based on the multidimensional DFT and used for modeling dynamic systems. Also, interpolation-based reconstruction methods in 3D positron emission tomography utilize interpolation techniques to estimate missing or unevenly distributed frequency data. This enables the use of the proposed FPGA implementation of small-sized DFT-II algorithms for image reconstruction (Li *et al.*, 2008).

References

Akkaş, M.A. (2017). Channel modeling of wireless sensor networks in oil, *Wireless Personal Communications* **95**(4): 4337–4355.

Almenar, V., Girona, A., Flores, S. and Marin-Roig, J. (2011). Transmit diversity scheme for OFDM systems using the odd DFT, *IEICE Transactions on Communications* **94**(8): 2411–2413.

Belega, D., Petri, D. and Dallet, D. (2016). Influence of wideband noise on sinusoid parameter estimation provided by the two-point interpolated odd-DFT algorithm, *2016 12th IEEE International Symposium on Electronics and Telecommunications (ISETC), Timisoara, Romania*, pp. 293–297.

Bhatti, J., Noels, N. and Moeneclaey, M. (2010). Phase noise estimation and compensation for OFDM systems: A DCT-based approach, *2010 IEEE 11th International Symposium on Spread Spectrum Techniques and Applications, Taichung, Taiwan*, pp. 93–97.

Bi, G. and Chen, Y. (1998). Fast generalized DFT and DHT algorithms, *Signal Processing* **65**(3): 383–390.

Bi, G. and Zeng, Y. (2012). *Transforms and Fast Algorithms for Signal Analysis and Representations*, Birkhäuser, Boston.

Britanak, V. and Rao, K.R. (1999). The fast generalized discrete Fourier transforms: A unified approach to the discrete sinusoidal transforms computation, *Signal Processing* **79**(2): 135–150.

Cariow, A. (2014). Strategies for the synthesis of fast algorithms for the computation of the matrix-vector products, *Journal of Signal Processing Theory and Applications* **3**(1): 1–19.

Cariow, A. and Paplinski, J.P. (2021). Algorithmic structures for realizing short-length circular convolutions with reduced complexity, *Electronics* **10**(22): 2800.

Dai, Y., Yi, H. and Tao, Y. (2019). An algorithm for linear convolution based on generalized discrete Fourier transform, *Chinese Journal of Engineering Mathematics* **36**(1): 106–114.

Dattoli, G., Palma, E.D., Nguyen, F. and Sabia, E. (2017). Generalized trigonometric functions and elementary applications, *International Journal of Applied and Computational Mathematics* **3**(2): 445–458.

Dun, Y. and Liu, G. (2015). A fine-resolution frequency estimator in the odd-DFT domain, *IEEE Signal Processing Letters* **22**(12): 2489–2493.

Elshirkasi, A.M., Siddiqi, M.U. and Habaebi, M.H. (2014). Generalized discrete Fourier transform based minimization of PAPR in OFDM systems, *2014 International Conference on Computer and Communication Engineering, Kuala Lumpur, Malaysia*, pp. 205–208.

Halawa, K. (2008). Determining the weights of a Fourier series neural network on the basis of the multidimensional discrete Fourier transform, *International Journal of Applied Mathematics and Computer Science* **18**(3): 369–375, DOI: 10.2478/v10006-008-0033-8.

Khosla, D., Singh, S., Singh, R. and Goyal, S. (2017). OFDM modulation technique & its applications: A review, *International Conference on Innovations in Computing (ICIC 2017), Liverpool, UK*, pp. 101–105.

Li, G., Jing, X., Sun, S. and Huang, H. (2014). An improved channel estimation based on generalized DFT for OFDM system, *2014 14th International Symposium on Communications and Information Technologies (ISCIT), Incheon, South Korea*, pp. 341–345.

Li, Y., Kummert, A., Boschen, F. and Herzog, H. (2008). Interpolation-based reconstruction methods for tomographic imaging in 3D positron emission tomography, *International Journal of Applied Mathematics and Computer Science* **18**(1): 63–73, DOI: 10.2478/v10006-008-0006-y.

Mansour, M.F. (2006). On the odd-DFT and its applications to DCT/IDCT computation, *IEEE Transactions on Signal Processing* **54**(7): 2819–2822.

Mazrooei, M., Rahimi, L. and Sahami, N. (2018). Two-dimensional generalized discrete Fourier transform and related quasi-cyclic Reed–Solomon codes, *Turkish Journal of Mathematics* **42**(1): 349–359.

Oraintara, S. (2002). The unified discrete Fourier–Hartley transforms: Theory and structure, *2002 IEEE International Symposium on Circuits and Systems: Proceedings, Phoenix–Scottsdale, USA*, Vol. 3, pp. III–III.

Otunniyi, T.O. and Myburgh, H.C. (2021). Improving generalized discrete Fourier transform (GDFT) filter banks with low-complexity and reconfigurable hybrid algorithm, *Digital* **1**(1): 1–17.

Patil, V., Singh, J. and Tiwari, M. (2012). Optimization of generalized discrete Fourier transform for CDMA, *International Journal of Engineering Research & Technology* **1**(3): 1–7.

Pei, S.-C. and Luo, T.-L. (1996). Split-radix generalized fast Fourier transform, *Signal Processing* **54**(2): 137–151.

Perera, S.M. (2018). Signal flow graph approach to efficient and forward stable DST algorithms, *Linear Algebra and Its Applications* **542**: 360–390.

Polyakova, M. and Cariow, A. (2024). Fast type-II Hartley transform algorithms for short-length input sequences, *Applied Sciences* **14**(22): 10719.

Polyakova, M., Witenberg, A. and Cariow, A. (2024). The design of fast type-V discrete cosine transform algorithms for short-length input sequences, *Electronics* **13**(21): 2079–9292.

Potipantong, P., Sirisuk, P., Oraintara, S. and Worapishet, A. (2009). FPGA implementation of highly modular fast universal discrete transforms, *IEICE Transactions on Electronics* **92**(4): 576–586.

Regalia, P.A. and Sanjit, M.K. (1989). Kronecker products, unitary matrices and signal processing applications, *SIAM Review* **31**(4): 586–613.

Sadinov, S. (2017). Simulation study of M-ary QAM modulation techniques using Matlab/Simulink, *2017 40th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), Opatija, Croatia*, pp. 547–554.

Saleh, H.I., Ashour, M. and Salama, A.E. (2003). GDFT types mapping algorithms and structured regular FPGA implementation, *Proceedings of the 2003 International Symposium on Circuits and Systems, ISCAS'03, Bangkok, Thailand,* Vol. 4, pp. IV–IV.

Silva-García, V.M., Flores-Carapia, R., Rentería-Márquez, C., Luna-Benoso, B. and Chimal-Eguía, J.C. (2020). Image cipher applications using the elliptical curve and chaos, *International Journal of Applied Mathematics and Computer Science* **30**(2): 377–391, DOI: 10.34768/amcs-2020-0029.

Spałek, D. (2018). Two relations for generalized discrete Fourier transform coefficients, *Bulletin of the Polish Academy of Sciences: Technical Sciences* **66**(3): 275–281.

Syrjala, V., Valkama, M., Tchamov, N.N. and Rinne, J. (2009). Phase noise modelling and mitigation techniques in OFDM communications systems, *2009 Wireless Telecommunications Symposium, Prague, Czech Republic*, pp. 1–7.

Tsitsas, N.L. (2010). On block matrices associated with discrete trigonometric transforms and their use in the theory of wave propagation, *Journal of Computational Mathematics* **28**(6): 864–878.

Wilbur, M.R., Davidson, T.N. and Reilly, J.P. (2004). Efficient design of oversampled NPR GDFT filterbanks, *IEEE Transactions on Signal Processing* **52**(7): 1947–1963.

Yaroslavsky, L.P. (2014). Fast transforms in image processing: compression, restoration, and resampling, *Advances in Electrical Engineering* **2014**(1): 276241.

Zhang, X., Yu, F.X., Guo, R., Kumar, S., Wang, S. and Chang, S.-F. (2015). Fast orthogonal projection based on Kronecker product, *Proceedings of the IEEE International Conference on Computer Vision, Santiago, Chile*, pp. 2929–2937.

Zheng, B. (1996). A unified algorithm for sinusoid-class orthogonal transforms, *Proceedings of the 3rd International Conference on Signal Processing (ICSP'96), Beijing, China,* Vol. 1, pp. 36–39.

Janusz P. Papliński received his master's degree in technical computer science and telecommunications in 1990 and earned his doctoral degree in 1996. In 2016, he joined the Department of Computer Science and Information Technology at the West Pomeranian University of Technology in Szczecin, where he now works in the Department of Computer Architectures and Telecommunications. His research interests include optimising digital signal processing and image processing algorithms, and implementing algorithms into programmable circuits.

Marina Polyakova received her MSc, PhD and DSc degrees in computer science in 1994, 2004 and 2014, respectively. In 2015, she joined the Institute of Computer Systems of Odesa Polytechnic National University, where she is currently a professor at the Department of Applied Mathematics and Information Technologies. Her research interests include digital signal processing algorithms, image processing, neural networks, and time series analysis.

Aleksandr Cariow received his MSc, PhD and DSc degrees in computer science in 1976, 1984 and 2001, respectively. In 1999, he joined the Faculty of Computer Science and Information Technology, West Pomeranian University of Technology in Szczecin, Poland, where he currently works in the Department of Computer Architectures and Telecommunications. His research interests include digital signal and image processing algorithms, VLSI architectures, and data processing parallelization.

Received: 9 May 2025
Revised: 28 August 2025
Re-revised: 23 September 2025
Accepted: 22 October 2025