electronics

Article

Algorithmic Structures for Realizing Short-Length Circular Convolutions with Reduced Complexity

Aleksandr Cariow † and **Janusz P. Paplinski** *,†

Faculty of Computer Science and Information Technology, West Pomeranian University of Technology, Żołnierska 49, 71-210 Szczecin, Poland; acariow@wi.zut.edu.pl
* Correspondence: janusz.paplinski@zut.edu.pl
† These authors contributed equally to this work.

Abstract: A set of efficient algorithmic solutions suitable to the fully parallel hardware implementation of the short-length circular convolution cores is proposed. The advantage of the presented algorithms is that they require significantly fewer multiplications as compared to the naive method of implementing this operation. During the synthesis of the presented algorithms, the matrix notation of the cyclic convolution operation was used, which made it possible to represent this operation using the matrix–vector product. The fact that the matrix multiplicand is a circulant matrix allows its successful factorization, which leads to a decrease in the number of multiplications when calculating such a product. The proposed algorithms are oriented towards a completely parallel hardware implementation, but in comparison with a naive approach to a completely parallel hardware implementation, they require a significantly smaller number of hardwired multipliers. Since the wired multiplier occupies a much larger area on the VLSI and consumes more power than the wired adder, the proposed solutions are resource efficient and energy efficient in terms of their hardware implementation. We considered circular convolutions for sequences of lengths $N = 2, 3, 4, 5, 6, 7, 8,$ and 9.

Keywords: digital signal processing; circular convolution; resource-efficient algorithms

Citation: Cariow, A.; Paplinski, J.P. Algorithmic Structures for Realizing Short-Length Circular Convolutions with Reduced Complexity. *Electronics* **2021**, *10*, 2800. https://doi.org/10.3390/electronics10222800

Academic Editor: Sai-Weng Sin

Received: 10 September 2021
Accepted: 9 November 2021
Published: 15 November 2021

1. Introduction

Digital convolution is used in various applications of digital signal and image processing. Its most interesting areas of application are wireless communication and artificial neural networks [1–5]. The general principles of developing convolution algorithms were described in [6–12]. Various algorithmic solutions have been proposed to speed up the computation of circular convolution [7–11,13–16]. The most common approach to efficiently computing the circular algorithm is the Fast Fourier Transform (FFT) algorithm, as well as a number of other discrete orthogonal transformations [17–20]. There are also known methods for implementing discrete orthogonal transformations using circular convolution [20–22]. FFT-based convolution relies on the fact that convolution can be performed as simple multiplication in the frequency domain [23]. The FFT-based approach to computing circular convolution is traditionally used for long-length sequences. However, in many practical applications, a situation arises where both convolving sequences are relatively short. As examples, we can refer to algorithms for calculating short linear convolutions, as well as overlap-save and overlap-add methods [24–26]. It is known that these methods use splitting a long data sequence into small segments, calculating short cyclic convolutions of these segments and the impulse response coefficients of a Finite Impulse Response (FIR) filter, and then, combining the short convolutions into a single whole.

To date, many algorithmic solutions have been developed that involve the computation of cyclic convolution in the time domain [7–10,21,27–29]. In the cited publications, methods for calculating short convolutions were presented either as a set of arithmetic relations or as a set of matrix–vector products. Such approaches to the description of computations do not at all give an idea of the organization of the structures of processor cores intended for the implementation of the convolution operation. The solutions presented in the literature do not give a complete picture of the structural organization of such cores, if only because they (except for the cases $N = 2$ and $N = 3$) do not show the corresponding signal flow graphs. The absence of signal flow graphs in known publications also does not allow us to assess the possibilities of the obtained solutions from the point of view of their parallel implementation.

Therefore, in this paper, we propose a set of algorithmic solutions for circular convolution of small length N sequences from 2–9.

2. Preliminary Remarks

Let $\{h_n\}$ and $\{x_n\}$ be two N-point sequences. Their circular convolution is the sequence $\{y_n\}$, defined by [6]:

$$y_n = \sum_{k=0}^{N-1} h_k x_{((n-k) mod N\ )}, \quad 0 \le n \le N-1 \tag{1}$$

Usually, the elements of one of the convolved sequences are constants. For correctness, we assume that it will be the elements of sequence $\{h_n\}$.

Because sequences $\{x_n\}$ and $\{h_n\}$ are finite in length, then their circular convolution (1) can also be represented as a matrix–vector product:

$$\boldsymbol{Y}_{N\times1} = \boldsymbol{H}_N \boldsymbol{X}_{N\times1} \tag{2}$$

where:

$$\boldsymbol{H}_N = \begin{bmatrix} h_0 & h_{N-1} & \cdots & h_1 \\ h_1 & h_0 & \cdots & h_2 \\ \vdots & \vdots & \ddots & \vdots \\ h_{N-1} & h_{N-2} & \cdots & h_0 \end{bmatrix} \tag{3}$$

$$\boldsymbol{X}_{N\times1} = [x_0, x_1, \ldots, x_{N-1}]^T, \quad \boldsymbol{Y}_{N\times1} = [y_0, y_1, \ldots, y_{N-1}]^T, \quad \boldsymbol{H}_{N\times1} = [h_0, h_1, \ldots, h_{N-1}]^T.$$

In the following, we assume that $\boldsymbol{X}_{N\times1}$ will be the input data vector, $\boldsymbol{Y}_{N\times1}$ will be the output data vector, and $\boldsymbol{H}_{N\times1}$ will be the vector containing constants.

Calculating (2) directly requires N^2 multiplications and $(N-1)N$ additions. This leads to the fact that for a completely parallel hardware implementation of the circular convolution, N^2 multipliers and N N-input adders are required. Since the multiplier is a very cumbersome device and, when implemented in hardware, requires much more hardware resources compared to the adder, minimizing the number of multipliers required for the fully parallel implementation of algorithms is an important task.

Thus, taking into account the above, the purpose of this article is to develop and describe fully parallel resource-efficient algorithms for N = 2, 3, 4, 5, 6, 7, 8, and 9.

3. Algorithms for Short-Length Circular Convolution

3.1. Circular Convolution for $N = 2$

Let $\boldsymbol{X}_{2\times1} = [x_0, x_1]^T$ and $\boldsymbol{H}_{2\times1} = [h_0, h_1]^T$ be two-dimensional data vectors being convolved and $\boldsymbol{Y}_{2\times1} = [y_0, y_1]^T$ be an output vector representing a circular convolution. The task is reduced to calculating the following product:

$$\boldsymbol{Y}_{2\times1} = \boldsymbol{H}_2 \boldsymbol{X}_{2\times1} \tag{4}$$

where:

$$\boldsymbol{H}_2 = \begin{bmatrix} h_0 & h_1 \\ h_1 & h_0 \end{bmatrix},$$

Calculating (4) directly requires four multiplications and two additions. It is easy to see that the $\boldsymbol{H}_2$ matrix has an unusual structure. Taking into account this specificity leads to the fact that the number of multiplications in the calculation of the two-point circular convolution can be reduced [7].

The optimized computational procedure for computing the two-point circular convolution is as follows:

$$\boldsymbol{Y}_{2\times1} = \widetilde{\boldsymbol{H}}_2 \boldsymbol{D}_2^{(2)} \widetilde{\boldsymbol{H}}_2 \boldsymbol{X}_{2\times1} \tag{5}$$

where:

$$\widetilde{\boldsymbol{H}}_2 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}, \quad \boldsymbol{D}_2^{(2)} = diag(s_0^{(2)}, s_1^{(2)}), \quad s_0^{(2)} = \frac{1}{2}(h_0 + h_1), \quad s_1^{(2)} = \frac{1}{2}(h_0 - h_1)$$

Figure 1 shows a signal flow graph for the proposed algorithm, which also provides a simplified algorithmic structure of a fully parallel processing core for resource-effective implementation of the two-point circular convolution. All signal flow graphs are oriented from left to right. Straight lines denote the data circuits. The circles in these figures show the hardwired multipliers by a constant inscribed inside a circle. Points, where lines converge, denote adders, and dotted lines indicate the sign-change data circuits (datapaths with multiplication by -1).

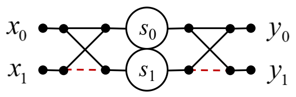

Figure 1. Algorithmic structure of the processing core for the computation of the 2-point circular convolution.

Therefore, it only takes two multiplications and four additions to compute the two-point circular convolution. As for the arithmetic blocks, for a completely parallel hardware implementation of the processor core to compute the two-point convolution, you need two multipliers and four two-input adders, instead of four multipliers and two two-input adders in the case of a completely parallel implementation (4).

3.2. *Circular Convolution for* $N = 3$

Let $\boldsymbol{X}_{3\times1} = [x_0, x_1, x_2]^T$ and $\boldsymbol{H}_{3\times1} = [h_0, h_1, h_2]^T$ be three-dimensional data vectors being convolved and $\boldsymbol{Y}_{3\times1} = [y_0, y_1, y_2]^T$ be an output vector representing circular convolution for $N = 3$. The task is reduced to calculating the following product:

$$\boldsymbol{Y}_{3\times1} = \boldsymbol{H}_3 \boldsymbol{X}_{3\times1} \tag{6}$$

where:

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

Calculating (6) directly requires nine multiplications and five additions. It is easy to see that the $\boldsymbol{H}_3$ matrix has an unusual structure. Taking into account this specificity leads to the fact that the number of multiplications in the calculation of the three-point circular convolution can be reduced [7,8,11,27].

Therefore, the optimized computational procedure for computing the three-point circular convolution is as follows:

$$\boldsymbol{Y}_3^{(3)} = \widetilde{\boldsymbol{A}}_3^{(3)} \boldsymbol{A}_{3\times4}^{(3)} \boldsymbol{D}_4^{(3)} \boldsymbol{A}_{4\times3}^{(3)} \boldsymbol{A}_3^{(3)} \boldsymbol{X}_{3\times1} \tag{7}$$

where:

$$A_3^{(3)} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & -1 & 0 \\ 0 & 1 & -1 \end{bmatrix}, \quad A_{4\times3}^{(3)} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix},$$

$$A_{3\times4}^{(3)} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix}, \quad \widetilde{A}_3^{(3)} = \begin{bmatrix} 1 & 1 & 0 \\ 1 & -1 & -1 \\ 1 & 0 & 1 \end{bmatrix},$$

$$D_4^{(3)} = \mathrm{diag}\left(s_0^{(3)}, s_1^{(3)}, s_2^{(3)}, s_3^{(3)}\right),$$

$$s_0^{(3)} = \frac{1}{3}(h_0 + h_1, +h_2), \quad s_1^{(3)} = (h_0 - h_1), \quad s_2^{(3)} = (h_1 - h_2), \quad s_3^{(3)} = \frac{1}{3}(h_0 + h_1 - 2h_2).$$

Figure 2 shows a signal flow graph of the proposed algorithm for the implementation of the three-point circular convolution.

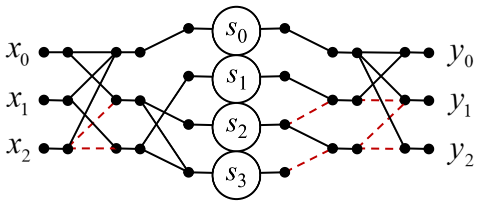

Figure 2. Algorithmic structure of the processing core for the computation of the 3-point circular convolution.

As for the arithmetic blocks, for a completely parallel hardware implementation of the processor core to compute the three-point convolution (7), you need four multipliers and eleven two-input adders, instead of nine multipliers and six two-input adders in the case of a completely parallel implementation (6). Therefore, we have exchanged five multipliers for five two-input adders.

3.3. *Circular Convolution for $N = 4$*

Let $X_{4\times1} = [x_0, x_1, x_2, x_3]^T$ and $H_{4\times1} = [h_0, h_1, h_2, h_3]^T$ be four-dimensional data vectors being convolved and $Y_{4\times1} = [y_0, y_1, y_2, y_3]^T$ be an output vector representing circular convolution for $N = 4$.

The task is reduced to calculating the following product:

$$Y_{4\times1} = H_4 X_{4\times1} \tag{8}$$

where:

$$H_4 = \begin{bmatrix} h_0 & h_3 & h_2 & h_1 \\ h_1 & h_0 & h_3 & h_2 \\ h_2 & h_1 & h_0 & h_3 \\ h_3 & h_2 & h_1 & h_0 \end{bmatrix}.$$

Calculating (8) directly requires 16 multiplications and 12 additions. It is easy to see that the H_4 matrix has an unusual structure. Taking into account this specificity leads to the fact that the number of multiplications in the calculation of the four-point circular convolution can be reduced.

Therefore, the optimized computational procedure for computing the four-point circular convolution is as follows:

$$Y_{4\times1} = A_4^{(4)} A_{4\times5}^{(4)} D_5^{(4)} A_{5\times4}^{(4)} A_4^{(4)} X_{4\times1} \tag{9}$$

where:

$$\mathbf{A}_4^{(4)} = \widetilde{\mathbf{H}}_2 \otimes \mathbf{I}_2 = \left[\begin{array}{cc|cc} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ \hline 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \end{array}\right], \quad \mathbf{A}_{5\times4}^{(4)} = \widetilde{\mathbf{H}}_2 \oplus \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix},$$

$$\mathbf{A}_{4\times5}^{(4)} = \widetilde{\mathbf{H}}_2 \oplus \begin{bmatrix} 0 & -1 & 1 \\ -1 & 0 & 1 \end{bmatrix},$$

$$\mathbf{D}_5^{(4)} = diag(s_0^{(4)} s_1^{(4)}, \ldots, s_4^{(4)}),$$

$$s_0^{(4)} = (h_0 + h_1 + h_2 + h_3)/4, \quad s_1^{(4)} = (h_0 - h_1 + h_2 - h_3)/4,$$

$$s_2^{(4)} = (h_0 - h_1 - h_2 + h_3)/2, \quad s_3^{(4)} = (h_0 + h_1 - h_2 - h_3)/2, \quad s_4^{(4)} = (h_0 - h_2)/2,$$

where $\mathbf{I}_N$ is an identity N matrix, $\widetilde{\mathbf{H}}_2$ is the 2×2 Hadamard matrix, and signs "$\otimes$" and "$\oplus$" denote the Kronecker product and direct sum of two matrices, respectively [30,31].

Figure 3 shows a signal flow graph of the proposed algorithm for the implementation of the four-point circular convolution.

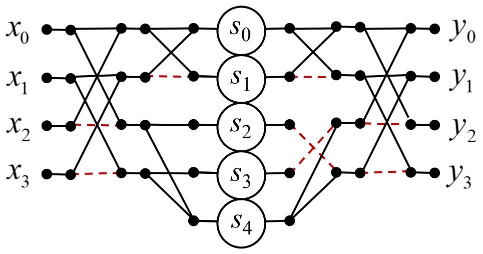

Figure 3. Algorithmic structure of the processing core for the computation of the 4-point circular convolution.

As for the arithmetic blocks, to compute the four-point convolution (9), you need five multipliers and fifteen two-input adders, instead of sixteen multipliers and twelve two-input adders in the case of a completely parallel implementation (8). The proposed algorithm saves eleven multiplications at the cost of three extra additions compared to the ordinary matrix–vector multiplication method.

3.4. Circular Convolution for $N = 5$

Let $\mathbf{X}_{5\times1} = [x_0, x_1, x_2, x_3, x_4]^T$ and $\mathbf{H}_{5\times1} = [h_0, h_1, h_2, h_3, h_4]^T$ be five-dimensional data vectors being convolved and $\mathbf{Y}_{9\times1} = [y_0, y_1, y_2, y_3, y_4]^T$ be an output vector representing a circular convolution for $N = 5$.

The task is reduced to calculating the following product:

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

where:

$$\mathbf{H}_5 = \begin{bmatrix} h_0 & h_4 & h_3 & h_2 & h_1 \\ h_1 & h_0 & h_4 & h_3 & h_2 \\ h_2 & h_1 & h_0 & h_4 & h_3 \\ h_3 & h_2 & h_1 & h_0 & h_4 \\ h_4 & h_3 & h_2 & h_1 & h_0 \end{bmatrix},$$

Calculating (10) directly requires 25 multiplications and 20 additions. It is easy to see that the $\mathbf{H}_5$ matrix has an unusual structure. Taking into account this specificity leads to the fact that the number of multiplications in the calculation of the five-point circular convolution can be reduced.

Therefore, an efficient algorithm for computing the five-point circular convolution can be represented using the following matrix–vector procedure:

$$\boldsymbol{Y}_{5\times1} = \boldsymbol{A}^{(5)}_{5\times7}\boldsymbol{A}^{(5)}_{7\times10}\boldsymbol{D}^{(5)}_{10}\boldsymbol{A}^{(5)}_{10\times9}\boldsymbol{A}^{(5)}_{9\times5}\boldsymbol{A}^{(5)}_{5}\boldsymbol{X}_{5\times1} \tag{11}$$

where:

$$\boldsymbol{A}^{(5)}_{5} = \left[\begin{array}{cccc|c} 1 & 0 & 0 & 0 & -1 \\ 0 & 1 & 0 & 0 & -1 \\ 0 & 0 & 1 & 0 & -1 \\ 0 & 0 & 0 & 1 & -1 \\ \hline 1 & 1 & 1 & 1 & 1 \end{array}\right], \quad \boldsymbol{A}^{(5)}_{9\times5} = \left[\begin{array}{cc|cc|c} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ \hline 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ \hline 1 & 0 & -1 & 0 & 0 \\ 0 & 1 & 0 & -1 & 0 \\ \hline 0 & 0 & 0 & 0 & 1 \end{array}\right],$$

$$\boldsymbol{D}^{(5)}_{16} = diag(s^{(5)}_0 s^{(5)}_1, \ \ldots \ , s^{(5)}_9),$$

$$s^{(5)}_0 = (h_0 - h_2 + h_3 - h_4)/4, \quad s^{(5)}_1 = (h_1 - h_2 + h_3 - h_4)/4,$$

$$s^{(5)}_2 = (3h_2 - 2h_1 + 2h_0 - 2h_3 + 3h_4)/5, \quad s^{(5)}_3 = (-h_0 + h_1 - h_2 + h_3),$$

$$s^{(5)}_4 = (-h_0 + h_1 - h_2 + h_3), \quad s^{(5)}_5 = (3h_0 - 2h_1 + 3h_2 - 2h_3 - 2h_4)/5,$$

$$s^{(5)}_6 = -h_2 + h_3, \quad s^{(5)}_7 = h_1 - h_2, \quad s^{(5)}_8 = (-h_0 - h_1 + 4h_2 - h_3 - h_4)/5,$$

$$s^{(5)}_9 = (h_0 + h_1 + h_2 + h_3 + h_4)/5,$$

$$\boldsymbol{A}^{(5)}_{10\times9} = \left[\begin{array}{cccccccc|c} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ \hline 0 & 0 & 1 & 0 & 0 & -1 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{array}\right],$$

$$\boldsymbol{A}^{(5)}_{7\times10} = \left[\begin{array}{ccc|ccc|ccc|c} 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{array}\right],$$

$$\boldsymbol{A}^{(5)}_{5\times7} = \left[\begin{array}{c|ccc|ccc} 1 & 0 & 0 & 0 & -1 & 0 & 1 \\ \hline -1 & -1 & -1 & -1 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & -1 & 1 \end{array}\right].$$

Figure 4 shows a data flow graph of the proposed algorithm for the implementation of the five-point circular convolution.

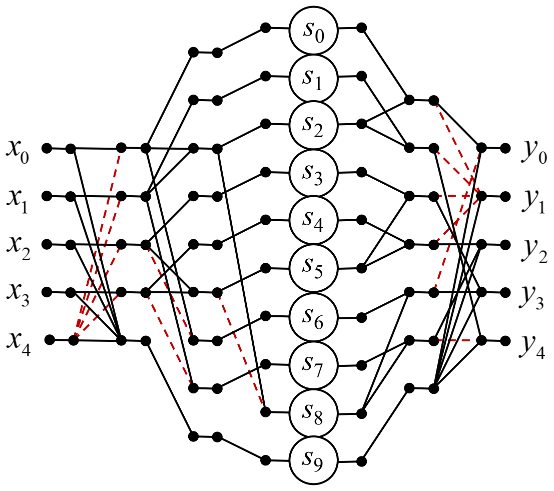

Figure 4. Algorithmic structure of the processing core for the computation of the 5-point circular convolution.

As for the arithmetic blocks, to compute the five-point convolution (11), you need ten multipliers, and thirty two-input adders, instead of twenty-five multipliers and twenty two-input adders in the case of a completely parallel implementation (10). The proposed algorithm saves 15 multiplications at the cost of 11 extra additions compared to the ordinary matrix–vector multiplication method.

3.5. Circular Convolution for $N = 6$

Let $\boldsymbol{X}_{6\times1} = [x_0, x_1, x_2, x_3, x_4, x_5]^T$ and $\boldsymbol{H}_{6\times1} = [h_0, h_1, h_2, h_3, h_4, h_5],^T$ be six-dimensional data vectors being convolved and $\boldsymbol{Y}_{11\times1} = [y_0, y_1, y_2, y_3, y_4, y_5]^T$ be an output vector representing a circular convolution for $N = 6$.

The task is reduced to calculating the following product:

$$\boldsymbol{Y}_{6\times1} = \boldsymbol{H}_6 \boldsymbol{X}_{6\times1} \tag{12}$$

where:

$$\boldsymbol{H}_6 = \begin{bmatrix} h_0 & h_5 & h_4 & h_3 & h_2 & h_1 \\ h_1 & h_0 & h_5 & h_4 & h_3 & h_2 \\ h_2 & h_1 & h_0 & h_5 & h_4 & h_3 \\ h_3 & h_2 & h_1 & h_0 & h_5 & h_4 \\ h_4 & h_3 & h_2 & h_1 & h_0 & h_5 \\ h_5 & h_4 & h_3 & h_2 & h_1 & h_0 \end{bmatrix},$$

Calculating (12) directly requires 36 multiplications and 30 additions. It is easy to see that the $\boldsymbol{H}_6$ matrix has an unusual structure. Taking into account this specificity leads to the fact that the number of multiplications in the calculation of the six-point circular convolution can be reduced.

Therefore, an efficient algorithm for computing the six-point circular convolution can be represented using the following matrix–vector procedure:

$$\boldsymbol{Y}_{6\times1} = \widetilde{\boldsymbol{P}}_6^{(6)} \boldsymbol{A}_6^{(6)} \boldsymbol{A}_6^{(6)} \boldsymbol{A}_{6\times8}^{(6)} \boldsymbol{D}_8^{(6)} \boldsymbol{A}_{8\times6}^{(6)} \widetilde{\boldsymbol{A}}_6^{(6)} \boldsymbol{A}_6^{(6)} \boldsymbol{P}_6^{(6)} \boldsymbol{X}_{6\times1} \tag{13}$$

where:

$$\boldsymbol{P}_6^{(6)} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}, \quad \boldsymbol{A}_6^{(6)} = \widetilde{\boldsymbol{H}}_2 \otimes \boldsymbol{I}_3 = \left[\begin{array}{ccc:ccc} 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ \hdashline 1 & 0 & 0 & -1 & 0 & 0 \\ 0 & 1 & 0 & 0 & -1 & 0 \\ 0 & 0 & 1 & 0 & 0 & -1 \end{array}\right],$$

$$\boldsymbol{D}_8 = diag(s_0^{(6)}, s_1^{(6)}, \ldots, s_7^{(6)}),$$

$$s_0^{(6)} = h_0 + h_3 + h_4 + h_1 + h_2 + h_5, \quad s_1^{(6)} = 3(h_4 + h_1 - h_0 - h_3),$$

$$s_2^{(6)} = 3(h_2 + h_5 - h_0 - h_3), \quad s_3^{(6)} = 3(h_0 + h_3) - (h_0 + h_3 + h_4 + h_1 + h_2 + h_5),$$

$$s_4^{(6)} = h_0 - h_3 + h_4 - h_1 + h_2 - h_5, \quad s_5^{(6)} = 3(h_4 - h_1 - h_0 + h_3),$$

$$s_6^{(6)} = 3(h_2 - h_5 - h_0 + h_3), \quad s_7^{(6)} = 3(h_0 + h_3) - (h_0 - h_3 + h_4 - h_1 + h_2 - h_5),$$

$$\widetilde{\boldsymbol{A}}_6^{(6)} = \left[\begin{array}{ccc:ccc} 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 & 0 & 0 \\ 1 & 0 & -1 & 0 & 0 & 0 \\ \hdashline 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 1 & 0 & -1 \end{array}\right], \oplus \boldsymbol{I}_5, \quad \widetilde{\boldsymbol{P}}_6^{(6)} = \left[\begin{array}{ccc:ccc} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ \hdashline 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{array}\right].$$

Figure 5 shows a data flow graph of the proposed algorithm for the implementation of the six-point circular convolution.

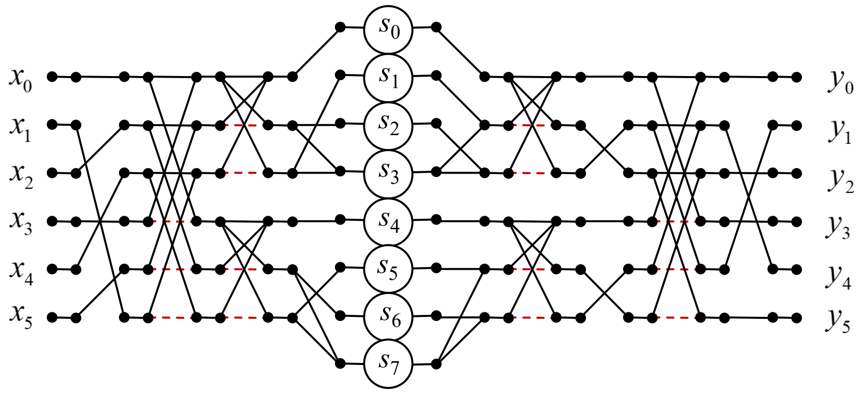

Figure 5. Algorithmic structure of the processing core for the computation of the 6-point circular convolution.

As far as arithmetic blocks are concerned, eight multipliers and thirty-four two-input adders are needed for the completely parallel hardware implementation of the processor core to compute the six-point convolution (13), instead of thirty-six multipliers and thirty two-input adders in the case of a completely parallel implementation (12). The proposed algorithm saves twenty-eight multiplications at the cost of six extra additions compared to the ordinary matrix–vector multiplication method.

3.6. Circular Convolution for $N = 7$

Let $\boldsymbol{X}_{7\times1} = [x_0, x_1, x_2, x_3, x_4, x_5, x_6]^T$ and $\boldsymbol{H}_{7\times1} = [h_0, h_1, h_2, h_3, h_4, h_5, h_6],^T$ be seven-dimensional data vectors being convolved and $\boldsymbol{Y}_{7\times1} = [y_0, y_1, y_2, y_3, y_4, y_5, y_6]^T$ be an output vector representing a circular convolution for $N = 7$.

The task is reduced to calculating the following product:

$$\boldsymbol{Y}_{7\times 1} = \boldsymbol{H}_7 \boldsymbol{X}_{7\times 1} \tag{14}$$

$$\boldsymbol{H}_7 = \begin{bmatrix} h_0 & h_6 & h_5 & h_4 & h_3 & h_2 & h_1 \\ h_1 & h_0 & h_6 & h_5 & h_4 & h_3 & h_2 \\ h_2 & h_1 & h_0 & h_6 & h_5 & h_4 & h_3 \\ h_3 & h_2 & h_1 & h_0 & h_6 & h_5 & h_4 \\ h_4 & h_3 & h_2 & h_1 & h_0 & h_6 & h_5 \\ h_5 & h_4 & h_3 & h_2 & h_1 & h_0 & h_6 \\ h_6 & h_5 & h_4 & h_3 & h_2 & h_1 & h_0 \end{bmatrix}.$$

Calculating (14) directly requires 49 multiplications and 42 additions. It is easy to see that the $\boldsymbol{H}_7$ matrix has an unusual structure. Taking into account this specificity leads to the fact that the number of multiplications in the calculation of the seven-point circular convolution can be reduced.

Therefore, an efficient algorithm for computing the seven-point circular convolution can be represented using the following matrix–vector procedure:

$$\boldsymbol{Y}_{7\times 1} = \boldsymbol{A}^{(7)}_{7\times 9}\boldsymbol{A}^{(7)}_{9\times 10}\boldsymbol{A}^{(7)}_{10\times 11}\boldsymbol{A}^{(7)}_{11\times 12}\boldsymbol{A}^{(7)}_{12\times 15}\boldsymbol{A}^{(7)}_{15\times 11}\boldsymbol{A}^{(7)}_{11}\boldsymbol{A}^{(7)}_{11\times 16}\boldsymbol{D}^{(7)}_{16}\boldsymbol{A}^{(7)}_{16}\boldsymbol{A}^{(7)}_{16\times 18}\boldsymbol{A}^{(7)}_{18\times 11}\boldsymbol{A}^{(7)}_{11\times 8}\boldsymbol{A}^{(7)}_{8\times 7}\boldsymbol{X}_{7\times 1} \tag{15}$$

where:

$$\boldsymbol{A}^{(7)}_{8\times 7} = \left[\begin{array}{cccccc:c} 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ \hdashline 1 & 0 & 0 & 0 & 0 & 0 & -1 \\ 0 & 1 & 0 & 0 & 0 & 0 & -1 \\ 0 & 0 & 1 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 1 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & 1 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 & 1 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{array}\right], \quad \boldsymbol{A}^{(7)}_{11\times 8} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix},$$

$$\boldsymbol{A}^{(7)}_{18\times 11} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix},$$

$$
\boldsymbol{A}_{16\times18}^{(7)} = \begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & -1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{bmatrix},
$$

$$
\boldsymbol{A}_{16}^{(7)} = \begin{bmatrix}
1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{bmatrix},
$$

$$\boldsymbol{D}_{16}^{(7)} = diag(s_0^{(7)}, s_1^{(7)}, \ldots, s_{15}^{(7)}),$$

$$s_0^{(7)} = (h_6 + h_5 + h_4 + h_3 + 2h_2 + h_1 + h_0)/7,$$

$$s_1^{(7)} = (-h_6 - 2h_5 + 3h_4 - h_3 - 2h_2 + h_1 + 2h_0)/2, \quad s_2^{(7)} = (2h_4 - h_3 - 2h_2 + h_1)/2,$$

$$s_3^{(7)} = (-h_6 + h_5 + 2h_4 - h_3 - 2h_2 + 3h_1 - h_0)/2,$$

$$s_4^{(7)} = (10h_6 + 3h_5 - 11h_4 + 10h_3 + 3h_2 - 11h_1 - 4h_0)/14,$$

$$s_5^{(7)} = (-2h_6 - 2h_5 - 2h_4 + 12h_3 + 5h_2 - 9h_1 - 2h_0)/14,$$

$$s_6^{(7)} = (2h_6 + 3h_5 - h_4 - 2h_3 + 3h_2 - h_1)/6,$$

$$s_7^{(7)} = (3h_6 - 11h_5 - 4h_4 + 10h_3 + 3h_2 - 11h_1 + 10h_0)/14,$$

$$s_8^{(7)} = (-2h_3 + 3h_2 - h_1)/6, \quad s_9^{(7)} = (3h_6 - h_5 - 2h_3 + 3h_2 - h_1 - 2h_0)/6,$$

$$s_{10}^{(7)} = (-h_6 + h_4 - h_3 + h_1)/6, \quad s_{11}^{(7)} = (-h_3 + h_1)/6, \quad s_{12}^{(7)} = (h_5 - h_3 + h_1 - h_0)/6,$$

$$s_{13}^{(7)} = 2h_6 - h_5 - 2h_4 + 3h_3 - 2h_2 - 2h_1 + h_0, \quad s_{14}^{(7)} = 2h_3 - h_2 - 2h_1 + h_0,$$

$$s_{15}^{(7)} = -h_6 - 2h_5 + h_4 + 2h_3 - h_2 - 2h_1 + 3h_0,$$

$$\boldsymbol{A}_{11\times 16}^{(7)} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 \end{bmatrix},$$

$$\boldsymbol{A}_{11}^{(7)} = \left[\begin{array}{ccccc:cccccc} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hdashline 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{array}\right],$$

$$\boldsymbol{A}_{15\times 11}^{(7)} = \left[\begin{array}{ccccc:cccccc} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hdashline 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{array}\right],$$

$$\boldsymbol{A}_{12\times 15}^{(7)} = \left[\begin{array}{cccccccc:ccccccc} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hdashline 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{array}\right],$$

$$\boldsymbol{A}_{11\times 12}^{(7)}=\left[\begin{array}{ccccccc:ccccc}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & -1 & -1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0\\
\hdashline
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0
\end{array}\right],$$

$$\boldsymbol{A}_{10\times 11}^{(7)}=\left[\begin{array}{ccccc:cccccc}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0\\
\hdashline
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{array}\right],$$

$$\boldsymbol{A}_{9\times 10}^{(7)}=\left[\begin{array}{ccccc:ccccc}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0\\
\hdashline
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & -1 & -1 & 0 & 0 & -1\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0
\end{array}\right],$$

$$\boldsymbol{A}_{7\times 9}^{(7)}=\begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0\\
1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0\\
1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0\\
1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0\\
1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0\\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{bmatrix}.$$

Figure 6 shows a data flow graph of the proposed algorithm for the implementation of the seven-point circular convolution.

As far as arithmetic blocks are concerned, sixteen multipliers and sixty-eight two-input adders are needed for the completely parallel hardware implementation of the processor core to compute the seven-point convolution (15), instead of forty-nine multipliers and forty-two two-input adders in the case of a completely parallel implementation (14). The proposed algorithm saves 33 multiplications at the cost of 26 extra additions compared to the ordinary matrix–vector multiplication method.

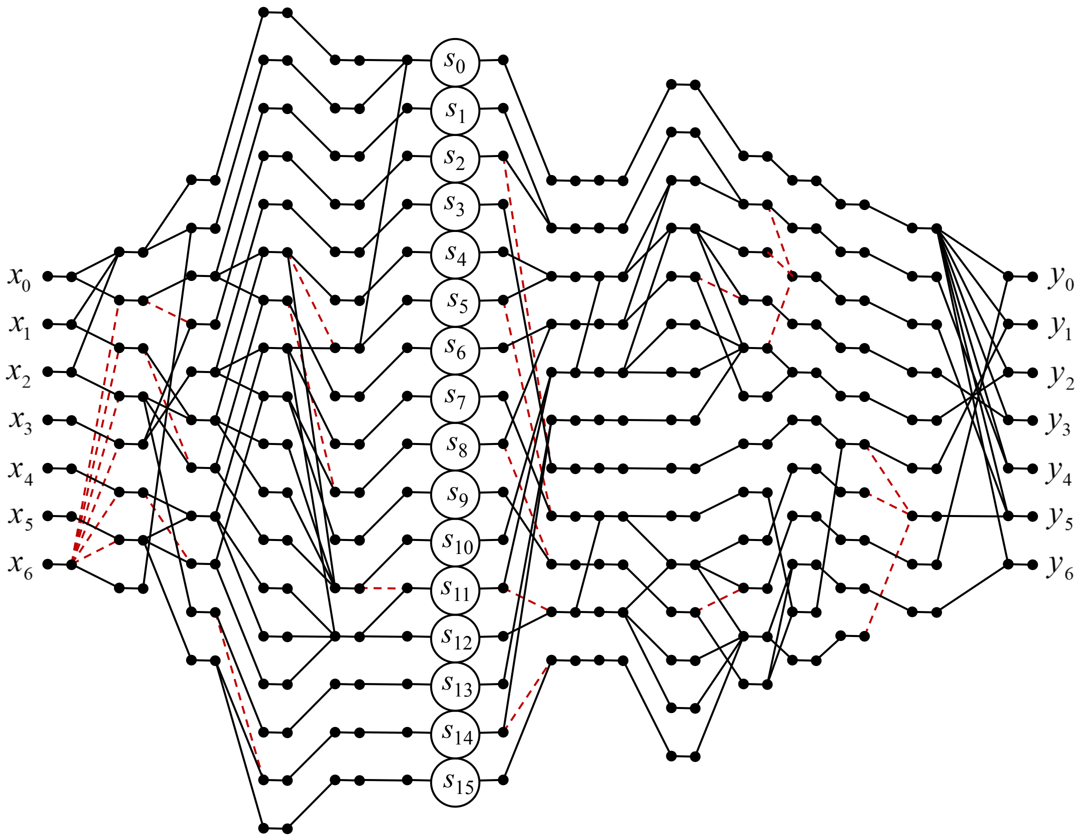

Figure 6. Algorithmic structure of the processing core for the computation of the 7-point circular convolution.

3.7. *Circular Convolution for* $N = 8$

Let $\mathbf{X}_{8\times1} = [x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7]^T$ and $\mathbf{H}_{8\times1} = [h_0, h_1, h_2, h_3, h_4, h_5, h_6, h_7],^T$ be eight-dimensional data vectors being convolved and $\mathbf{Y}_{8\times1} = [y_0, y_1, y_2, y_3, y_4, y_5, y_6, y_7]^T$ be an output vector representing a circular convolution for $N = 8$.

The task is reduced to calculating the following product:

$$\mathbf{Y}_{8\times1} = \mathbf{H}_8 \mathbf{X}_{8\times1} \tag{16}$$

$$\mathbf{H}_8 = \begin{bmatrix} h_0 & h_7 & h_6 & h_5 & h_4 & h_3 & h_2 & h_1 \\ h_1 & h_0 & h_7 & h_6 & h_5 & h_4 & h_3 & h_2 \\ h_2 & h_1 & h_0 & h_7 & h_6 & h_5 & h_4 & h_3 \\ h_3 & h_2 & h_1 & h_0 & h_7 & h_6 & h_5 & h_4 \\ h_4 & h_3 & h_2 & h_1 & h_0 & h_7 & h_6 & h_5 \\ h_5 & h_4 & h_3 & h_2 & h_1 & h_0 & h_7 & h_6 \\ h_6 & h_5 & h_4 & h_3 & h_2 & h_1 & h_0 & h_7 \\ h_7 & h_6 & h_5 & h_4 & h_3 & h_2 & h_1 & h_0 \end{bmatrix}.$$

Calculating (16) directly requires 64 multiplications and 56 additions. It is easy to see that the $\mathbf{H}_8$ matrix has an unusual structure. Taking into account this specificity leads to the fact that the number of multiplications in the calculation of the eight-point circular convolution can be reduced.

Therefore, an efficient algorithm for computing the eight-point circular convolution can be represented using the following matrix–vector procedure:

$$\mathbf{Y}_{8\times1} = \mathbf{P}_8^{(8)} \mathbf{A}_8^{(8)} \mathbf{A}_{8\times10}^{(8)} \mathbf{A}_{10\times14}^{(8)} \mathbf{D}_{14}^{(8)} \mathbf{A}_{14\times10}^{(8)} \mathbf{A}_{10\times8}^{(8)} \mathbf{A}_8^{(8)} \mathbf{X}_{8\times1} \tag{17}$$

where:

$$A_8^{(8)} = \widetilde{H}_2 \otimes I_4 = \left[\begin{array}{cccc|cccc} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ \hline 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & -1 \end{array}\right],$$

$$A_{10\times 8}^{(8)} = (\widetilde{H}_2 \otimes I_2) \oplus \left[\begin{array}{c|c} I_2 & 0_2 \\ \hline 0_2 & I_2 \\ \hline I_2 & I_2 \end{array}\right], \quad A_{14\times 10}^{(8)} = \widetilde{H}_2 \oplus \left(I_4 \otimes \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix} \right),$$

$$D_{14}^{(8)} = diag(s_0^{(8)}, s_1^{(8)}, \ldots, s_{13}^{(8)}),$$

$$s_0^{(8)} = \frac{1}{8}(h_0 + h_1 + h_2 + h_3 + h_4 + h_5 + h_6 + h_7),$$

$$s_1^{(8)} = \frac{1}{8}(h_0 - h_1 + h_2 - h_3 + h_4 - h_5 + h_6 - h_7),$$

$$s_2^{(8)} = \frac{1}{4}(-h_0 + h_1 + h_2 - h_3 - h_4 + h_5 + h_6 - h_7),$$

$$s_3^{(8)} = \frac{1}{4}(-h_0 - h_1 + h_2 + h_3 - h_4 - h_5 + h_6 + h_7), \quad s_4^{(8)} = \frac{1}{4}(h_0 - h_2 + h_4 - h_6),$$

$$s_5^{(8)} = \frac{1}{2}(h_0 - h_1 - h_2 + h_3 - h_4 + h_5 + h_6 - h_7),$$

$$s_6^{(8)} = \frac{1}{2}(h_0 + h_1 - h_2 + h_3 - h_4 - h_5 + h_6 - h_7), \quad s_7^{(8)} = \frac{1}{2}(-h_0 + h_2 + h_4 - h_6),$$

$$s_8^{(8)} = \frac{1}{2}(h_0 - h_1 + h_2 - h_3 - h_4 + h_5 - h_6 + h_7),$$

$$s_9^{(8)} = \frac{1}{2}(h_0 - h_1 + h_2 + h_3 - h_4 + h_5 - h_6 - h_7), \quad s_{10}^{(8)} = \frac{1}{2}(-h_0 - h_2 + h_4 + h_6),$$

$$s_{11}^{(8)} = \frac{1}{2}(-h_0 + h_1 + h_4 - h_5), \quad s_{12}^{(8)} = \frac{1}{2}(-h_0 - h_3 + h_4 + h_7), \quad s_{13}^{(8)} = \frac{1}{2}(h_0 - h_4),$$

$$A_{10\times 14}^{(8)} = \widetilde{H}_2 \oplus \left(I_4 \otimes \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix} \right), \quad A_{8\times 10}^{(8)} = (\widetilde{H}_2 \otimes I_2) \oplus \left[\begin{array}{c|c|c} 0_2 & I_2 & I_2 \\ \hline I_2 & 0_2 & I_2 \end{array}\right],$$

$$P_8^{(8)} = \left[\begin{array}{c|c} 0_{5\times 3} & I_5 \\ \hline I_3 & 0_{3\times 5} \end{array}\right] = \left[\begin{array}{ccc|ccccc} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ \hline 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{array}\right].$$

Figure 7 shows a data flow graph of the proposed algorithm for the implementation of the eight-point circular convolution.

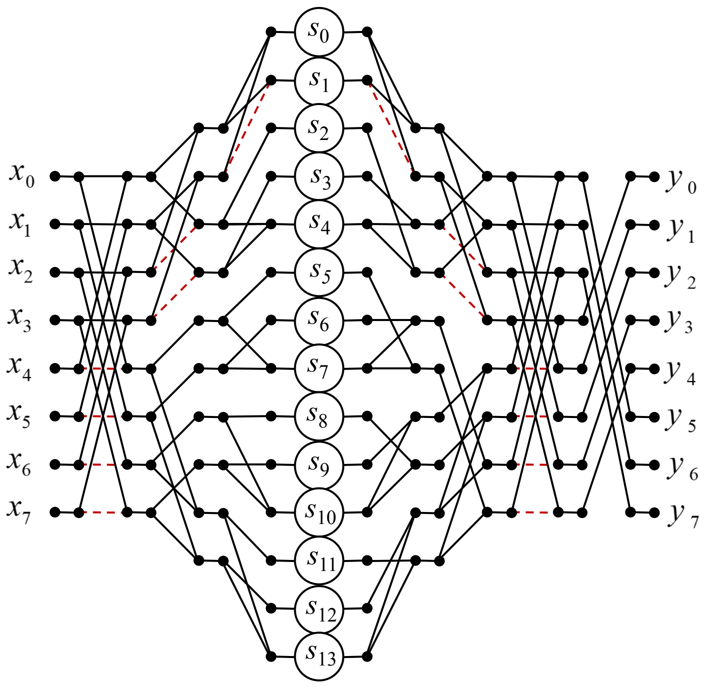

Figure 7. Algorithmic structure of the processing core for the computation of the 8-point circular convolution.

As far as arithmetic blocks are concerned, fourteen multipliers and forty-six two-input adders are needed for the completely parallel hardware implementation of the processor core to compute the eight-point convolution (17), instead of sixty-four multipliers and fifty-six two-input adders in the case of a completely parallel implementation (16). The proposed algorithm saves 50 multiplications and 10 additions compared to the ordinary matrix–vector multiplication method.

3.8. Circular Convolution for $N = 9$

Let $\boldsymbol{X}_{9\times1} = [x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8]^T$ and $\boldsymbol{H}_{9\times1} = [h_0, h_1, h_2, h_3, h_4, h_5, h_6, h_7, h_8]^T$ be nine-dimensional data vectors being convolved and $\boldsymbol{Y}_{9\times1} = [y_0, y_1, y_2, y_3, y_4, y_5, y_6, y_7, y_8]^T$ be an output vector representing a circular convolution for $N = 9$.

The task is reduced to calculating the following product:

$$\boldsymbol{Y}_{9\times1} = \boldsymbol{H}_9\boldsymbol{X}_{9\times1} \tag{18}$$

$$\boldsymbol{H}_9 = \begin{bmatrix} h_0 & h_8 & h_7 & h_6 & h_5 & h_4 & h_3 & h_2 & h_1 \\ h_1 & h_0 & h_8 & h_7 & h_6 & h_5 & h_4 & h_3 & h_2 \\ h_2 & h_1 & h_0 & h_8 & h_7 & h_6 & h_5 & h_4 & h_3 \\ h_3 & h_2 & h_1 & h_0 & h_8 & h_7 & h_6 & h_5 & h_4 \\ h_4 & h_3 & h_2 & h_1 & h_0 & h_8 & h_7 & h_6 & h_5 \\ h_5 & h_4 & h_3 & h_2 & h_1 & h_0 & h_8 & h_7 & h_6 \\ h_6 & h_5 & h_4 & h_3 & h_2 & h_1 & h_0 & h_8 & h_7 \\ h_7 & h_6 & h_5 & h_4 & h_3 & h_2 & h_1 & h_0 & h_8 \\ y_8 & h_7 & h_6 & h_5 & h_4 & h_3 & h_2 & h_1 & h_0 \end{bmatrix}.$$

Calculating (18) directly requires 81 multiplications and 72 additions. It is easy to see that the $\boldsymbol{H}_9$ matrix has an unusual structure. Taking into account this specificity leads to the fact that the number of multiplications in the calculation of the nine-point circular convolution can be reduced.

Therefore, an efficient algorithm for computing the nine-point circular convolution can be represented using the following matrix–vector procedure:

$$\boldsymbol{Y}_{9\times1}=\widetilde{\boldsymbol{A}}_9^{(9)}\boldsymbol{A}_{9\times13}^{(9)}\boldsymbol{A}_{13\times15}^{(9)}\boldsymbol{A}_{15}^{(9)}\boldsymbol{A}_{15\times19}^{(9)}\boldsymbol{D}_{19}^{(9)}\boldsymbol{A}_{19\times16}^{(9)}\boldsymbol{A}_{16\times14}^{(9)}\boldsymbol{A}_{14\times9}^{(9)}\boldsymbol{A}_9^{(9)}\boldsymbol{X}_{9\times1} \tag{19}$$

where:

$$\boldsymbol{A}_9^{(9)}=\left[\begin{array}{ccc|ccc|ccc}
1&0&0&0&0&0&-1&0&0\\
0&1&0&0&0&0&0&-1&0\\
0&0&1&0&0&0&0&0&-1\\
\hline
0&0&0&1&0&0&-1&0&0\\
0&0&0&0&1&0&0&-1&0\\
0&0&0&0&0&1&0&0&-1\\
\hline
1&0&0&1&0&0&1&0&0\\
0&1&0&0&1&0&0&1&0\\
0&0&1&0&0&1&0&0&1
\end{array}\right],$$

$$\boldsymbol{A}_{14\times9}^{(9)}=\begin{bmatrix}
1&0&0&0&0&0&0&0&0\\
1&0&0&-1&0&0&0&0&0\\
0&0&0&0&0&1&0&0&0\\
0&1&0&0&0&0&0&0&0\\
0&0&0&1&0&0&0&0&0\\
0&0&0&0&1&0&0&0&0\\
0&0&0&1&0&1&0&0&0\\
1&0&1&0&0&0&0&0&0\\
0&0&0&0&0&0&1&1&1\\
0&0&0&0&0&0&1&0&-1\\
0&0&0&0&0&1&0&0&0\\
0&0&1&0&0&-1&0&0&0\\
0&0&0&0&0&0&0&1&-1\\
0&0&1&0&0&0&0&0&0
\end{bmatrix},$$

$$\boldsymbol{A}_{16\times14}^{(9)}=\begin{bmatrix}
0&1&0&0&0&0&0&0&0&0&0&0&0&0\\
0&1&1&-1&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&1&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&1&1&0&0&0&0&0&0&0\\
0&0&0&1&0&0&0&1&0&0&0&0&0&0\\
0&0&0&0&0&-1&1&0&0&0&0&0&0&0\\
0&0&0&-1&0&0&1&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&1&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&1&0&0&0&0\\
0&0&0&0&0&0&0&0&0&1&0&0&1&0\\
1&0&0&0&0&0&0&0&0&0&0&0&0&0\\
1&0&0&0&0&-1&0&0&0&0&0&-1&0&0\\
0&0&0&0&0&0&0&0&0&0&1&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&1&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&1&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&1
\end{bmatrix},$$

$$
\boldsymbol{A}_{19\times16}^{(9)}=\left[\begin{array}{ccccccc|ccccccccc}
1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&-1&0&0&0&0&0&0&0&0&0&1&0&0&0&0\\
0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&-1&1&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&-1&1&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0\\
\hline
0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1
\end{array}\right]
$$

$$\boldsymbol{D}_{19}^{(9)}=diag(s_0^{(9)},s_1^{(9)},\ldots,s_{18}^{(9)}),$$

$$s_0^{(9)}=b_{17}=\frac{1}{3}(2b_0-b_2+b_4),\quad s_1^{(9)}=b_{23}=\frac{1}{9}(-b_0+b_2-b_3+b_5),$$

$$s_2^{(9)}=b_{24}=b_{23}-b_{22},\quad s_3^{(9)}=b_{16}=\frac{1}{3}(2b_0+b_1-b_2-2b_3+b_5),$$

$$s_4^{(9)}=b_{10}=\frac{1}{18}(b_0+3b_1+2b_2-2b_3-3b_4-b_5),\quad s_5^{(9)}=b_{12}=b_{10}+b_{11},$$

$$s_6^{(9)}=b_{11}=\frac{1}{18}(b_0-b_2+b_3+3b_4+2b_5),\quad s_7^{(9)}=b_{13}=\frac{1}{6}(-b_0+b_1-b_4+b_5),$$

$$s_8^{(9)}=b_{15}=b_{13}+b_{14},\quad s_9^{(9)}=b_{14}=\frac{1}{6}(b_0-b_2-b_3+b_4),\quad s_{10}^{(9)}=b_9=\frac{1}{9}(b_6+b_7+b_8),$$

$$s_{11}^{(9)}=b_{25}=\frac{1}{3}(b_6-b_8),\quad s_{12}^{(9)}=b_{27}=\frac{1}{3}(b_{25}+b_{26}),\quad s_{13}^{(9)}=b_{18}=b_{17}-b_{16},$$

$$s_{14}^{(9)}=b_{22}=\frac{1}{9}(b_0-b_2-2b_3+2b_5),\quad s_{15}^{(9)}=b_{19}=\frac{1}{3}(b_0-b_1-2b_2+b_4),$$

$$s_{16}^{(9)}=b_{20}=\frac{1}{3}(-b_1+b_3-2b_5),\quad s_{17}^{(9)}=b_{26}=\frac{1}{3}(b_7-b_8),\quad s_{18}^{(9)}=b_{21}=b_{20}-b_{19},$$

$$b_0=h_0-h_3+2h_6,\quad b_1=h_1-h_4+2h_7,\quad b_2=-h_2-h_5+2h_8,\quad b_3=h_0-2h_3+h_6,$$

$$b_4=h_1-2h_4+h_7,\quad b_5=h_2-2h_5+h_8,\quad b_6=h_0+h_3+h_6,\quad b_7=h_1+h_4+h_7,$$

$$b_8=h_2+h_5+h_8,$$

$$\boldsymbol{A}^{(9)}_{15\times 19} = \begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{bmatrix},$$

$$\boldsymbol{A}^{(9)}_{15} = \begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix},$$

$$\boldsymbol{A}^{(9)}_{13\times 15} = \begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{bmatrix},$$

$$\boldsymbol{A}^{(9)}_{9\times 13} = \left[\begin{array}{ccccccc:cccccc}
1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
\hdashline
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{array}\right],$$

$$\widetilde{\boldsymbol{A}}_9^{(9)} = \begin{bmatrix} -1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 1 \\ 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{bmatrix}.$$

Figure 8 shows a data flow graph of the proposed algorithm for the implementation of the nine-point circular convolution.

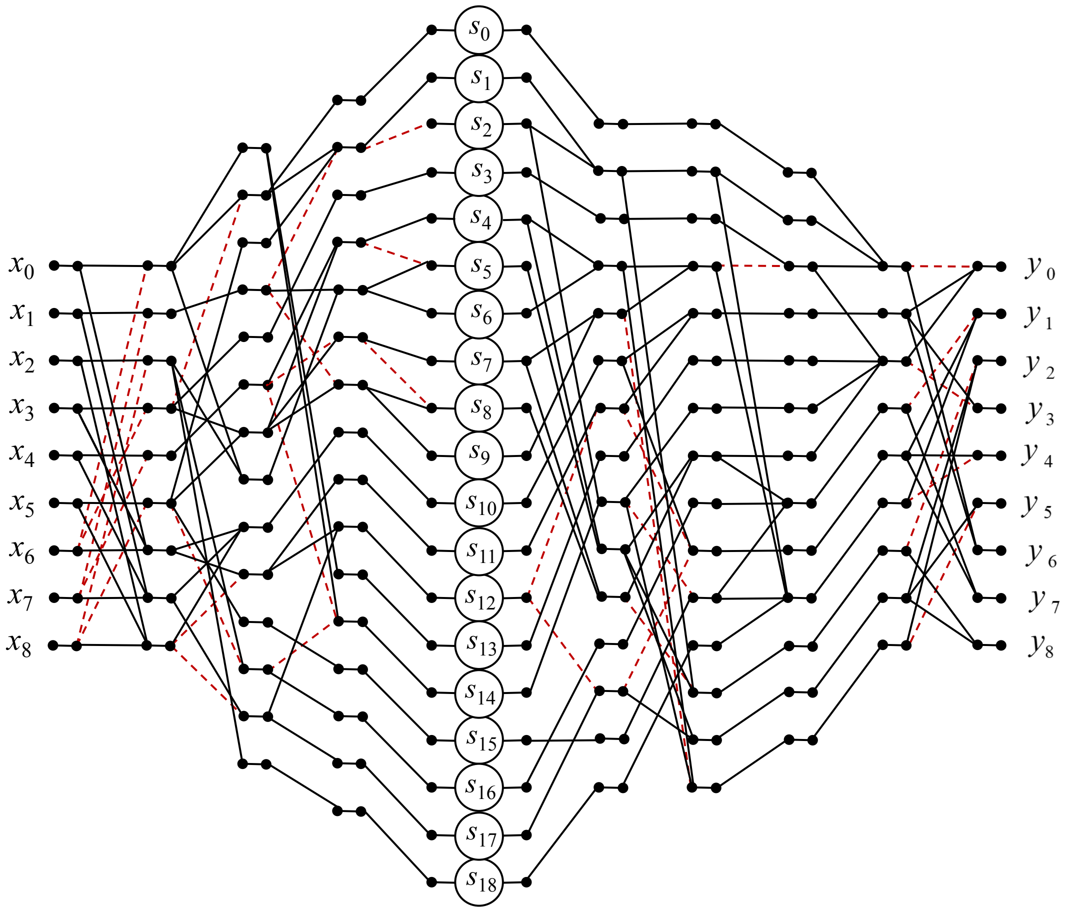

Figure 8. Algorithmic structure of the processing core for the computation of the 9-point circular convolution.

As far as arithmetic blocks are concerned, nineteen multipliers and seventy-four two-input adders are needed for the completely parallel hardware implementation of the processor core to compute the nine-point convolution (19), instead of eighty-one multipliers and seventy-two two-input adders in the case of a completely parallel implementation (18). The proposed algorithm saves sixty-two multiplications at the cost of the one extra addition compared to the ordinary matrix–vector multiplication method.

4. Implementation Complexity

We now estimate the hardware implementation costs of each solution. We assumed that the hardware implementation cost of the hardwired multiplier is α and the hardware implementation cost of the two-input adder is β. By the hardware implementation cost of the estimated solution, we mean a generalized assessment of the hardware complexity of implementing specific solutions, considering the area within the VLSI, the dissipation power, and therefore, the consumed energy. We also took into account that the N-input adder consists of $N-1$ two-input adders. In this way, we treated the implementation cost of an N-input adder as the sum of the implementation costs of $N-1$ two-input adders. Then, the total hardware implementation cost C of each solution is equal to:

$$C = \alpha M + \beta A$$

where M and A mean, respectively, the number of multipliers and the number of two-input adders required for the fully parallel implementation of a particular solution. We can normalize the above equation regarding the cost of the adder, obtaining the normalized cost:

$$\widetilde{C} = \gamma M + A$$

where $\gamma = \alpha/\beta$ is the relative cost coefficient of the multiplier.

Table 1 shows estimates of the number of arithmetic blocks for the fully parallel implementation of the short-length circular convolution algorithms. The last two columns of the table show the unified hardware costs for the implementation of the corresponding solutions, expressed in terms of the implementation cost of one two-input adder. The charts presented in Figure 9 illustrate the normalized hardware implementation costs $\widetilde{C}$ of the proposed solution and naive method for various values of γ and N.

Table 1. Comparative estimates of the number of hardwired multipliers and adders for the case of completely parallel implementations of the naive-method-based solutions and of the proposed solutions.

Length N	**Number of Arithmetical Blocks (Multipliers—"×" and Adders—"+")**				**Implementation Normalized Cost Estimate**	
	Naive Method		**Proposed Solutions**		$\widetilde{C}$	
0	"×"	"+"	"×"	"+"	**Naive Method**	**Proposed Solutions**
2	4	2	2	4	$4\gamma + 2$	$2\gamma + 4$
3	9	6	4	11	$9\gamma + 6$	$4\gamma + 11$
4	16	12	5	15	$16\gamma + 12$	$5\gamma + 15$
5	25	20	10	31	$25\gamma + 20$	$10\gamma + 31$
6	36	30	8	32	$36\gamma + 30$	$8\gamma + 32$
7	49	42	16	70	$49\gamma + 42$	$16\gamma + 70$
8	64	56	14	46	$64\gamma + 56$	$14\gamma + 46$
9	81	72	19	73	$81\gamma + 72$	$19\gamma + 74$

Cost comparisons can also be made using percentage changes:

$$\delta_c = \frac{\widetilde{C}_p - \widetilde{C}_n}{\widetilde{C}_n} 100\%$$

where $\widetilde{C}_p$ and $\widetilde{C}_n$ are the normalized cost of proposed algorithm and naive method, respectively.

Figure 10 shows the value of percentage changes as a function of the relative cost coefficient for various values of N. Assuming that the cost of the multiplier is always at least equal to, and most often greater than, the cost of the adder $\gamma \geq 1$, the cost of the proposed algorithm is never greater than that of the naive method, and in some cases, even cost savings of over 70% are obtained.

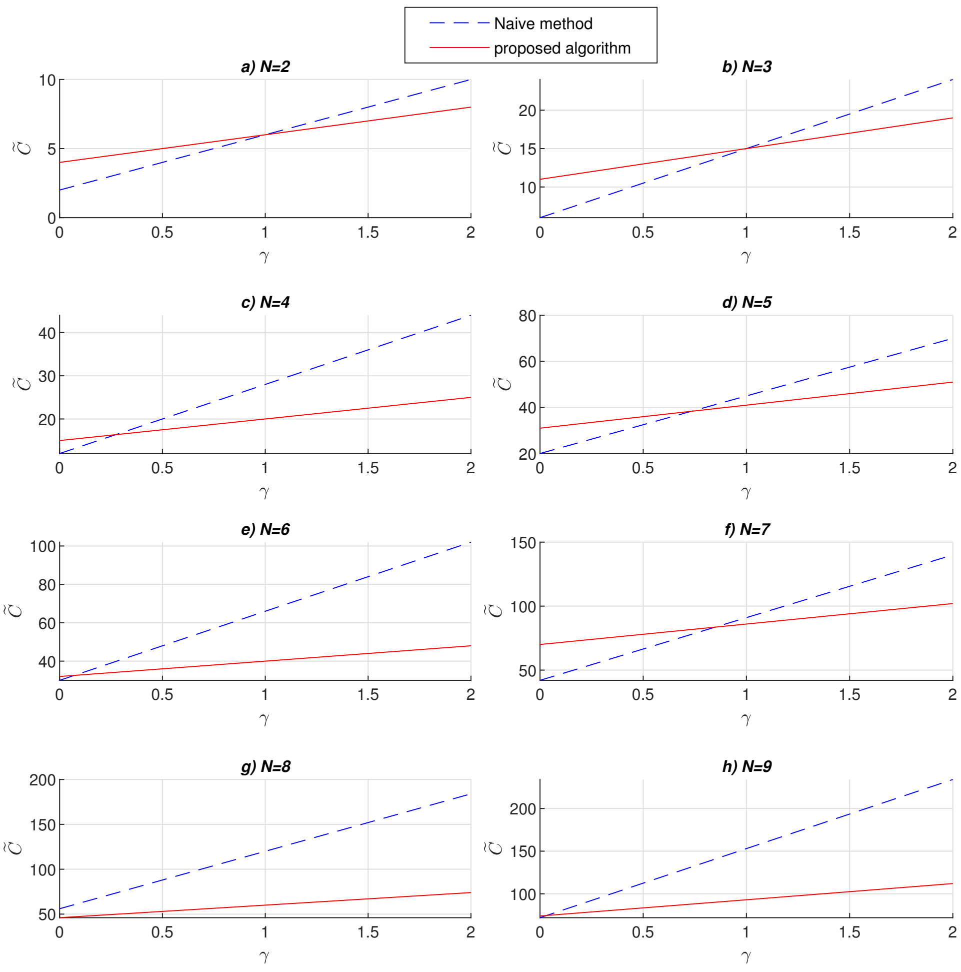

Figure 9. The normalized hardware implementation costs $\widetilde{C}$ of the proposed solution and naive method, as a function of the relative cost coefficient γ of the multiplier, for various N.

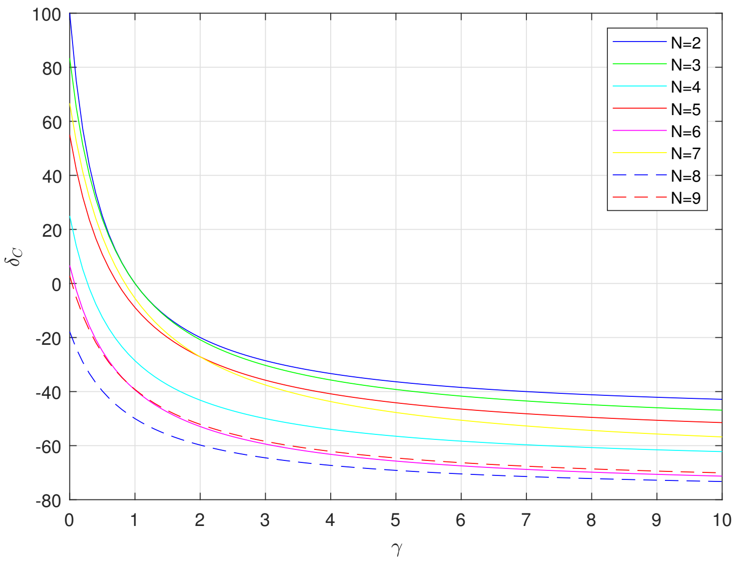

Figure 10. The value of percentage changes δ_c as a function of the relative cost coefficient γ for various values of N.

5. Conclusions

In this article, we analyzed the possibilities of reducing the multiplicative complexity of computing circular convolutions for input sequences of small lengths. We synthesized new hardware-efficient, fully parallel algorithms to implement these operations for $N = 3, 4, 5, 6, 7, 8$, and 9. The reduced multiplicative complexity of the proposed algorithms is especially important when developing specialized fully parallel VLSI processors, since it minimizes the number of necessary hardware multipliers and reduces the power dissipation, as well as the total cost of the implementation of the entire system being introduced [30–32]. Thus, a decrease in the number of multipliers, even at the expense of a moderate increase in the number of adders, plays an important role in the hardware implementation of the proposed algorithms. Consequently, the use of the proposed solutions makes it possible to reduce the complexity of the hardware implementation of the cyclic convolution kernels. In addition, as can be seen from Figures 1–8, the algorithms presented in the article have a pronounced regular and modular structure. This facilitates the mapping of these algorithms to the ASIC structure and unifies their implementation in FPGAs. Thus, the acceleration of computations in the implementation of these algorithms can also be achieved by parallelizing the computations.

Author Contributions: Conceptualization, A.C. and J.P.P.; methodology, A.C. and J.P.P.; software, J.P.P.; validation, A.C. and J.P.P.; formal analysis, A.C. and J.P.P.; investigation, A.C. and J.P.P.; resources, A.C.; data curation, J.P.P.; writing—original draft preparation, A.C.; writing—review and editing, J.P.P.; visualization, A.C. and J.P.P.; supervision, A.C.; project administration, J.P.P.; funding acquisition, A.C. and J.P.P. 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.

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

References

1. Loulou, A.; Yli-Kaakinen, J.; Renfors, M. Efficient fast-convolution based implementation of 5G waveform processing using circular convolution decomposition. In Proceedings of the 2017 IEEE International Conference on Communications (ICC), Paris, France, 21–25 May 2017; pp. 1–7.
2. Loulou, A.; Yli-Kaakinen, J.; Renfors, M. Advanced low-complexity multicarrier schemes using fast-convolution processing and circular convolution decomposition. *IEEE Trans. Signal Process.* **2019**, *67*, 2304–2319. [CrossRef]
3. Mathieu, M.; Henaff, M.; LeCun, Y. Fast training of convolutional networks through ffts. *arXiv* **2013**, arXiv:1312.5851.
4. Lin, S.; Liu, N.; Nazemi, M.; Li, H.; Ding, C.; Wang, Y.; Pedram, M. FFT-based deep learning deployment in embedded systems. In Proceedings of the 2018 Design, Automation & Test in Europe Conference & Exhibition (DATE), Dresden, Germany, 19–23 March 2018; pp. 1045–1050.
5. Abtahi, T.; Shea, C.; Kulkarni, A.; Mohsenin, T. Accelerating convolutional neural network with fft on embedded hardware. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **2018**, *26*, 1737–1749. [CrossRef]
6. Burrus, S.C.; Parks, T.W. *DFT/FFT and Convolution Algorithms*; John Wiley & Sons: Hoboken, NJ, USA, 1985.
7. Blahut, R.E. *Fast Algorithms for Signal Processing*; Cambridge University Press: Cambridge, UK, 2010.
8. McClellen, J.H.; Rader, C.M. *Number Theory in Digital Signal Processing*; Professional Technical Reference; Prentice Hall: Hoboken, NJ, USA, 1979.
9. Tolimieri, R.; An, M.; Lu, C. *Algorithms for Discrete Fourier Transform and Convolution*; Springer: Berlin/Heidelberg, Germany, 1989.
10. Berg, L.; Nussbaumer, H. Fast Fourier Transform and Convolution Algorithms. *Z. Angew. Math. Mech.* **1982**, *62*, 282. [CrossRef]
11. Garg, H.K. *Digital Signal Processing Algorithms: Number Theory, Convolution, Fast Fourier Transforms, and Applications*; Routledge: London, UK, 2017.
12. Bi, G.; Zeng, Y. *Transforms and Fast Algorithms for Signal Analysis and Representations*; Springer Science & Business Media: Berlin/Heidelberg, Germany, 2003.
13. Selesnick, I.W.; Burrus, C.S. Extending Winograd's small convolution algorithm to longer lengths. In Proceedings of the IEEE International Symposium on Circuits and Systems-ISCAS'94, London, UK, 30 May–2 June 1994; Volume 2, pp. 449–452.
14. Stasinski, R. Extending sizes of effective convolution algorithms. *Electron. Lett.* **1990**, *26*, 1602–1604. [CrossRef]

15. Karas, P.; Svoboda, D. Algorithms for efficient computation of convolution. In *Design and Architectures for Digital Signal Processing*; IntechOpen: London, UK, 2013; pp. 179–208.
16. Mohammad, K.; Agaian, S. Efficient FPGA implementation of convolution. In Proceedings of the 2009 IEEE International Conference on Systems, Man and Cybernetics, San Antonio, TX, USA, 11–14 October 2009; pp. 3478–3483.
17. Duhamel, P.; Vetterli, M. Cyclic convolution of real sequences: Hartley versus fourier and new schemes. In Proceedings of the ICASSP'86. IEEE International Conference on Acoustics, Speech, and Signal Processing, Tokyo, Japan, 7–11 April 1986; Volume 11, pp. 229–232.
18. Duhamel, P.; Vetterli, M. Improved Fourier and Hartley transform algorithms: Application to cyclic convolution of real data. *IEEE Trans. Acoust. Speech Signal Process.* **1987**, *35*, 818–824. [CrossRef]
19. Meher, P.; Panda, G. Fast Computation of Circular Convolution of Real Valued Data using Prime Factor Fast Hartley Transform Algorithm. *IEEE J. Res.* **1995**, *41*, 261–264. [CrossRef]
20. Reju, V.G.; Koh, S.N.; Soon, Y. Convolution using discrete sine and cosine transforms. *IEEE Signal Process. Lett.* **2007**, *14*, 445–448. [CrossRef]
21. Cheng, C.; Parhi, K.K. Hardware efficient fast DCT based on novel cyclic convolution structures. *IEEE Trans. Signal Process.* **2006**, *54*, 4419–4434. [CrossRef]
22. Chan, Y.H.; Siu, W.C. General approach for the realization of DCT/IDCT using convolutions. *Signal Process.* **1994**, *37*, 357–363. [CrossRef]
23. Hunt, B. A matrix theory proof of the discrete convolution theorem. *IEEE Trans. Audio Electroacoust.* **1971**, *19*, 285–288. [CrossRef]
24. Cariow, A.; Paplinski, J.P. Some algorithms for computing short-length linear convolution. *Electronics* **2020**, *9*, 2115. [CrossRef]
25. Adámek, K.; Dimoudi, S.; Giles, M.; Armour, W. GPU fast convolution via the overlap-and-save method in shared memory. *ACM Trans. Archit. Code Optim. (TACO)* **2020**, *17*, 1–20. [CrossRef]
26. Narasimha, M.J. Modified overlap-add and overlap-save convolution algorithms for real signals. *IEEE Signal Process. Lett.* **2006**, *13*, 669–671. [CrossRef]
27. Huang, T.S. Two-dimensional digital signal processing II. Transforms and median filters. In *Two-Dimensional Digital Signal Processing II. Transforms and Median Filters*; Topics in Applied Physics; Springer: Berlin/Heidelberg, Germany, 1981; Volume 43.
28. Parhi, K.K. *VLSI Digital Signal Processing Systems: Design and Implementation*; John Wiley & Sons: Hoboken, NJ, USA, 2007.
29. Ju, C.; Solomonik, E. Derivation and Analysis of Fast Bilinear Algorithms for Convolution. *arXiv* **2019**, arXiv:1910.13367.
30. Regalia, P.A.; Sanjit, M.K. Kronecker products, unitary matrices and signal processing applications. *SIAM Rev.* **1989**, *31*, 586–613. [CrossRef]
31. 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]
32. Saha, P.; Banerjee, A.; Dandapat, A.; Bhattacharyya, P. ASIC implementation of high speed processor for calculating discrete fourier transformation using circular convolution technique. *WSEAS Trans. Circuits Syst.* **2011**, *10*, 278–288.