electronics

MDPI

Article

Some Algorithms for Computing Short-Length Linear Convolution

Aleksandr Cariow * and Janusz P. Paplinski

Faculty of Computer Science and Information Technology, West Pomeranian University of Technology, Żołnierska 49, 71-210 Szczecin, Poland; jpaplinski@wi.zut.edu.pl

* Correspondence: acariow@wi.zut.edu.pl

Received: 19 October 2020; Accepted: 7 December 2020; Published: 10 December 2020

Abstract: In this article, we propose a set of efficient algorithmic solutions for computing short linear convolutions focused on hardware implementation in VLSI. We consider convolutions for sequences of length $N = 2, 3, 4, 5, 6, 7$, and 8. Hardwired units that implement these algorithms can be used as building blocks when designing VLSI -based accelerators for more complex data processing systems. The proposed algorithms are focused on fully parallel hardware implementation, but compared to the naive approach to fully parallel hardware implementation, they require from 25% to about 60% less, depending on the length N and hardware multipliers. Since the multiplier takes up a much larger area on the chip than the adder and consumes more power, the proposed algorithms are resource-efficient and energy-efficient in terms of their hardware implementation.

Keywords: linear convolution algorithms; fast hardware-oriented computations; convolution neural networks

1. Introduction

Discrete convolution is found in many applications in science and engineering. Above all, it plays a key role in modern digital signal and image processing. In digital signal processing, it is the basis of filtering, multiresolution decomposition, and optimization of the calculation of orthogonal transform [1–10]. In digital image processing, convolution is a basic operation of denoising, smoothing, edge detection, blurring, focusing, etc. [11–13]. There are two types of discrete convolutions: the cyclic convolution and the linear convolution. General principles for the synthesis of convolution algorithms were described in [1–3]. The main emphasis in these works was made primarily on the calculation of cyclic convolution, while in many digital signal and image processing applications, the calculation of linear convolutions is required.

In recent years, convolution has found unusually wide application in neural networks and deep learning. Among the various kinds of deep neural networks, convolutional neural networks (CNNs) are most widely used [14–17]. In CNNs, linear convolutions are the most computationally intensive operations, since in a typical implementation, their multiple computations occupy more than 90% of the CNN execution time [14]. Only one convolutional level in a typical CNN requires more than two thousand multiplications and additions. Usually, there are several such levels in the CNN. That is why developers of such type of networks seek and design efficient ways of implementing linear convolution using the smallest possible number of arithmetic operations.

To speed up linear convolution computation, various algorithmic methods have been proposed. The most common approach to effective calculating linear convolution is dipping it in the space of a double-size cyclic convolution with the subsequent application of a fast Fourier transform (FFT) algorithm [15–17]. The FFT-based linear convolution method is traditionally used for large length finite impulse response (FIR) filters; however, modern CNNs use predominantly small length FIR

filters. In this situation, the most effective algorithms used in the computation of a small-length linear convolution are the Winograd-like minimal filtering algorithms [18–20], which are the most used currently. The algorithms compute linear convolution over small tiles with minimal complexity, which makes them more effective with small filters and small batch sizes; however, these algorithms do not calculate the whole convolution. They calculate only two inner products of neighboring vectors formed from the current data stream by a moving time window of length N; therefore, these algorithms do not compute the true linear convolution.

At the same time, there are a number of CNNs in which it is necessary to calculate full-size small-length linear convolutions. In addition, in many applications of digital signal processing, there is the problem of calculating a one-dimensional convolution using its conversion into a multidimensional convolution. The algorithm thus obtained has a modular structure, and each module calculates a short-length one-dimensional convolution [21].

The most popular sizes of sequences being convoluted are sequences of length 2, 3, 4, 5, 6, 7, and 8. However, in the papers known to the authors, there is no description of resource-efficient algorithms for calculation of linear convolutions for lengths greater than four [1,4,6,21,22]. In turn, the solutions given in the literature for $N = 2$, $N = 3$, and $N = 4$ do not give a complete imagination about the organization of the linear convolution calculation process, since their corresponding signal flow graphs are not presented anywhere. In this paper, we describe a complete set of solutions for linear convolution of small length N sequences from 2 to 8.

2. Preliminaries

Let$\{h_m\}$ and, $\{x_n\}$, $m = 0, 1, \ldots, M-1$, $n = 0, 1, \ldots, N-1$ be two finite sequences of length M and N, respectively. Their linear convolution is the sequence $\{y_1\}$, $i = 0, 1, \ldots, M+N-2$ defined by [1]:

$$y_i = \sum_{n=0}^{N-1} h_{i-n} x_n, \tag{1}$$

where we take $h_{i-n} = 0$, if $i - n < 0$.

As a rule, the elements of one of the sequences to be convolved are constant numbers. For definiteness, we assume that it will be a sequence $\{h_m\}$.

Because sequences$\{x_n\}$ and $\{h_m\}$are finite length, then their linear convolution (1) can also be implemented as matrix-vector multiplication:

$$\mathbf{Y}_{(N+M-1)\times 1} = \mathbf{H}_{(N+M-1)\times N} \mathbf{X}_{N\times 1} \tag{2}$$

where

$$\mathbf{H}_{(N+M-1)\times N} = \begin{bmatrix} h_0 & & & \\ h_1 & h_0 & & \\ \vdots & h_1 & \ddots & \\ h_{M-1} & \vdots & \cdots & h_0 \\ & h_{M-1} & \cdots & h_1 \\ & & \ddots & \vdots \\ & & & h_{M-1} \end{bmatrix}, \tag{3}$$

$$\mathbf{X}_{N\times 1} = [x_0, x_1, \ldots x_{N-1}]^{\mathrm{T}},$$

$$\mathbf{Y}_{(N+M-1)\times 1} = [y_0, y_1, \ldots, y_{N+M-2}]^{\mathrm{T}},$$

$$\mathbf{H}_{M\times 1} = [h_0, h_1, \ldots, h_{M-1}]^{\mathrm{T}}.$$

In the future, we assume that $\mathbf{X}_{N\times 1}$ is a vector of input data, $\mathbf{Y}_{(N+M-1)\times 1}$ is a vector of output data, and $\mathbf{H}_{M\times 1}$ is a vector containing constants.

Direct computation of (3) takes MN multiplications and $(M-1)(N-1)$ addition. This means that the fully parallel hardware implementation of the linear convolution operation requires MN multipliers and $N+M-3$ multi-input adders with a different number of inputs, which depends on the length of the sequences being convolved. Traditionally, the convolution for which $M = N$ is assumed as a basic linear convolution operation. Resource-effective cyclic convolution algorithms for benchmark lengths (N = 2, 3,...,16) have long been published [1–9]. For linear convolution, optimized algorithms are described only for the cases N = 2, 3, 4 [4,6,21,22]. Below we show how to reduce the implementation complexity of some benchmark-lengths linear convolutions for the case of completely parallel hardware their implementation. For completeness, we also consider algorithms for the sequences of lengths $M = N$ = 2, 3, and 4.

So, considering the above, the goal of this article is to develop and describe fully parallel resource-efficient algorithms for N = 2, 3, 4, 5, 6, 7, 8.

3. Algorithms for Short-Length Linear Convolution

The main idea of presented algorithm is to transform the linear convolution matrix into circular matrix and two Toeplitz matrices. Then we can rewrite (3) in following form:

$$\mathbf{Y}_{(2N-1)\times 1} = \mathbf{H}_{(2N-1)\times N}\mathbf{X}_{N\times 1} = \left[\begin{bmatrix}\mathbf{H}_{K\times N}\\ \check{\mathbf{H}}_{N}\\ \mathbf{H}_{L\times N}\end{bmatrix} - \begin{bmatrix}\mathbf{0}_{K\times N}\\ \mathbf{H}_{L\times N}\\ \mathbf{0}_{1\times N}\\ \mathbf{H}_{K\times N}\\ \mathbf{0}_{L\times N}\end{bmatrix}\right]\mathbf{X}_{N\times 1} \tag{4}$$

where $\mathbf{H}_{K\times N} = \begin{bmatrix}\mathbf{T}_{K}^{(l)} & \mathbf{0}_{K\times(N-K)}\end{bmatrix}$ and $\mathbf{H}_{L\times N} = \begin{bmatrix}\mathbf{0}_{L\times(N-L)} & \mathbf{T}_{L}^{(r)}\end{bmatrix}$ are matrices that are horizontal concatenations of null matrices and left-triangular or right-triangular Toeplitz matrices, respectively:

$$\mathbf{T}_{K\times N}^{(l)} = \begin{bmatrix} h_0 & 0 & \cdots & 0\\ h_1 & h_0 & \cdots & 0\\ \vdots & \vdots & \ddots & \vdots\\ h_{K-1} & h_{K-2} & \cdots & h_0\end{bmatrix}, \quad \mathbf{T}_{L}^{(r)} = \begin{bmatrix} h_{N-1} & h_{N-2} & \cdots & h_{N-L-2}\\ 0 & h_{N-1} & \cdots & h_{N-L-1}\\ \vdots & \vdots & \ddots & \vdots\\ 0 & 0 & \cdots & h_{N-1}\end{bmatrix},$$

which gives

$$\mathbf{H}_{K\times N} = \begin{bmatrix} h_0 & 0 & \cdots & 0 & 0 & 0 & \cdots & 0\\ h_1 & h_0 & \cdots & 0 & 0 & 0 & \vdots & 0\\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots\\ h_{K-1} & h_{K-2} & \cdots & h_0 & 0 & 0 & \cdots & 0\end{bmatrix},$$

$$\mathbf{H}_{L\times N} = \begin{bmatrix} 0 & 0 & \cdots & 0 & h_{N-1} & h_{N-2} & \cdots & h_{N-L-2}\\ 0 & 0 & \cdots & 0 & 0 & h_{N-1} & \cdots & h_{N-L-1}\\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots\\ 0 & 0 & \cdots & 0 & 0 & 0 & \cdots & h_{N-1}\end{bmatrix}.$$

A circulant matrix $\check{\mathbf{H}}_N$ is a matrix of cyclic convolution $\mathbf{H}_N$ with rows cyclically shifted by n positions down:

$$\breve{\mathbf{H}}_N = \mathbf{I}_N^{(\leftarrow n)}\mathbf{H}_N = \begin{bmatrix} h_K & h_{K-1} & \cdots & h_{K+1} \\ h_{K+1} & h_K & \cdots & h_{K+2} \\ \vdots & \vdots & \ddots & \vdots \\ h_{N-1} & h_{N-2} & \cdots & h_0 \\ h_0 & h_{N-1} & \cdots & h_1 \\ \vdots & \vdots & \ddots & \vdots \\ h_{K-1} & h_{K-2} & \cdots & h_K \end{bmatrix},$$

where $\mathbf{I}_N^{(\leftarrow n)}$- is permutation matrix obtained from the identity matrix by cyclic shift of its columns by n positions to the left and:

$$\mathbf{H}_N = \begin{bmatrix} h_0 & h_{N-1} & \cdots & h_1 \\ h_1 & h_0 & \cdots & h_2 \\ \vdots & \vdots & \vdots & \vdots \\ h_{N-1} & h_{N-2} & \cdots & h_0 \end{bmatrix}.$$

The coefficients K and L are natural numbers arbitrary taken and fulfilling the dependence $K + L = N - 1$. These values are selected heuristically for each N separately.

The product $\breve{\mathbf{H}}_N\mathbf{X}_{N\times 1}$ is calculated using the well-known fast convolution algorithm. The products of $\mathbf{H}_{K\times N}\mathbf{X}_{N\times 1}$ and $\mathbf{H}_{L\times N}\mathbf{X}_{N\times 1}$ are also calculated using fast algorithms for matrix-vector multiplication with Toeplitz matrices. We use all of the above techniques to synthesize the final short-length linear convolution algorithms with reduced multiplicative complexity.

3.1. Algorithm for N = 2

Let $\mathbf{X}_{2\times 1} = [x_0, x_1]^{\mathrm{T}}$ and $\mathbf{H}_{2\times 1} = [h_0, h_1]^{\mathrm{T}}$ be 2-dimensional data vectors being convolved and $\mathbf{Y}_{3\times 1} = [y_0, y_1, y_2]^{\mathrm{T}}$ be an input vector representing a linear convolution. The problem is to calculate the product

$$\mathbf{Y}_{3\times 1} = \mathbf{H}_{3\times 2}\mathbf{X}_{2\times 1}, \tag{5}$$

where

$$\mathbf{H}_{3\times 2} = \begin{bmatrix} h_0 & \\ h_1 & h_0 \\ & h_1 \end{bmatrix}.$$

Direct computation of (5) takes four multiplications and one addition. It is easy to see that the matrix $\mathbf{H}_{3\times 2}$ possesses an uncommon structure. By the Toom–Cook algorithmic trick, the number of multiplications in the calculation of the 2-point linear convolution can be reduced [1,21].

With this in mind, the rationalized computational procedure for computing 2-point linear convolution has the following form:

$$\mathbf{Y}_{3\times 1} = \mathbf{A}_3^{(2)}\mathbf{D}_3\mathbf{A}_{3\times 2}^{(2)}\mathbf{X}_{2\times 1} \tag{6}$$

where

$$\mathbf{A}_{3\times 2}^{(2)} = \begin{bmatrix} 1 & \\ 1 & -1 \\ & 1 \end{bmatrix}, \quad \mathbf{A}_3^{(2)} = \begin{bmatrix} 1 & & \\ 1 & -1 & 1 \\ & & 1 \end{bmatrix}, \quad \mathbf{D}_3 = diag(h_0, h_0 - h_1, h_1), \tag{7}$$

$$s_0^{(2)} = h_0, \quad s_1^{(2)} = h_0 - h_1, \quad s_2^{(2)} = h_1. \tag{8}$$

Figure 1 shows a signal flow graph for the proposed algorithm, which also provides a simplified schematic view of a fully parallel processing unit for resource-effective implementing of 2-point linear convolution. In this paper, the all data flow graphs are oriented from left to right. Straight lines in the figures denote the data transfer (data path) operations. The circles in these figures show the

operation of multiplication (multipliers in the case of hardware implementation) by a number inscribed inside a circle. Points where lines converge denote summation (adders in the case of hardware implementation) and dotted lines indicate the sign-change data paths (data paths with multiplication by -1) . We use the usual lines without arrows on purpose, so as not to clutter the picture [23].

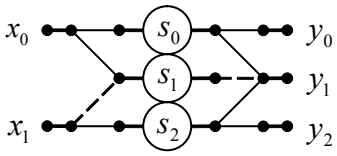

Figure 1. The signal flow graph of the proposed algorithm for computation of 2-point linear convolution.

As it can be seen, the calculation of 2-point linear convolution requires only three multiplications and three additions. In terms of arithmetic units, a fully parallel hardware implementation of the processor unit for calculating a 2-point convolution will require three multipliers, one two-input adder, and one three-input adder. In terms of arithmetic units, a fully parallel hardware implementation of the processor unit for calculating a 2-point convolution will require three multipliers, one two-input adder, and one three-input adder instead of four multipliers and one two-input adder in the case of completely parallel implementation of (6). So, we exchanged one multiplier for one three-input adder.

3.2. Algorithm for N = 3

Let $\mathbf{X}_{3\times1} = [x_0, x_1, x_2]^{\mathrm{T}}$ and $\mathbf{H}_{3\times1} = [h_0, h_1, h_2]^{\mathrm{T}}$ be 3-dimensional data vectors being convolved and $\mathbf{Y}_{5\times1} = [y_0, y_1, y_2, y_3, y_4]^{\mathrm{T}}$ be an input vector represented linear convolution for $N = 3$. The problem is to calculate the product

$$\mathbf{Y}_{5\times1} = \mathbf{H}_{5\times3}\mathbf{X}_{3\times1}, \tag{9}$$

where

$$\mathbf{H}_{5\times3} = \begin{bmatrix} h_0 & & \\ h_1 & h_0 & \\ h_2 & h_1 & h_0 \\ & h_2 & h_1 \\ & & h_2 \end{bmatrix}. \tag{10}$$

Direct computation of (9) takes nine multiplications and five addition. Because the matrix $\mathbf{H}_{5\times3}$ also possesses uncommon structure, the number of multiplications in the calculation of the 3-point linear convolution can be reduced too [1,4,21].

An algorithm for computation 3-point linear convolution with reduced multiplicative complexity can be written with the help of following matrix-vector calculating procedure:

$$\mathbf{Y}^{(3)}_{5\times1} = \mathbf{A}^{(3)}_{5\times6}\mathbf{D}^{(3)}_{6}\mathbf{A}^{(3)}_{6\times3}\mathbf{X}_{3\times1}, \tag{11}$$

where

$$\mathbf{A}^{(3)}_{6\times3} = \begin{bmatrix} 1 & & \\ & 1 & \\ & & 1 \\ 1 & 1 & \\ 1 & & 1 \\ & 1 & 1 \end{bmatrix}, \quad \mathbf{A}^{(3)}_{5\times6} = \begin{bmatrix} 1 & & & & & \\ -1 & -1 & & 1 & & \\ -1 & 1 & -1 & & 1 & \\ & -1 & -1 & & & 1 \\ & & 1 & & & \end{bmatrix}, \tag{12}$$

$$\mathbf{D}^{(3)}_{6} = diag(h_0, h_1, h_2, h_0 + h_1, h_0 + h_2, h_1 + h_2). \tag{13}$$

Figure 2 shows a signal flow graph of the proposed algorithm for the implementation of 3-point linear convolution. As it can be seen, the calculation of 3-point linear convolution requires only

6 multiplications and 10 additions. Thus, the proposed algorithm saves six multiplications at the cost of six extra additions compared to the ordinary matrix-vector multiplication method.

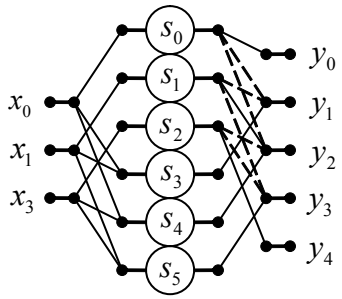

Figure 2. The signal flow graph of the proposed algorithm for computation of 3-point linear convolution.

In terms of arithmetic units, a fully parallel hardware implementation of the processor unit for calculating a 3-point linear convolution (11) will require only six multipliers, four two-input adders, one three-input adder, and one four-input adder instead of 12 multipliers, 2 two-input adders, and 1 two-input adder in the case of fully parallel implementation of (9).

3.3. Algorithm for N = 4

Let $\mathbf{X}_{4\times1} = [x_0, x_1, x_2, x_3]^{\mathrm{T}}$ and $\mathbf{H}_{4\times1} = [h_0, h_1, h_2, h_3]^{\mathrm{T}}$ be 4-dimensional data vectors being convolved and $\mathbf{Y}_{7\times1} = [y_0, y_1, y_2, y_3, y_4, y_5, y_6]^{\mathrm{T}}$ be an input vector represented linear convolution for $N = 4$.

The problem is to calculate the product:

$$\mathbf{Y}_{7\times1} = \mathbf{H}_{7\times4}\mathbf{X}_{4\times1}, \tag{14}$$

where

$$\mathbf{H}_{7\times4} = \begin{bmatrix} h_0 & & & \\ h_1 & h_0 & & \\ h_2 & h_1 & h_0 & \\ h_3 & h_2 & h_1 & h_0 \\ & h_3 & h_2 & h_1 \\ & & h_3 & h_2 \\ & & & h_3 \end{bmatrix}.$$

Direct computation of (14) takes 16 multiplications and 9 addition. Due to the specific structure of matrix $\mathbf{H}_{7\times4}$, the number of multiplication operations in the calculation of (14) can be significantly reduced.

An algorithm for computation 4-point linear convolution with reduced multiplicative complexity can be written using the following matrix-vector calculating procedure:

$$\mathbf{Y}_{7\times1} = \mathbf{A}_{7\times8}^{(4)}\mathbf{A}_{8\times9}^{(4)}\mathbf{D}_{9}^{(4)}\mathbf{A}_{9\times8}^{(4)}\mathbf{A}_{8\times4}^{(4)}\mathbf{X}_{4\times1}, \tag{15}$$

where

$$\mathbf{A}_{8\times4}^{(4)} = \begin{bmatrix} 1 & & & \\ 1 & & 1 & \\ & 1 & & 1 \\ 1 & & -1 & \\ & 1 & & -1 \\ & & 1 & \\ & & & 1 \\ & & & 1 \end{bmatrix}, \quad \mathbf{A}_{9\times8}^{(4)} = 1 \oplus \mathbf{H}_2 \oplus \begin{bmatrix} 1 & \\ & 1 \\ 1 & 1 \end{bmatrix} \oplus \mathbf{I}_3,$$

where $\mathbf{I}_N$ is an identity $N \times N$ matrix, $\mathbf{H}_2$ is the (2×2) Hadamard matrix, and sign ,"$\oplus$" denotes direct sum of two matrices [23–25].

$$\mathbf{D}_9^{(4)} = diag(s_0^{(4)} s_1^{(4)}, ..., s_8^{(4)}),$$

$$s_0^{(4)} = h_0, \quad s_1^{(4)} = (h_0 + h_1 + h_2 + h_3)/4\,, \quad s_2^{(4)} = (h_0 - h_1 + h_2 - h_3)/4\,,$$
$$s_3^{(4)} = (h_0 - h_1 - h_2 + h_3)/2, \quad s_4^{(4)} = (h_0 + h_1 - h_2 - h_3)/2, \quad s_5^{(4)} = (h_0 - h_2)/2\,, \quad s_6^{(4)} = h_3\,,$$
$$s_7^{(4)} = h_2\,, \quad s_8^{(4)} = h_3,$$

$$\mathbf{A}_{8\times 9}^{(4)} = 1 \oplus \mathbf{H}_2 \oplus \begin{bmatrix} & & -1 & 1 \\ -1 & & & 1 \end{bmatrix} \oplus \mathbf{I}_3, \quad \mathbf{A}_{7\times 8}^{(4)} = \left[\begin{array}{c|ccc|ccc} 1 & & & & & & \\ \hline & & 1 & & 1 & -1 & -1 \\ & 1 & & -1 & & & -1 \\ & & 1 & & -1 & & \\ \hline -1 & 1 & & 1 & & & \\ \hline & & & & & 1 & 1 \\ & & & & & & 1 \end{array}\right].$$

Figure 3 shows a signal flow graph of the proposed algorithm for the implementation of 4-point linear convolution. So, the proposed algorithm saves 7 multiplications at the cost of 11 extra additions compared to the ordinary matrix-vector multiplication method.

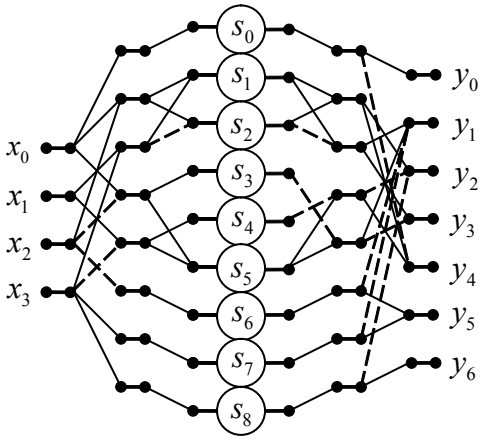

Figure 3. The signal flow graph of the proposed algorithm for computation of 4-point linear convolution.

In terms of arithmetic units, a fully parallel hardware implementation of the processor unit for calculating a 4-point linear convolution will require only 9 multipliers, 13 two-input adders, 2 three-input adders, and 1 four-input adder instead of 16 multipliers, 2 two-input adders, 2 three-input adders, and 1 four-input adder in the case of fully parallel implementation of (14).

3.4. Algorithm for N = 5

Let $\mathbf{X}_{5\times 1} = [x_0, x_1, x_2, x_3, x_4]^{\mathrm{T}}$ and $\mathbf{H}_{5\times 1} = [h_0, h_1, h_2, h_3, h_4]^{\mathrm{T}}$ be 5-dimensional data vectors being convolved and $\mathbf{Y}_{9\times 1} = [y_0, y_1, y_2, y_3, y_4, y_5, y_6, y_7, y_8]^{\mathrm{T}}$ be an input vector representing a linear convolution for $N = 5$.

The problem is to calculate the product:

$$\mathbf{Y}_{9\times 1} = \mathbf{H}_{9\times 5}\mathbf{X}_{5\times 1}, \tag{16}$$

where

$$\mathbf{H}_{9\times5} = \begin{bmatrix} h_0 & & & \\ h_1 & h_0 & & \\ \vdots & h_1 & \ddots & \\ h_4 & \vdots & \ddots & h_0 \\ & h_4 & \ddots & h_1 \\ & & \ddots & \vdots \\ & & & h_4 \end{bmatrix}.$$

Direct computation of (16) takes 25 multiplications and 16 addition. Due to the specific structure of matrix $\mathbf{H}_{9\times5}$, the number of multiplication operations in the calculation of (16) can be significantly reduced.

Thus, an algorithm for computation 5-point linear convolution with reduced multiplicative complexity can be written using the following matrix-vector calculating procedure:

$$\mathbf{Y}_{9\times1} = \mathbf{A}^{(5)}_{9\times11}\mathbf{A}^{(5)}_{11\times13}\mathbf{A}^{(5)}_{13\times16}\mathbf{D}^{(5)}_{16}\mathbf{A}^{(5)}_{16\times15}\mathbf{A}^{(5)}_{15\times11}\mathbf{A}^{(5)}_{11\times5}\mathbf{X}_{5\times1}, \tag{17}$$

where

$$\mathbf{A}^{(5)}_{11\times5} = \left[\begin{array}{ccc|cc} 1 & & & & \\ 1 & & & \multicolumn{2}{c}{\mathbf{0}_3} \\ & 1 & & & \\ \hline 1 & & & & -1 \\ & 1 & & & -1 \\ & & 1 & & -1 \\ & & & 1 & -1 \\ 1 & 1 & 1 & 1 & 1 \\ \hline & & & 1 & \\ \multicolumn{3}{c|}{\mathbf{0}_3} & & 1 \\ & & & & 1 \end{array}\right], \quad \mathbf{A}^{(5)}_{15\times11} = \left[\begin{array}{ccccccc|c} 1 & & & & & & & \\ & 1 & & & & & & \\ & & 1 & & & & & \\ & & & 1 & & & & \\ & & & & 1 & & & \\ & & & 1 & 1 & & & \multicolumn{1}{c}{\mathbf{0}_{11\times4}} \\ & & & & & 1 & & \\ & & & & & & 1 & \\ & & & & & 1 & 1 & \\ & & & 1 & & -1 & & \\ & & & & 1 & & -1 & \\ \hline \multicolumn{5}{c}{\mathbf{0}_{4\times7}} & & & \multicolumn{1}{c}{\mathbf{I}_4} \end{array}\right]$$

and $\mathbf{0}_{M\times N}$ is a null matrix of order $M \times N$ [23–25],

$$\mathbf{D}^{(5)}_{16} = diag(s^{(5)}_0 s^{(5)}_1, ..., s^{(5)}_{15}),$$

$$s^{(5)}_0 = h_0\,,\quad s^{(5)}_1 = h_1\,,\quad s^{(5)}_2 = h_0\,,\quad s^{(5)}_3 = (h_0 - h_2 + h_3 - h_4)/4,\quad s^{(5)}_4 = (h_1 - h_2 + h_3 - h_4)/4,$$
$$s^{(5)}_5 = (3h_2 - 2h_1 + 2h_0 - 2h_3 + 3h_4)/5,\quad s^{(5)}_6 = (-h_0 + h_1 - h_2 + h_3),\quad s^{(5)}_7 = (-h_0 + h_1 - h_2 + h_3),$$
$$s^{(5)}_8 = (3h_0 - 2h_1 + 3h_2 - 2h_3 - 2h_4)/5\,,\quad s^{(5)}_9 = -h_2 + h_3\,,\quad s^{(5)}_{10} = h_1 - h_2\,,$$
$$s^{(5)}_{11} = (-h_0 - h_1 + 4h_2 - h_3 - h_4)/5\,,\quad s^{(5)}_{12} = (h_0 + h_1 + h_2 + h_3 + h_4)/5\,,\quad s^{(5)}_{13} = h_4\,,\quad s^{(5)}_{14} = h_3\,,$$
$$s^{(5)}_{15} = h_4\,,$$

$$\mathbf{A}^{(5)}_{16\times 15} = \left[\begin{array}{ccccccccccccccc|c} 1 & & & & & & & & & & & & & & & \\ & 1 & & & & & & & & & & & & & & \\ & & 1 & & & & & & & & & & & & & \\ & & & 1 & & & & & & & & & & & & \\ & & & & 1 & & & & & & & & & & & \\ & & & & & 1 & & & & & & & & & & \\ & & & & & & 1 & & & & & & & & & \\ & & & & & & & 1 & & & & & & & & \\ & & & & & & & & 1 & & & & & & & \\ & & & & & & & & & 1 & & & & & & \\ & & & & & & & & & & 1 & & & & & \\ & & & & & & & & & & & 1 & & & & \\ & & & & & & & & & & & & 1 & & & \\ \hline & & & & & & & & & & & & & 1 & -1 & \\ \hline & & & & & & & & & & & & & & & \mathbf{I}_4 \end{array}\right], \quad \mathbf{0}_{12\times 4}, \quad \mathbf{0}_{4\times 11}$$

$$\mathbf{A}^{(5)}_{13\times 16} = \mathbf{I}_3 \oplus \left[\begin{array}{cccccccc|cccc} 1 & & & 1 & & & & & & & & \\ & 1 & 1 & & & & & & & \mathbf{0}_3 & & \\ & & & & 1 & & & 1 & & & & \\ \hline & & & & & & 1 & 1 & & & & \\ & \mathbf{0}_3 & & & & & & & 1 & & & 1 \\ & & & & & & & & & 1 & 1 & \end{array}\right] \oplus \mathbf{I}_4,$$

$$\mathbf{A}^{(5)}_{11\times 13} = \mathbf{I}_3 \oplus \left[\begin{array}{ccccccc|ccc} 1 & & & & & & & & -1 & & 1 \\ -1 & -1 & -1 & -1 & & & & & & & 1 \\ \hline & & & & & & 1 & & & 1 & 1 \\ & & & & 1 & & & & 1 & & 1 \\ & & & 1 & & & & & & -1 & 1 \end{array}\right] \oplus \mathbf{I}_3,$$

$$\mathbf{A}^{(5)}_{9\times 11} = \left[\begin{array}{ccccc|ccccc} 1 & & & & & & & \mathbf{0}_{2\times 6} & & & \\ & 1 & & 1 & & & & & & & \\ \hline & & & & & & 1 & & -1 & -1 & \\ & & \mathbf{0}_{3\times 5} & & & & & 1 & & & -1 \\ & & & & & & & & 1 & & \\ \hline -1 & & & & 1 & & & & & & \\ & & & & & & & \mathbf{0}_{2\times 6} & & & \\ & -1 & & -1 & & 1 & & & & & \\ \hline & & & & & & & & 1 & 1 & \\ & & \mathbf{0}_{2\times 5} & & & & & & & & \\ & & & & & & & & & & 1 \end{array}\right].$$

Figure 4 shows a data flow diagram of the proposed algorithm for the implementation of 5-point linear convolution. The algorithm saves 9 multiplications at the cost of 22 extra additions compared to the ordinary matrix-vector multiplication method.

In terms of arithmetic units, a fully parallel hardware implementation of the processor unit for calculating a 5-point linear convolution will require 16 multipliers, 20 two-input adders, 2 three-input adders, 1 four-input adder, and 1 five-input adder instead of 25 multipliers, 2 two-input adders, 2 three-input adders, 2 four-input adders, and 1 five-input adder in the case of fully parallel implementation of expression (16).

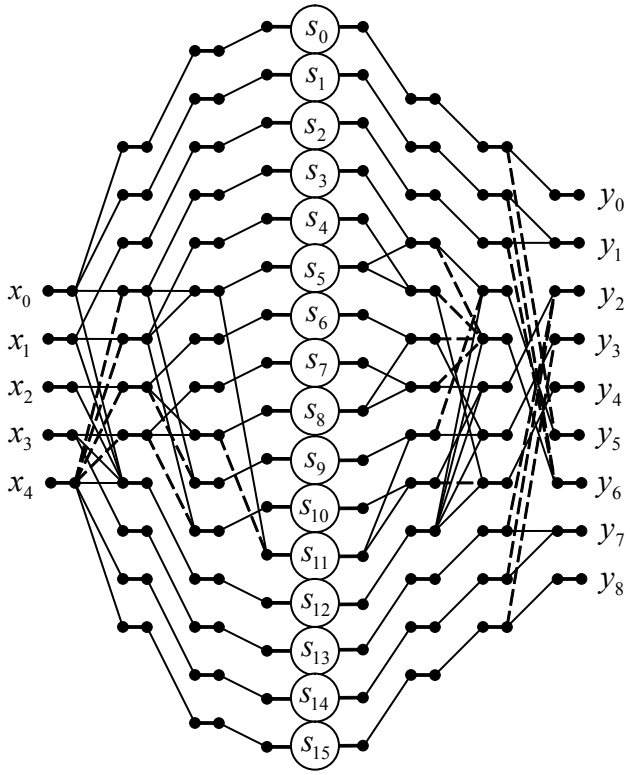

Figure 4. The signal flow graph of the proposed algorithm for computation of 5-point linear convolution.

3.5. *Algorithm for N = 6*

Let $\mathbf{X}_{6\times1} = [x_0, x_1, x_2, x_3, x_4, x_5]^{\mathrm{T}}$ and $\mathbf{H}_{6\times1} = [h_0, h_1, h_2, h_3, h_4, h_5]^{\mathrm{T}}$ be 6-dimensional data vectors being convolved and $\mathbf{Y}_{11\times1} = [y_0, y_1, y_2, y_3, y_4, y_5, y_6, y_7, y_8, y_9, y_{10}]^{\mathrm{T}}$ be an input vector representing a linear convolution for $N = 6$.

The problem is to calculate the product:

$$\mathbf{Y}_{11\times1} = \mathbf{H}_{11\times6}\mathbf{X}_{6\times1}, \tag{18}$$

where

$$\mathbf{H}_{11\times6} = \begin{bmatrix} h_0 & & & \\ h_1 & h_0 & & \\ \vdots & h_1 & \ddots & \\ h_5 & \vdots & \ddots & h_0 \\ & h_5 & \ddots & h_1 \\ & & \ddots & \vdots \\ & & & h_5 \end{bmatrix}.$$

Direct computation of (18) takes 36 multiplications and 25 addition. We proposed an algorithm that takes only 16 multiplications and 44 additions. It saves 20 multiplications at the cost of 19 extra additions compared to the ordinary matrix-vector multiplication method.

Proposed algorithm for computation 6-point linear convolution can be written with the help of following matrix-vector calculating procedure:

$$Y_{11}^{(6)} = \mathbf{A}_{11}^{(6)}\breve{\mathbf{A}}_{11}^{(6)}\mathbf{A}_{11\times14}^{(6)}\hat{\mathbf{A}}_{14}^{(6)}\breve{\mathbf{A}}_{14}^{(6)}\mathbf{A}_{14\times16}^{(6)}\mathbf{D}_{16}^{(6)}\mathbf{A}_{16\times14}^{(6)}\breve{\mathbf{A}}_{14}^{(6)}\mathbf{A}_{14}^{(6)}\mathbf{A}_{14\times6}^{(6)}\mathbf{X}_{6\times1}, \tag{19}$$

where

$$\mathbf{A}_{14\times 6}^{(6)}=\left[\begin{array}{cc|c} 1 & & \\ 1 & & \\ & 1 & \mathbf{0}_{5\times 3} \\ 1 & & \\ & & \\ & 1 & \\ \hline & & 1 \\ \mathbf{0}_3 & & 1 \\ & & \\ & & 1 \\ \hline 1 & & \\ \hline & & 1 \\ \mathbf{0}_3 & & 1 \\ & & \\ & & 1 \\ \hline & & 1 \quad 1 \\ \mathbf{0}_{2\times 3} & & \\ & & 1 \end{array}\right],\quad \mathbf{A}_{14}^{(6)}=\mathbf{I}_3\oplus(\mathbf{H}_2\otimes\mathbf{I}_3)\oplus\mathbf{I}_5,$$

$$\check{\mathbf{A}}_{14}^{(6)}=\mathbf{I}_3\oplus\left(\mathbf{I}_2\otimes\begin{bmatrix}1 & 1 & 1\\ 1 & -1 & \\ 1 & & -1\end{bmatrix}\right)\oplus\mathbf{I}_5,\quad \mathbf{A}_{16\times 14}^{(6)}=\mathbf{I}_3\oplus\left(\mathbf{I}_2\otimes\begin{bmatrix}1 & & \\ & & 1\\ & 1 & \\ & 1 & 1\end{bmatrix}\right)\oplus\mathbf{I}_5,$$

$$\mathbf{D}_{16}=diag(s_0^{(6)},s_1^{(6)},...,s_{15}^{(6)}),$$

$$s_0^{(6)}=6h_0\,,\quad s_1^{(6)}=6h_1\,,\quad s_2^{(6)}=6h_0\,,\quad s_3^{(6)}=h_0+h_3+h_4+h_1+h_2+h_5\,,$$
$$s_4^{(6)}=3(h_4+h_1-h_0-h_3),\quad s_5^{(6)}=3(h_2+h_5-h_0-h_3),$$
$$s_6^{(6)}=3(h_0+h_3)-(h_0+h_3+h_4+h_1+h_2+h_5),\quad s_7^{(6)}=h_0-h_3+h_4-h_1+h_2-h_5),$$
$$s_8^{(6)}=3(h_4-h_1-h_0+h_3),\quad s_9^{(6)}=3(h_2-h_5-h_0+h_3),$$
$$s_{10}^{(6)}=3(h_0+h_3)-(h_0-h_3+h_4-h_1+h_2-h_5),\quad s_{11}^{(6)}=6h_5\,,\quad s_{12}^{(6)}=6(-h_4+h_5),$$
$$s_{13}^{(6)}=6(h_3-h_4),\quad s_{14}^{(6)}=6h_4\,,\quad s_{15}^{(6)}=6h_5\,,$$

$$\mathbf{A}_{14\times 16}^{(6)}=\mathbf{I}_4\oplus\left[\begin{array}{cc|c|cc} 1 & & 1 & & \\ & 1 & 1 & & \mathbf{0}_{2\times 4} \\ \hline & \mathbf{0}_{1\times 3} & & 1 & \mathbf{0}_{1\times 3} \\ \hline & \mathbf{0}_{2\times 4} & & 1 & 1 \\ & & & & 1 \quad 1 \end{array}\right]\oplus\mathbf{I}_5,\quad \hat{\mathbf{A}}_{14}^{(6)}=\mathbf{I}_4\oplus\left[\begin{array}{ccccc} & 1 & & & \\ 1 & & & & \\ & & 1 & & \\ & & & & 1 \\ & & & 1 & \end{array}\right]\oplus\mathbf{I}_5,$$

$$\mathbf{A}_{11\times 14}^{(6)}=\begin{bmatrix}1 & & & \\ & 1 & 1 & \end{bmatrix}\oplus(\mathbf{H}_2\otimes\mathbf{I}_3)\oplus\begin{bmatrix}1 & & 1 & 1 & \\ & 1 & & 1 & \\ & & & & 1\end{bmatrix},\quad \check{\mathbf{A}}_{11}^{(6)}=\left[\begin{array}{ccc|ccc} \multicolumn{3}{c|}{\mathbf{I}_3} & \multicolumn{2}{c|}{\mathbf{0}_{3\times 4}} & \mathbf{0}_3 \\ \hline & & & & 1 & \\ & & & 1 & & \\ \mathbf{0}_{4\times 3} & & & & 1 & \mathbf{0}_{4\times 3} \\ & & & 1 & & \\ \hline \multicolumn{3}{c|}{\mathbf{0}_{4\times 3}} & \multicolumn{2}{c|}{\mathbf{0}_{4\times 3}} & \mathbf{I}_4 \end{array}\right],$$

$$\mathbf{A}_{11}^{(6)} = \left[\begin{array}{c|c|c} \begin{matrix}1 & \\ & 1\end{matrix} & \mathbf{0}_{2\times 7} & \\ \hline & \begin{matrix}1 & & & \\ & 1 & & \\ & & 1 & \\ & & & 1\end{matrix} & \begin{matrix}-1 & & \\ & -1 & \\ & & -1\end{matrix} \\ \mathbf{0}_4 & & \\ \hline \begin{matrix}-1 & & 1 & \\ & -1 & & 1\end{matrix} & \mathbf{0}_{2\times 7} & \\ \hline & & \begin{matrix}1 & \\ & 1 \\ & & 1\end{matrix} \\ \mathbf{0}_{3\times 4} & \mathbf{0}_{3\times 4} & \end{array}\right]$$

and sign "$\otimes$" denotes tensor or Kronecker product of two matrices [23–25].

Figure 5 shows a data flow diagram of the proposed algorithm for the implementation of 6-point linear convolution.

In terms of arithmetic units, a fully parallel hardware implementation of the processor unit for calculating a 6-point linear convolution will require 16 multipliers, 32 two-input adders, and 5 three-input adders, instead of 36 multipliers, 2 two-input adders, 2 three-input adders, 2 four-input adders, 2 five-input adders, and 1 six-input adder in the case of completely parallel implementation of expression (18).

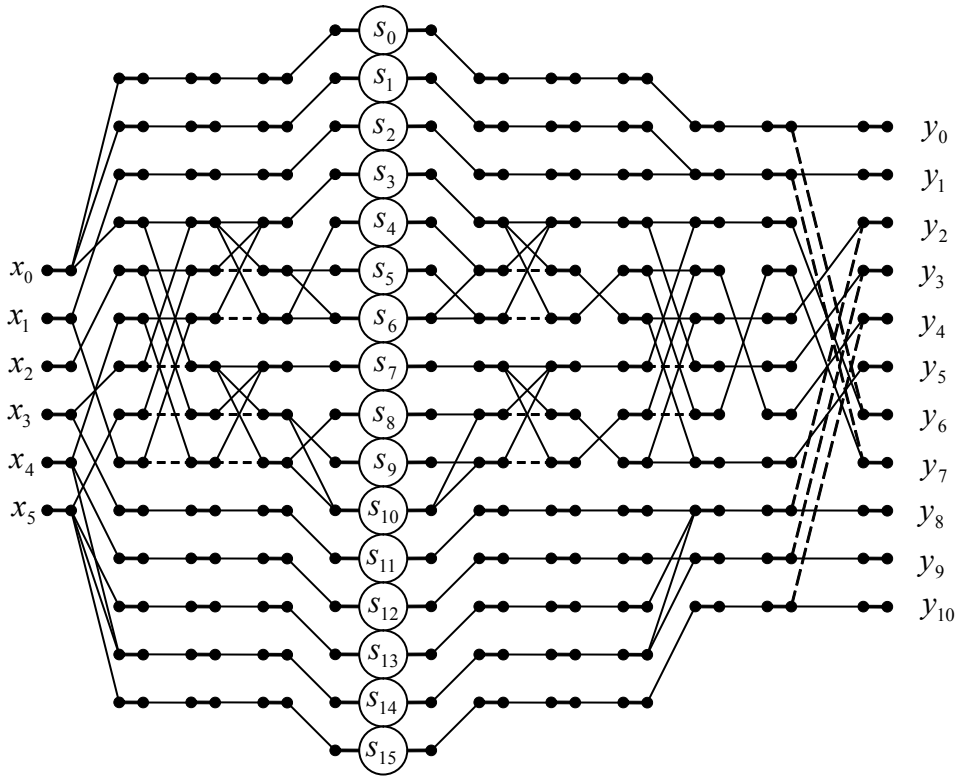

Figure 5. The signal flow graph of the proposed algorithm for computation of 6-point linear convolution.

3.6. Algorithm for N = 7

Let $\mathbf{X}_{7\times 1} = [x_0, x_1, x_2, x_3, x_4.x_5, x_6]^{\mathrm{T}}$ and $\mathbf{H}_{7\times 1} = [h_0, h_1, h_2, h_3, h_4, h_5, h_6],^{\mathrm{T}}$ be 7-dimensional data vectors being convolved and $\mathbf{Y}_{13\times 1} = [y_0, y_1, y_2, y_3, y_4, y_5, y_6, y_7, y_8, y_9, y_{10}, y_{11}, y_{12}]^{\mathrm{T}}$ be an input vector representing a linear convolution for $N = 7$.

The problem is to calculate the product:

$$\mathbf{Y}_{13\times1} = \mathbf{H}_{13\times7}\mathbf{X}_{7\times1}, \tag{20}$$

where

$$\mathbf{H}_{13\times7} = \begin{bmatrix} h_0 & & & \\ h_1 & h_0 & & \\ \vdots & h_1 & \ddots & \\ h_6 & \vdots & \ddots & h_0 \\ & h_6 & \ddots & h_1 \\ & & \ddots & \vdots \\ & & & h_6 \end{bmatrix}.$$

Direct computation of (20) takes 49 multiplications and 36 addition. We developed an algorithm that contains only 26 multiplications and 79 additions. It saves 23 multiplications at the cost of 43 extra additions compared to the ordinary matrix-vector multiplication method.

The proposed algorithm for computation 7-point linear convolution with reduced multiplicative complexity can be written using the following matrix-vector calculating procedure:

$$\mathbf{Y}^{(7)}_{13\times1} = \mathbf{A}^{(7)}_{13\times26}\mathbf{D}^{(7)}_{26}\mathbf{A}^{(7)}_{26\times7}\mathbf{X}_{7\times1}, \tag{21}$$

where

$$\mathbf{A}^{(7)}_{13\times26} = \mathbf{A}^{(7)}_{13\times15}\mathbf{A}^{(7)}_{15\times20}\mathbf{A}^{(7)}_{20\times21}\mathbf{A}^{(7)}_{21\times22}\mathbf{A}^{(7)}_{22\times25}\mathbf{A}^{(7)}_{25\times21}\mathbf{A}^{(7)}_{21}\mathbf{A}^{(7)}_{21\times26},$$

$$\mathbf{A}^{(7)}_{26\times7} = \mathbf{A}^{(7)}_{26}\mathbf{A}^{(7)}_{26\times28}\mathbf{A}^{(7)}_{28\times21}\mathbf{A}^{(7)}_{21\times18}\mathbf{A}^{(7)}_{18\times7}$$

and

$$\mathbf{A}^{(7)}_{18\times7} = \left[\begin{array}{ccc|c} 1 & & & \\ 1 & & & \\ & 1 & & \\ 1 & 1 & & \\ & & 1 & \mathbf{0}_{6\times3} \\ 1 & 1 & 1 & \\ \hline \multicolumn{3}{c|}{\mathbf{I}_4} & \begin{array}{c|c} \mathbf{0}_{4\times2} & -\mathbf{1}_{4\times1} \end{array} \\ \hline \multicolumn{3}{c|}{\mathbf{0}_{2\times4}} & \begin{array}{cc} 1 & -1 \\ 1 & -1 \end{array} \\ \hline & & & 1 \\ & & & 1 \\ & & & 1 \\ \multicolumn{3}{c|}{\mathbf{0}_{6\times4}} & 1 \\ & & & 1 \quad 1 \\ & & & 1 \end{array}\right], \quad \mathbf{A}^{(7)}_{21\times18} = \mathbf{I}_5 \oplus \begin{bmatrix} 1 & & & & & & & & & & & & 1 \\ 1 & & & & & & & & & & & & \\ & 1 & & & & & & & & & & & \\ & -1 & & & & 1 & & & & & & & \\ & & & & & 1 & & & & & & & \\ & & & 1 & 1 & & & & & & & & \\ & & & -1 & 1 & & & & & & & & \\ & & & & & & & 1 & 1 & & & & \\ & & & & & & & -1 & 1 & & & & \\ & & & & & & 1 & & & & & & \\ & & & & & & & & & & & & 1 \end{bmatrix} \oplus \mathbf{I}_5,$$

$$-1_{4\times1} = [-1, -1, -1, -1]^{\mathrm{T}},$$

$$\mathbf{A}_{28\times 21}^{(7)} = \mathbf{I}_7 \oplus \left[\begin{array}{ccc|ccc|ccc} 1 & & & & & & & & \\ & 1 & & & \mathbf{0}_{3\times 4} & & & \mathbf{0}_3 & \\ & & 1 & & & & & & \\ \hline 1 & & & 1 & & & & & \\ 1 & & & & 1 & & & \mathbf{0}_{4\times 3} & \\ & & 1 & & & 1 & & & \\ & & 1 & & & & 1 & & \\ \hline & & & 1 & & & & & \\ & & & 1 & & & & & \\ & \mathbf{0}_{6\times 3} & & & 1 & & & \mathbf{0}_{6\times 3} & \\ & & & & & 1 & & & \\ & & & & & 1 & & & \\ & & & & & & 1 & & \\ \hline & & & & & & 1 & & \\ & & & & & & -1 & 1 & \\ & \mathbf{0}_{4\times 3} & & & \mathbf{0}_4 & & & 1 & \\ & & & & & & & & 1 \end{array}\right] \oplus \mathbf{I}_4,$$

$$\mathbf{A}_{26\times 28}^{(7)} = \mathbf{I}_{10} \oplus \left[\begin{array}{cccccc|cccccccc} 1 & & & & & & & & & & & & & \\ -1 & & 1 & & & & & & & & & & & \\ & 1 & & & & & & & & & & & & \\ & & & 1 & & & & & & \mathbf{0}_6 & & & & \\ & -1 & & & 1 & & & & & & & & & \\ & & & & 1 & & & & & & & & & \\ \hline 1 & & & & & & 1 & 1 & 1 & & & & & \\ & & 1 & & & & & & & 1 & 1 & 1 & & \end{array}\right] \oplus \mathbf{I}_8$$

$$\mathbf{A}_{26}^{(7)} = \mathbf{I}_5 \oplus \left[\begin{array}{ccccccc|ccccccc} 1 & 1 & & & & & 1 & & & & & & & \\ & & 1 & & & & & & & & & & & \\ & & & 1 & & & & & & & & & & \\ & & & & 1 & & & & & \mathbf{0}_6 & & & & \\ & & & & & 1 & & & & & & & & \\ & & & & & & 1 & & & & & & & \\ \hline & & & & & & & 1 & & & & & & \\ & & & & & & & & 1 & & & & & \\ & & & & & & & & & 1 & & & & \\ & & \mathbf{0}_7 & & & & & & & & 1 & & & \\ & & & & & & & & & & & 1 & & \\ & & & & & & & & & & & -1 & 1 & \\ & & & & & & & & & & & & & 1 \end{array}\right] \oplus \mathbf{I}_8,$$

$$\mathbf{D}_{26}^{(7)} = diag(s_0^{(7)}, s_1^{(7)}, ..., s_{25}^{(7)}),$$

$$s_0^{(7)} = h_0, \quad s_1^{(7)} = h_2 - h_1, \quad s_2^{(7)} = h_0 - h_1, \quad s_3^{(7)} = h_1, \quad s_4^{(7)} = h_0,$$

$$s_5^{(7)} = (h_6 + h_5 + h_4 + h_3 + 2h_2 + h_1 + h_0)/7, \quad s_6^{(7)} = (-h_6 - 2h_5 + 3h_4 - h_3 - 2h_2 + h_1 + 2h_0)/2,$$

$$s_7^{(7)} = (2h_4 - h_3 - 2h_2 + h_1)/2, \quad s_8^{(7)} = (-h_6 + h_5 + 2h_4 - h_3 - 2h_2 + 3h_1 - h_0)/2,$$

$$s_9^{(7)} = (10h_6 + 3h_5 - 11h_4 + 10h_3 + 3h_2 - 11h_1 - 4h_0)/14,$$

$$s_{10}^{(7)} = (-2h_6 - 2h_5 - 2h_4 + 12h_3 + 5h_2 - 9h_1 - 2h_0)/14, \quad s_{11}^{(7)} = (2h_6 + 3h_5 - h_4 - 2h_3 + 3h_2 - h_1)/6,$$

$$s_{12}^{(7)} = (3h_6 - 11h_5 - 4h_4 + 10h_3 + 3h_2 - 11h_1 + 10h_0)/14, \quad s_{13}^{(7)} = (-2h_3 + 3h_2 - h_1)/6,$$

$$s_{14}^{(7)} = (3h_6 - h_5 - 2h_3 + 3h_2 - h_1 - 2h_0)/6, \quad s_{15}^{(7)} = (-h_6 + h_4 - h_3 + h_1)/6, \quad s_{16}^{(7)} = (-h_3 + h_1)/6,$$

$$s_{17}^{(7)} = (h_5 - h_3 + h_1 - h_0)/6, \quad s_{18}^{(7)} = 2h_6 - h_5 - 2h_4 + 3h_3 - 2h_2 - 2h_1 + h_0,$$

$$s_{19}^{(7)} = 2h_3 - h_2 - 2h_1 + h_0, \quad s_{20}^{(7)} = -h_6 - 2h_5 + h_4 + 2h_3 - h_2 - 2h_1 + 3h_0, \quad s_{21}^{(7)} = h_6,$$
$$s_{22}^{(7)} = h_4 - h_5, \quad s_{23}^{(7)} = h_6 - h_5, \quad s_{24}^{(7)} = h_5, \quad s_{25}^{(7)} = h_6\,,$$

$$\mathbf{A}_{21\times 26}^{(7)} = \mathbf{I}_6 \oplus \left[\begin{array}{cc|c} \begin{matrix} 1 & 1 & & \\ & 1 & 1 & \\ & & 1 & 1 \end{matrix} & & \mathbf{0}_{3\times 6} \\ \hline & \mathbf{0}_{2\times 9} & & \begin{matrix} 1 & 1 & \\ & 1 & 1 \end{matrix} \\ \hline \begin{matrix} -1 & 1 & \\ & -1 & 1 \end{matrix} & & \\ & & \mathbf{0}_{3\times 6} \\ \hline & \mathbf{0}_{2\times 9} & & \begin{matrix} -1 & 1 & \\ & -1 & 1 \end{matrix} \end{array}\right] \oplus \mathbf{I}_5,$$

$$\mathbf{A}_{21}^{(7)} = \mathbf{I}_7 \oplus \left[\begin{array}{cc|c} \begin{matrix} 1 & 1 & \\ & 1 & \\ & & 1 \end{matrix} & & \mathbf{0}_4 \\ \hline & \mathbf{0}_4 & & \begin{matrix} 1 & & \\ & 1 & 1 \\ & & 1 \end{matrix} \end{array}\right] \oplus \mathbf{I}_6, \quad \mathbf{A}_{25\times 21}^{(7)} = \mathbf{I}_7 \oplus \left[\begin{array}{cc|c} \begin{matrix} 1 & 1 & \\ 1 & & 1 \\ & 1 & \\ & & 1 \\ & & 1 \end{matrix} & & \mathbf{0}_{6\times 4} \\ \hline & \mathbf{0}_{6\times 4} & & \begin{matrix} 1 & & \\ & 1 & \\ & 1 & 1 \\ & & 1 \\ & & \\ & & 1 \\ & & 1 \end{matrix} \end{array}\right] \oplus \mathbf{I}_6,$$

$$\mathbf{A}_{22\times 25}^{(7)} = \mathbf{I}_6 \oplus \left[\begin{array}{cc|c} \begin{matrix} 1 & 1 & \\ & 1 & \\ & 1 & -1 \\ \hline 1 & & \\ & 1 & \end{matrix} & \begin{matrix} & \\ \mathbf{0}_3 & \\ & \\ \hline 1 & 1 & 1 \\ & & \end{matrix} & \mathbf{0}_{5\times 7} \\ \hline & \mathbf{0}_{6\times 7} & \begin{matrix} 1 & & \\ & 1 & \\ & & 1 \\ \hline & 1 & -1 \\ & 1 & \\ & & 1 \end{matrix} & \begin{matrix} & \\ \mathbf{0}_3 & \\ & \\ \hline & \\ 1 & 1 & 1 \\ & & \end{matrix} \end{array}\right] \oplus \mathbf{I}_5,$$

$$\mathbf{A}_{21\times 22}^{(7)} = \mathbf{I}_6 \oplus \left[\begin{array}{cc|c} \begin{matrix} 1 & & & & \\ -1 & -1 & & -1 & \\ & & 1 & & \\ & & & 1 & 1 \\ & & & & \\ \hline & & & & 1 \end{matrix} & & \mathbf{0}_5 \\ \hline & \mathbf{0}_{5\times 6} & & \begin{matrix} & \\ 1 & \\ & 1 \\ & & 1 & 1 \\ & & & 1 \\ & & 1 & \end{matrix} \end{array}\right] \oplus \mathbf{I}_5, \quad \mathbf{A}_{20\times 21}^{(7)} = \mathbf{I}_{10} \oplus \left[\begin{array}{ccc} 1 & & 1 \\ & 1 & \\ & & 1 \\ & & 1 \end{array}\right] \oplus \mathbf{I}_6,$$

$$\mathbf{A}^{(7)}_{15\times 20} = \begin{bmatrix} 1 & & & & & \\ & & 1 & 1 & & \\ & 1 & & 1 & 1 & \end{bmatrix} \oplus \mathbf{I}_5 \oplus \left[\begin{array}{cccc|ccc} 1 & & & & & & \\ -1 & -1 & & -1 & & \mathbf{0}_{4\times 5} & \\ & & 1 & & & & \\ & & & 1 & & & \\ \hline & & & & 1 & & 1 \ 1 \\ & \mathbf{0}_{3\times 5} & & & & 1 \ 1 & \\ & & & & 1 & & \end{array}\right],$$

$$\mathbf{A}^{(7)}_{13\times 15} = \left[\begin{array}{ccc|ccc|ccc} \multicolumn{3}{c|}{\mathbf{I}_3} & \multicolumn{3}{c|}{\mathbf{0}_{3\times 4}} & \multicolumn{2}{c|}{\mathbf{0}_{3\times 5}} & \multicolumn{2}{c}{\mathbf{0}_3} \\ \hline \multicolumn{3}{c|}{} & 1 & & & & 1 & -1 & \\ \multicolumn{3}{c|}{} & 1 & & & 1 & & & -1 \\ \multicolumn{3}{c|}{\mathbf{0}_{4\times 3}} & 1 & & & & & & \\ \multicolumn{3}{c|}{} & 1 & & 1 & 1 & & & \\ \hline -1 & & 1 & 1 & & & & & & \\ & -1 & 1 & & 1 & & & 1 & & \mathbf{0}_3 \\ & -1 & 1 & & & & & & 1 & \\ \hline \multicolumn{3}{c|}{\mathbf{0}_{3\times 4}} & \multicolumn{3}{c|}{\mathbf{0}_3} & \multicolumn{2}{c|}{\mathbf{0}_{3\times 5}} & \multicolumn{2}{c}{\mathbf{I}_3} \end{array}\right],$$

Figure 6 shows a data flow diagram of the proposed algorithm for the implementation of 7-point linear convolution.

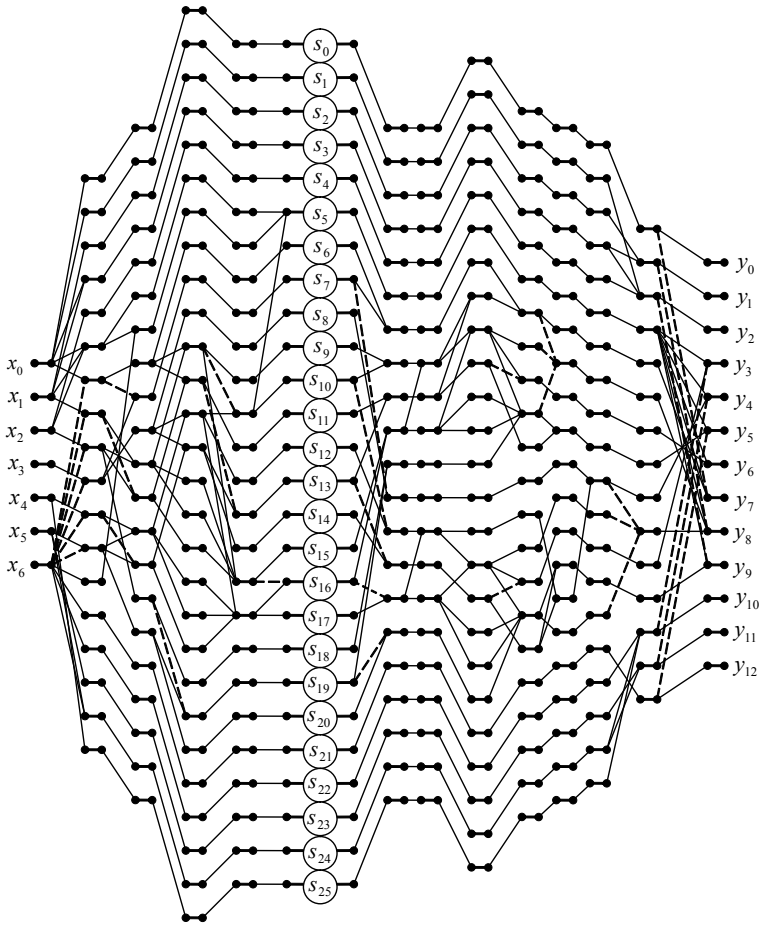

Figure 6. The signal flow graph of the proposed algorithm for computation of 7-point linear convolution.

In terms of arithmetic units, a fully parallel hardware implementation of the processor unit for calculating a 7-point linear convolution will require 27 multipliers, 49 two-input adders, 7 three-input adders, and 5 four-input adders, instead of 49 multipliers, 2 two-input adders,

2 three-input adders, 2 four-input adders, 2 five-input adders, 2 six-input adders, and 1 seven-input adder in the case of completely parallel implementation of expression (20).

3.7. Algorithm for N = 8

Let $\mathbf{X}_{8\times1} = [x_0, x_1, x_2, x_3, x_4.x_5, x_6, x_7]^{\mathrm{T}}$ and $\mathbf{H}_{8\times1} = [h_0, h_1, h_2, h_3, h_4, h_5, h_6, h_7]^T$ be 8-dimensional data vectors being convolved and $\mathbf{Y}_{15\times1} = [y_0, y_1, y_2, y_3, y_4, y_5, y_6, y_7, y_8, y_9, y_{10}, y_{11}, y_{12}, y_{13}, y_{14}]^T$ be an input vector representing a linear convolution for $N = 8$.

The problem is to calculate the product:

$$\mathbf{Y}_{15\times1} = \mathbf{H}_{15\times8}\mathbf{X}_{8\times1}, \tag{22}$$

where

$$\mathbf{H}_{15\times8} = \begin{bmatrix} h_0 & & & \\ h_1 & h_0 & & \\ \vdots & h_1 & \ddots & \\ h_7 & \vdots & \ddots & h_0 \\ & h_7 & \ddots & h_1 \\ & & \ddots & \vdots \\ & & & h_7 \end{bmatrix}.$$

Direct computation of (22) takes 64 multiplications and 49 addition. We developed an algorithm that contains only 27 multiplications and 67 additions. Thus, the proposed algorithm saves 22 multiplications at the cost of 18 extra additions compared to the ordinary matrix-vector multiplication method.

Proposed algorithm for computation 8-point linear convolution with reduced multiplicative complexity can be written using the following matrix-vector calculating procedure:

$$\mathbf{Y}_{15\times1} = \mathbf{A}_{15}^{(8)}\hat{\mathbf{A}}_{15}^{(8)}\check{\mathbf{A}}_{15}^{(8)}\mathbf{A}_{15\times17}^{(8)}\mathbf{A}_{17\times27}^{(8)}\mathbf{D}_{27}^{(8)}\mathbf{A}_{27\times17}^{(8)}\mathbf{A}_{17\times15}^{(8)}\mathbf{A}_{15\times8}^{(8)}\mathbf{X}_{8\times1} \tag{23}$$

where

$$\mathbf{A}_{15\times8}^{(8)} = \left[\begin{array}{c|c} \mathbf{I}_3 & \mathbf{0}_{3\times5} \\ \hline \mathbf{I}_4 & \mathbf{I}_4 \\ \hline \mathbf{I}_4 & -\mathbf{I}_4 \\ \hline \mathbf{0}_4 & \mathbf{I}_4 \end{array}\right], \quad \mathbf{A}_{17\times15}^{(8)} = \mathbf{I}_3 \oplus (\mathbf{H}_2 \otimes \mathbf{I}_2) \oplus \left[\begin{array}{c|c} \mathbf{I}_2 & \mathbf{0}_2 \\ \hline \mathbf{0}_2 & \mathbf{I}_2 \\ \hline \mathbf{I}_2 & \mathbf{I}_2 \end{array}\right] \oplus \mathbf{I}_4,$$

$$\mathbf{A}_{27\times17}^{(8)} = \begin{bmatrix} 1 & & & \\ 1 & & & \\ & 1 & & \\ 1 & 1 & & \\ & & & 1 \end{bmatrix} \oplus \mathbf{H}_2 \oplus \left(\mathbf{I}_4 \otimes \begin{bmatrix} 1 & \\ & 1 \\ 1 & 1 \end{bmatrix}\right) \oplus \left[\begin{array}{c|cc} & & 1 \\ 1 & 1 & 1 \\ 1 & -1 & 1 \\ 1 & & \\ \hline 1 & & \\ & & 1 \\ & & 1 \\ & 1 & \end{array}\right],$$

$$\mathbf{D}_{27}^{(8)} = diag(s_0^{(8)}, s_1^{(8)}, ..., s_{27}^{(8)}),$$

$$s_0^{(8)} = h_0, \quad s_1^{(8)} = h_2 - h_1, \quad s_2^{(8)} = h_0 - h_1, \quad s_3^{(8)} = h_1, \quad s_4^{(8)} = h_0,$$
$$s_5^{(8)} = \tfrac{1}{8}(h_0 + h_1 + h_2 + h_3 + h_4 + h_5 + h_6 + h_7), \quad s_6^{(8)} = \tfrac{1}{8}(h_0 - h_1 + h_2 - h_3 + h_4 - h_5 + h_6 - h_7),$$
$$s_7^{(8)} = \tfrac{1}{4}(-h_0 + h_1 + h_2 - h_3 - h_4 + h_5 + h_6 - h_7), \quad s_8^{(8)} = \tfrac{1}{4}(-h_0 - h_1 + h_2 + h_3 - h_4 - h_5 + h_6 + h_7),$$

$$s_9^{(8)} = \tfrac{1}{4}(h_0 - h_2 + h_4 - h_6), \quad s_{10}^{(8)} = \tfrac{1}{2}(h_0 - h_1 - h_2 + h_3 - h_4 + h_5 + h_6 - h_7),$$
$$s_{11}^{(8)} = \tfrac{1}{2}(h_0 + h_1 - h_2 + h_3 - h_4 - h_5 + h_6 - h_7), \quad s_{12}^{(8)} = \tfrac{1}{2}(-h_0 + h_2 + h_4 - h_6),$$
$$s_{13}^{(8)} = \tfrac{1}{2}(h_0 - h_1 + h_2 - h_3 - h_4 + h_5 - h_6 + h_7), \quad s_{14}^{(8)} = \tfrac{1}{2}(h_0 - h_1 + h_2 + h_3 - h_4 + h_5 - h_6 - h_7),$$
$$s_{15}^{(8)} = \tfrac{1}{2}(-h_0 - h_2 + h_4 + h_6), \quad s_{16}^{(8)} = \tfrac{1}{2}(-h_0 + h_1 + h_4 - h_5), \quad s_{17}^{(8)} = \tfrac{1}{2}(-h_0 - h_3 + h_4 + h_7),$$
$$s_{18}^{(8)} = \tfrac{1}{2}(h_0 - h_4), \quad s_{19}^{(8)} = h_4 - h_6, \quad s_{20}^{(8)} = \tfrac{1}{2}(h_5 + h_6), \quad s_{21}^{(8)} = \tfrac{1}{2}(h_6 - h_5), \quad s_{22}^{(8)} = h_5 - h_7,$$
$$s_{23}^{(8)} = h_7, \quad s_{24}^{(8)} = h_6, \quad s_{25}^{(8)} = h_7, \quad s_{26}^{(8)} = h_7\,,$$

$$\mathbf{A}_{17\times 27}^{(8)} = \begin{bmatrix} 1 & & & & & & \\ & & 1 & 1 & & & \\ & 1 & & 1 & 1 & & \end{bmatrix} \oplus \mathbf{H}_2 \oplus \left(\mathbf{I}_4 \otimes \begin{bmatrix} & 1 & 1 \\ 1 & & 1 \end{bmatrix} \right) \oplus \begin{bmatrix} 1 & 1 & 1 & & 1 \\ & 1 & -1 & -1 & \end{bmatrix} \oplus \begin{bmatrix} 1 & & 1 \\ & 1 & \end{bmatrix},$$

$$\mathbf{A}_{15\times 17}^{(8)} = \mathbf{I}_3 \oplus (\mathbf{H}_2 \otimes \mathbf{I}_2) \oplus \left[\begin{array}{c|c|c} \mathbf{0}_2 & \mathbf{I}_2 & \mathbf{I}_2 \\ \hline \mathbf{I}_2 & \mathbf{0}_2 & \mathbf{I}_2 \end{array}\right] \oplus \mathbf{I}_4, \quad \check{\mathbf{A}}_{15}^{(8)} = \mathbf{I}_3 \oplus (\mathbf{H}_2 \otimes \mathbf{I}_4) \oplus \mathbf{I}_4,$$

$$\hat{\mathbf{A}}_{15}^{(8)} = \mathbf{I}_3 \oplus \left[\begin{array}{c|c} \mathbf{0}_{5\times 3} & \mathbf{I}_5 \\ \hline \mathbf{I}_3 & \mathbf{0}_{3\times 5} \end{array}\right] \oplus \mathbf{I}_4, \quad \mathbf{A}_{15}^{(8)} = \left[\begin{array}{c|c} \mathbf{I}_7 & \mathbf{0}_{3\times 4} \oplus (-\mathbf{I}_4) \\ \hline \begin{bmatrix} -1 & & \\ & -1 & \\ & & -1 \end{bmatrix} \oplus \mathbf{I}_4 & \mathbf{I}_8 \end{array}\right].$$

Figure 7 shows a data flow diagram of the proposed algorithm for the implementation of 8-point linear convolution.

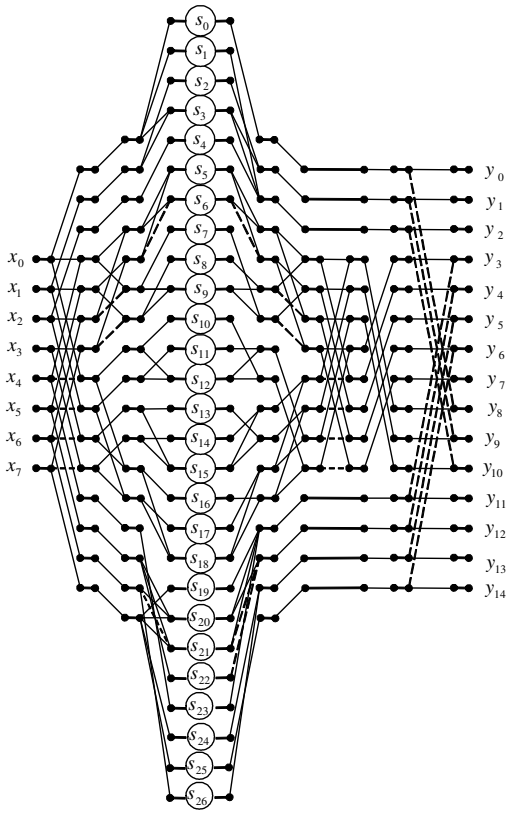

Figure 7. The signal flow graph of the proposed algorithm for computation of 8-point linear convolution.

In terms of arithmetic units, a fully parallel hardware implementation of the processor unit for calculating a 8-point linear convolution will require 27 multipliers, 57 two-input adders, 4 three-input adders and 1 four-input adder, instead of 64 multipliers, 2 two-input adders, 2 three-input adders,

2 four-input adders, 2 five-input adders, 2 six-input adders, 2 seven-point adders and 1 eight-input adder in the case of completely parallel implementation of expression (22).

4. Implementation Complexity

Since the lengths of the input sequences are relatively small, and the data flow graphs representing the organization of the computation process are fairly simple, it is easy to estimate the implementation complexity of the proposed solutions. Table 1 shows estimates of the number of arithmetic blocks for the fully parallel implementation of the short lengths linear convolution algorithms. Since a parallel N-input adder consists of N-1 two-input adders, we give integrated estimates of the implementing costs of the sets of adders for each proposed solution expressed as the sums of two-input adders. The penultimate column of the Table 1 shows the percentage reduction in the number of multipliers, while the last column shows the percentage increase in the number of adders. As you can see, the implementation of the proposed algorithms requires fewer multipliers than the implementation based on naive methods of performing the linear convolution operations.

Table 1. Implementation complexities of naïve method and proposed solutions.

Length N	Number of Arithmetical Units (Multipliers — "×" and Adders — "+")					
	Naïve Method		Proposed Solutions		Percentage Stimate	
	"×"	"+"	"×"	"+"	"×"	"+"
2	4	1	3	3	25%	66.7%
3	9	4	6	10	33.3%	60%
4	16	9	9	20	43.8%	55%
5	25	16	16	38	57.9%	66%
6	36	25	16	44	55.6%	43.2%
7	49	36	26	79	47%	54.4%
8	64	49	27	67	58%	26.9%

It should be noted that our solutions are primarily focused on efficient implementation in application specific integrated circuits (ASICs). In low-power designing low-power digital circuits, optimization must be performed both at the algorithmic level and at the logic level. From the point of view of designing an ASIC-chip that implements fast linear convolution, you should pay attention to the fact that the hardwired multiplier is a very resource-intensive arithmetic unit. The multiplier is also the most energy-intensive arithmetic unit, occupying a large crystal area [26] and dissipating a lot of energy [27]. Reducing the number of multipliers is especially important in the design of specialized fully parallel ASIC-based processors because minimizing the number of necessary multipliers reduces power dissipation and lowers the cost implementation of the entire system being implemented. It is proved that the implementation complexity of a hardwired multiplier grows quadratically with operand size, while the hardware complexity of a binary adder increases linearly with operand size [28]. Therefore, a reduction in the number of multipliers, even at the cost of a small increase in the number of adders, has a significant role in the ASIC-based implementation of the algorithm. Thus, it can be argued categorically that algorithmic solutions that require fewer hardware multipliers in an ASIC-based implementation are better than those that require more embedded multipliers.

This statement is also true for field-programmable gate array (FPGA)-based implementations. Most modern high-performance FPGAs contain a number of built-in multipliers. This means that instead of implementing the multipliers with a help of a set of conventional logic gates, you can use the hardwired multipliers embedded in the FPGA. Thus, all multiplications contained in a fully parallel algorithm can be efficiently implemented using these embedded multipliers; however, their number may not be enough to meet the requirements of a fully parallel implementation of the algorithm. So, the developer uses the embedded multipliers to implement the multiplication operations until all of the multipliers built into the chip have been used. If the embedded multipliers available in the FPGA run out, the developer will be forced to use ordinary logic gates instead. This will lead to

significant difficulties in the design and implementation of the computing unit. Therefore, the problem of reducing the number of multiplications in fully parallel hardware-oriented algorithms is critical. It is clear that you can go the other way—use a more complex FPGA chip from the same or another family, which contains a larger number of embedded multipliers; however, it should be remembered that the hardwired multiplier is a very resource-intensive unit. The multiplier is the most resource-intensive and energy-consuming arithmetic unit, occupying a large area of the chip and dissipating a lot of power; therefore, the use of complex and resource-intensive FPGAs containing a large number of multipliers without a special need is impractical.

Table 2 shows FPGA devices of the Spartan-3 family, in which the number of hardwired multipliers allows one to implement the linear convolution operation in a single chip. So, for example, a 4-point convolution implemented using our proposed algorithm can be implemented using a single Spartan XC3S200 device, while a 4-point convolution implemented using a naive method requires a more voluminous Spartan XC3S400 device. A 5-point convolution implemented using our proposed algorithm can be implemented using a single Spartan XC3S200A chip, while a 5-point convolution implemented using a naive method requires a more voluminous Spartan XC3S1500A chip, and so on.

Table 2. The possibility of implementation the naive method and proposed solution on the field-programmable gate array (FPGA) devices of the Spartan-3 family.

Length N	Features of the Implementation in Spartan-3 Family Devices	
	Naïve Method	Proposed Solutions
	Type of Device	Type of Device
2	XC3S50	XC3S50AN
3	XC3S200	XC3S200
4	XC3S400	XC3S200
5	XC3S1400AN	XC3S200AN
6	XC3S2000	XC3S200
7	XC3S4000	XC3S1400AN

Thus, the hardware implementation of our algorithms requires fewer hardware multipliers than the implementation of naive calculation methods, all other things being equal. Taking into account the previously listed arguments, this proves their effectiveness.

5. Conclusions

In this paper, we analyzed possibilities to reduce the multiplicative complexity of calculating the linear convolutions for small length input sequences. We also synthesized new algorithms for implementing these operations for N = 3, 4, 5, 6, 7, and 8. Using these algorithms reduces the computational complexity of linear convolution, thus reducing its hardware implementation complexity too. In addition, as can be seen from Figures 1–7, the proposed algorithms have a pronounced parallel modular structure. This simplifies the mapping of the algorithms into an ASIC structure and unifies its implementation in FPGAs. Thus, the acceleration of computations during the implementation of these algorithms can also be achieved due to the parallelization of the computation processes.

Author Contributions: Conceptualization, A.C.; methodology, A.C.; validation, J.P.P.; formal analysis, A.C. and 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. and J.P.P. Both authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

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

References

1. Blahut, R.E. *Fast Algorithms for Signal Processing*; Cambridge University Press: Cambridge, UK, 2010.
2. Tolimieri, R.; An, M.; Lu, C. *Algorithms for Discrete Fourier Transform and Convolution*; Springer Science & Business Media: New York, NY, USA, 1989.
3. McClellan, J.H.; Rader, C.M. *Number Theory in Digital Signal Processing*; Prentice-Hall: Englewood Cliffs, NJ, USA, 1979.
4. Berg, L.; Nussbaumer, H. Fast Fourier Transform and Convolution Algorithms. *Z. fur Angew. Math. Und Mech.* **1982**, *62*, 282–282. [CrossRef]
5. Burrus, C.S.; Parks, T. *Convolution Algorithms*; John Wiley and Sons: New York, NY, USA, 1985.
6. Krishna, H. *Digital Signal Processing Algorithms: Number Theory, Convolution, Fast Fourier Transforms, and Applications*; Routledge: New York, NY, USA, 2017.
7. Bi, G.; Zeng, Y. *Transforms and Fast Algorithms for Signal Analysis and Representations*; Springer Science & Business Media: New York, NY, USA, 2004.
8. Mallat, S.G. A theory for multiresolution signal decomposition: The wavelet representation. *IEEE Trans. Pattern Anal. Mach. Intell.* **1989**, *11*, 674–693. [CrossRef]
9. Parhi, K.K. *VLSI Digital Signal Processing Systems: Design and Implementation*; John Wiley & Sons: New York, NY, USA, 2007.
10. Chan, Y.H.; Siu, W.C. General approach for the realization of DCT/IDCT using convolutions. *Signal Process.* **1994**, *37*, 357–363. [CrossRef]
11. 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.
12. Pratt, W.K. *Digital Image Processing*; John Wiley & Sons: Hoboken, NJ, USA, 2007.
13. Wolberg, G.; Massalin, H. A fast algorithm for digital image scaling. In *Communicating with Virtual Worlds*; Springer: Tokyo, Japan, 1993; pp. 526–539.
14. Krizhevsky, A.; Sutskever, I.; Hinton, G.E. Imagenet classification with deep convolutional neural networks. *Commun. ACM* **2017**, *60*, 84–90. [CrossRef]
15. Mathieu, M.; Henaff, M.; LeCun, Y. Fast training of convolutional networks through ffts. *arXiv* **2013**, arXiv:1312.5851.
16. 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.
17. 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]
18. Lavin, A.; Gray, S. Fast algorithms for convolutional neural networks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 27–30 June 2016; pp. 4013–4021.
19. Wang, X.; Wang, C.; Zhou, X. Work-in-progress: WinoNN: Optimising FPGA-based neural network accelerators using fast winograd algorithm. In Proceedings of the 2018 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ ISSS), Turin, Italy, 30 September–5 October 2018; pp. 1–2.
20. Cariow, A.; Cariowa, G. Minimal Filtering Algorithms for Convolutional Neural Networks. *arXiv* **2020**, arXiv:2004.05607.
21. Wang, Y.; Parhi, K. Explicit Cook-Toom algorithm for linear convolution. In Proceedings of the 2000 IEEE International Conference on Acoustics, Speech, and Signal Processing, Istanbul, Turkey, 5–9 June 2000; Volume 6, pp. 3279–3282.
22. Ju, C.; Solomonik, E. Derivation and Analysis of Fast Bilinear Algorithms for Convolution. *arXiv* **2019**, arXiv:1910.13367.
23. Cariow, A. Strategies for the Synthesis of Fast Algorithms for the Computation of the Matrix-vector Products. *J. Signal Process. Theory Appl.* **2014**, *3*, 1–19. [CrossRef]
24. Regalia, P.A.; Sanjit, M.K. Kronecker products, unitary matrices and signal processing applications. *SIAM Rev.* **1989**, *31*, 586–613. [CrossRef]

25. 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]
26. Wong, H.; Betz, V.; Rose, J. Comparing FPGA vs. custom CMOS and the impact on processor microarchitecture. In Proceedings of the 19th ACM/SIGDA International Symposium on Field Programmable Gate Arrays, Monterey, CA, USA, 27 February–1 March 2011; pp. 5–14.
27. Jhamb, M.; Lohani, H. Design, implementation and performance comparison of multiplier topologies in power-delay space. *Eng. Sci. Technol. Int. J.* **2016**, *19*, 355–363. [CrossRef]
28. Oudjida, A.K.; Chaillet, N.; Berrandjia, M.L.; Liacha, A. A new high radix-2r $(r \geq 8)$ multibit recoding algorithm for large operand size $(N \geq 32)$ multipliers. *J. Low Power Electron.* **2013**, *9*, 50–62. [CrossRef]