electronics

Article

An Area-Efficient Unified VLSI Architecture for Type IV DCT/DST Having an Efficient Hardware Security with Low Overheads

Doru Florin Chiper [1,2,3] and Arcadie Cracan [1,*]

[1] Faculty of Electronics, Telecommunications and Information Technology, "Gheorghe Asachi" Technical University of Iasi, 700506 Iasi, Romania; chiper@etti.tuiasi.ro
[2] Technical Sciences Academy of Romania—ASTR, 030167 Bucharest, Romania
[3] Academy of Romanian Scientists—AOSR, 030167 Bucharest, Romania
* Correspondence: acracan@etti.tuiasi.ro

Abstract: This paper introduces an efficient solution for designing a unified VLSI implementation for type IV DCT/DST while solving one challenging problem in obtaining high performance VLSI chips for common goods, which is solving the security of the hardware while obtaining a VLSI implementation with high performance. The new solution uses a new systolic array algorithm for type IV DST that can allow us to obtain an efficient unified VLSI architecture with one previously designed for type IV DCT. The proposed method uses special arithmetic structures that have been called quasi-cycle convolutions that can be efficiently mapped on linear systolic arrays. Moreover, the obtained unified VLSI architecture, besides being an efficient implementation with a low hardware complexity and high-speed performance, allows for an efficient inclusion of the obfuscation technique with very low overheads.

Keywords: DCT IV transform; DST IV transform; discrete transforms; hardware security; systolic arrays; time-varying obfuscation; VLSI algorithm

check for updates

Citation: Chiper, D.F.; Cracan, A. An Area-Efficient Unified VLSI Architecture for Type IV DCT/DST Having an Efficient Hardware Security with Low Overheads. *Electronics* **2023**, *12*, 4471. https://doi.org/10.3390/electronics12214471

Academic Editor: Antonio G. M. Strollo

Received: 2 August 2023
Revised: 18 October 2023
Accepted: 25 October 2023
Published: 30 October 2023

Copyright: © 2023 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

In the past years, there have been some fields, for example, telemedicine, of growing interest, and these fields involve an efficient transmission of data at distance. For such a kind of medicine, an important aspect is data compression using the discrete cosine transform, type IV (DCT-IV), or the discrete sine transform, type IV (DST-IV).

The type IV discrete cosine and sine transforms introduced by Jain [1] have some important applications, such as spectral analyses, signal and image coding, implementation of orthogonal overlapping transforms, internet audio/video streaming, filter banks, etc. [2–5], and can be used as good candidates in data compression. Both of these two transforms are computationally intensive and, in real-time applications, are of great importance in finding efficient hardware implementations.

To obtain good VLSI implementation, it is necessary to cleverly reformulate the basic expression of these algorithms or to define new ones. To obtain this, we have taken into consideration that, for an optimal implementation of these DSP algorithms, it is vital to investigate the flow of the data within the algorithm structure and to consider computational structures with a special form [6–11], such as cyclic convolution, circular correlation, quasi-cycle convolutions, quasi-circular correlations, band convolution, or band correlation and, at the same time, reduce the overall arithmetic complexity. All of these computational structures can be used to design optimal implementations using systolic arrays [12] or distributed arithmetic [13].

In our days, due to the globalization of the design of integrated circuits, many companies must use IP cores from many international companies in order to optimize the cost.

Due to this, IC piracy, overbuilding, and reverse engineering represent major challenges for electronics engineering [14,15]. Thus, companies that produce pirated IC chips can produce more chips at a lower cost. Also, untrusted companies can obtain valuable IP information by reverse engineering and can illegally use it in their chips. Thus, it is important to integrate hardware security techniques in the new designs, but this can introduce large overheads, which, in the case of common goods, represent a real problem.

There are multiple hardware security augmenting techniques, but in this paper we have chosen obfuscation [16–19] due to its simplicity and efficiency. We have used a mode-based obfuscation based on the control flow, in which we can use the properties of the control flow to simply generate functional modes that lead to an incorrect operation to obfuscate the correct one. Thus, only for the correct key are the suitable control signals applied, and, for the other combinations, incorrect control signals are applied.

There are very many good VLSI implementations for type II and type III DCT, and quite a few good hardware implementations for type IV DCT or type IV DST [20–29]. Among them, there are only a few unified solutions that can efficiently execute both transforms using a large portion of the area in common on the same chip [20–22,28].

Related Works

In this section, we present and summarize the most relevant VLSI implementations from the last 10 years.

In [24], the authors present a VLSI architecture for DCT IV where two systolic arrays operate in parallel, as opposed to our solution, wherein we have three systolic arrays working in parallel with a higher speed performance.

The above solutions have not been designed to incorporate hardware security techniques.

In [26], is the authors presented a VLSI architecture for type IV DST with the same length, $N = 13$, whereas we have eight linear systolic arrays with three PEs for each one, and we are using general multipliers. The number of multipliers is $2(N-1)$.

In [28], is the authors presented a unified VLSI architecture for type IV DCT/DST that is the best reported in the literature, whereas we have eight computational structures implemented using eight short systolic arrays and where $2(N-1)$ general multipliers and $2(N-1)$ adders have been used. Because it uses general multipliers with a higher hardware complexity as compared to multipliers with a constant, it has a significantly higher hardware complexity.

The reference [29] is about a VLSI implementation of DCT IV. Since the VLSI algorithm for type IV DCT has been used with our proposed VLSI algorithm for DST IV for unification, it has similar performance as our unified VLSI architecture, while our solution computes both transforms using the same chip with a similar hardware complexity as that used for DCT IV presented in [29].

The architectures described in [26,28,29] have been developed to facilitate the incorporation of hardware security techniques.

In this paper, we are using the obfuscation technique presented in [15] and a new VLSI algorithm for type IV DST that allows us to obtain a unified VLSI architecture having a significantly lower hardware complexity than existing architecture using only six regular and modular computation structures, called quasi-cycle convolutions, that can be computed in parallel. Also, it can allow an optimal incorporation of the hardware security with very low overheads.

Some important contributions of the paper are:

- A new VLSI algorithm for type IV DST using only six special computation structures that allow an efficient VLSI implementation, called quasi-cycle convolutions, as compared with that in [28], where eight such structures are used.
- The new VLSI algorithm for type IV DST can be used to obtain a significantly reduced hardware complexity as compared to existing ones by using multiplications where one operand is a constant instead of the usual case where the hardware complexity of the multiplier is significant higher.

- A new unified VLSI algorithm for type IV DCT and DST has been obtained that leads to an efficient unified VLSI architecture, in which most of the chip area is used in common by the two transforms.
- The obtained unified VLSI architecture allows the inclusion of hardware security with very low overheads.

The rest of the paper is organized as follows: In Section 2, we present the new VLSI algorithm for type IV DST together with a unified version based on a previously designed algorithm for type IV DCT that allows for the designing of a unified VLSI architecture for type IV DCT/DST in an optimal way. In Section 3, we present the obtained unified VLSI architecture for type IV DCT/DST, which can execute both transforms on the same VLSI architecture with very few changes, and, at the same time, it allows for the incorporation of hardware security in an optimal way. In Section 4, we present the results, and in Section 5, we discuss the obtained solution. Finally, in Section 6, we draw the conclusions.

2. Methods

2.1. A New VLSI Algorithm for DST IV

For a real input sequence $x(i) : i = 0, 1, \ldots, N-1$, type IV DST (DST-IV) is defined as below:

$$Y(k) = \sqrt{2/N} \cdot \sum_{i=0}^{N-1} x(i) \cdot \sin[(2i+1)(2k+1)\alpha/2] \tag{1}$$

where $k = 0, 1, \ldots, N-1$, and where

$$\alpha = \frac{\pi}{2N} \tag{2}$$

To efficiently reformulate (1) with the goal of obtaining a new VLSI algorithm with an efficient implementation, we have introduced several auxiliary input and output sequences, and the obtained sequences have been permuted appropriately based on the properties of the Galois Field. We have obtained a parallel form of the algorithm in which some computation structures with a particular form have been used.

The output sequence $\{Y(k) : k = 1, 2, \ldots, N-1\}$ can be computed using the following equation, as detailed in Appendix A:

$$Y(k) = x_a(0) \cdot \sin[(2k+1)\alpha/2] + 2Y_a^S(k) \cdot \cos[(2k+1)\alpha/2] \tag{3}$$

for $k = 1, \ldots, N-1$, where we have introduced an auxiliary output sequence $\{Y_a(k) : k = 1, 2, \ldots, N-1\}$ that can be computed recursively as follows:

$$Y_a^S(0) = \sum_{i=0}^{N-1} \left(-1\right)^i x_a(i) \sin i\alpha \tag{4}$$

$$Y_a^S(k) = T^S(k) - Y_a^S(k-1) \tag{5}$$

where $T^S(k)$ is computed using Equations (26) and (34) and in which we have introduced the auxiliary input sequence $\{x_a(i) : i = 0, \ldots, N-1\}$, which is recursively computed as follows:

$$x_a(N-1) = x(N-1) \tag{6}$$

$$x_a(i) = (-1)^i x(i) + x_a(i+1) \tag{7}$$

for $i = N-2, \ldots, 0$.

The new auxiliary output sequence $\{T^S(k) : k = 1, 2, \ldots, N-1\}$ can be computed using 6 short computational structures, called quasi-cycle convolutions, that can be implemented using 6 linear systolic arrays having $M/2$ processing elements (PEs) if the transform length N is a prime number where $N = 2M + 1$. In the following, we have considered the prime length $N = 13$. Thus, in the following, we are using 6 such short computational structures that can be implemented using 6 systolic arrays with 3 PEs.

We are introducing the following auxiliary input sequence:

$$x^C(i+j) = x^C(i) + x^C(j) \tag{8}$$

with

$$x^C(i) = x_a(i) \cdot \cos i\alpha \tag{9}$$

We have the following matrix–vector product that computes a partial result, which is used in the computation of the transform outputs, as will be presented later:

$$T_{1a}^S = \begin{bmatrix} x^C(4+9) & -x^C(3+10) & -x^C(1+12) \\ -x^C(1+12) & x^C(4+9) & x^C(3+10) \\ x^C(3+10) & -x^C(1+12) & -x^C(4+9) \end{bmatrix} \cdot \begin{bmatrix} s_a(1) \\ s_a(2) \\ s_a(3) \end{bmatrix} \tag{10}$$

with:

$$s_a(1) = s(4) - s(5) \tag{11}$$

$$s_a(2) = s(3) + s(6) \tag{12}$$

$$s_a(3) = s(1) + s(2) \tag{13}$$

in which:

$$s(i) = 2 \cdot \sin 4i\alpha \tag{14}$$

We also have:

$$T_{1b}^S = \begin{bmatrix} x_q^C(2,4) & x_q^C(3,5) & -x_q^C(1,6) \\ -x_q^C(1,6) & x_q^C(2,4) & -x_q^C(3,5) \\ -x_q^C(3,5) & -x_q^C(1,6) & -x_q^C(2,4) \end{bmatrix} \cdot \begin{bmatrix} s_b(1) \\ s_b(2) \\ s_b(3) \end{bmatrix} \tag{15}$$

with:

$$x_q^C(2,4) = x^C(2+11) - x^C(4+9) \tag{16}$$

$$x_q^C(3,5) = x^C(3+10) + x^C(5+8) \tag{17}$$

$$x_q^C(1,6) = -\left[x^C(1+12) - x^C(6+7)\right] \tag{18}$$

and

$$s_b(1) = s(4) \tag{19}$$

$$s_b(2) = s(3) \tag{20}$$

$$s_b(3) = s(1) \tag{21}$$

The third computational structure is given by:

$$T_{1c}^S = \begin{bmatrix} x^C(2+11) & -x^C(5+8) & -x^C(6+7) \\ -x^C(6+7) & -x^C(2+11) & -x^C(5+8) \\ -x^C(5+8) & x^C(6+7) & -x^C(2+11) \end{bmatrix} \cdot \begin{bmatrix} s_c(1) \\ s_c(2) \\ s_c(3) \end{bmatrix} \tag{22}$$

with:

$$s_c(1) = s(3) - s(5) \tag{23}$$

$$s_c(2) = s(1) - s(6) \tag{24}$$

$$s_c(3) = s(2) + s(4) \tag{25}$$

Finally, we can compute the even part of the auxiliary output sequence $T^S(k)$ combining the results of the quasi-cycle convolutions from above as follows:

$$\begin{bmatrix} T^S(4) \\ T^S(8) \\ T^S(10) \\ T^S(6) \\ T^S(12) \\ T^S(2) \end{bmatrix} = \begin{bmatrix} T^S_{1a}(1) + T^S_{1b}(1) \\ T^S_{1c}(1) - T^S_{1b}(2) \\ -T^S_{1a}(2) - T^S_{1b}(2) \\ T^S_{1c}(2) - T^S_{1b}(3) \\ T^S_{1a}(3) + T^S_{1b}(3) \\ -T^S_{1c}(3) - T^S_{1b}(1) \end{bmatrix} \tag{26}$$

Then, we obtain the 4th one with:

$$T^S_{2a} = \begin{bmatrix} x^C(4-9) & x^C(3-10) & x^C(1-12) \\ x^C(1-12) & x^C(4-9) & -x^C(3-10) \\ -x^C(3-10) & x^C(1-12) & -x^C(4-9) \end{bmatrix} \cdot \begin{bmatrix} s_a(1) \\ s_a(2) \\ s_a(3) \end{bmatrix} \tag{27}$$

in which

$$x^C(i-j) = x^C(i) - x^C(j) \tag{28}$$

and the 5th quasi-cycle convolution as follows:

$$T^S_{2b} = \begin{bmatrix} x^C_r(2,4) & -x^C_r(3,5) & -x^C_r(1,6) \\ -x^C_r(1,6) & x^C_r(2,4) & x^C_r(3,5) \\ x^C_r(3,5) & -x^C_r(1,6) & -x^C_r(2,4) \end{bmatrix} \cdot \begin{bmatrix} s_b(1) \\ s_b(2) \\ s_b(3) \end{bmatrix} \tag{29}$$

with:

$$x^C_r(2,4) = x^C(2-11) - x^C(4-9) \tag{30}$$

$$x^C_r(3,5) = x^C(3-10) + x^C(5-8) \tag{31}$$

$$x^C_r(1,6) = x^C(1-12) + x^C(6-7) \tag{32}$$

The 6th quasi-cycle convolution is computed as follows:

$$T^S_{2c} = \begin{bmatrix} x^C(2-11) & x^C(5-8) & -x^C(6-7) \\ -x^C(6-7) & -x^C(2-11) & x^C(5-8) \\ x^C(5-8) & x^C(6-7) & -x^C(2-11) \end{bmatrix} \cdot \begin{bmatrix} s_c(1) \\ s_c(2) \\ s_c(3) \end{bmatrix} \tag{33}$$

And finally, we obtain the odd part of the sequence $T^S(k)$ by combining the results of the above computational structures as follows:

$$\begin{bmatrix} T^S(9) \\ T^S(5) \\ T^S(3) \\ T^S(7) \\ T^S(1) \\ T^S(11) \end{bmatrix} = \begin{bmatrix} -T^S_{2a}(1) - T^S_{2b}(1) \\ -T^S_{2c}(1) + T^S_{2b}(2) \\ T^S_{2a}(2) + T^S_{2b}(2) \\ -T^S_{2c}(2) + T^S_{2b}(3) \\ -T^S_{2a}(3) - T^S_{2b}(3) \\ T^S_{2c}(3) + T^S_{2b}(1) \end{bmatrix} \tag{34}$$

Using the auxiliary input sequences and the recurrence given by (6) and (7) and the two auxiliary output sequences, and then reordering the resulted computations using the properties of the Galois Field, we can compute in parallel the DST IV transform using 6 short quasi-cyclic convolutions structures instead of 8 such computational structures as in [28].

2.2. A Unified VLSI Algorithm for DCT/DST IV

Using the above VLSI algorithm for DST IV and a previous VLSI algorithm for DCT IV [29], we can obtain a unified VLSI algorithm for DCT/DST IV presented below.

The output sequence $\{Y(k) : k = 1, 2, \ldots, N-1\}$ can be computed as follows:

$$Y(k) = x_a(0) \cdot c_0^U + 2Y_a^U(k) \cdot \cos[(2k+1)\alpha/2] \tag{35}$$

for $k = 1, \ldots, N-1$, where the auxiliary input sequence $\{x_a(i) : i = 0, ..., N-1\}$ is computed as in Equations (6) and (7) and where:

$$c_0^U = \begin{cases} \sin[(2k+1)\alpha/2] & \text{for DST IV} \\ \cos[(2k+1)\alpha/2] & \text{for DCT IV} \end{cases} \tag{36}$$

and where we have used an auxiliary output sequence $\left\{Y_a^U(k) : k = 1, 2, \ldots, N-1\right\}$, which can be computed recursively as follows:

$$Y_a^U(0) = \sum_{i=0}^{N-1} \left(-1\right)^i x_a(i) \cdot c^U(i) \tag{37}$$

$$Y_a^U(k) = T^U(k) - Y_a^U(k-1) \tag{38}$$

where $T^U(k)$ is computed using Equations (48) and (56) and with the following notation for the multiplying coefficient:

$$c^U(i) = \begin{cases} \sin i\alpha & \text{for DST IV} \\ \cos i\alpha & \text{for DCT IV} \end{cases} \tag{39}$$

The new auxiliary output sequence $\left\{T^U(k) : k = 1, 2, \ldots, N-1\right\}$ can be computed in parallel using 6 computational structures for the case in which the length N is a prime number.

We introduce the following auxiliary input sequence:

$$x^U(i+j) = x^U(i) + x^U(j) \tag{40}$$

with

$$x^U(i) = \begin{cases} x_a(i) \cdot \cos i\alpha & \text{for DST IV} \\ x_a(i) \cdot \sin i\alpha & \text{for DCT IV} \end{cases} \tag{41}$$

Thus, we have:

$$T_{1a}^U = \begin{bmatrix} x^U(4+9) & -x^U(3+10) & -x^U(1+12) \\ -x^U(1+12) & x^U(4+9) & x^U(3+10) \\ x^U(3+10) & -x^U(1+12) & -x^U(4+9) \end{bmatrix} \cdot \begin{bmatrix} s_a(1) \\ s_a(2) \\ s_a(3) \end{bmatrix} \tag{42}$$

where $s_{1a}(i)$ have the same expressions as in Equations (11)–(13).

We also have:

$$T_{1b}^U = \begin{bmatrix} x_q^U(2,4) & x_q^U(3,5) & -x_q^U(1,6) \\ -x_q^U(1,6) & x_q^U(2,4) & -x_q^U(3,5) \\ -x_q^U(3,5) & -x_q^U(1,6) & -x_q^U(2,4) \end{bmatrix} \cdot \begin{bmatrix} s_b(1) \\ s_b(2) \\ s_b(3) \end{bmatrix} \tag{43}$$

with:

$$x_q^U(2,4) = x^U(2+11) - x^U(4+9) \tag{44}$$

$$x_q^U(3,5) = x^U(3+10) + x^U(5+8) \tag{45}$$

$$x_q^U(1,6) = -\left[x^U(1+12) - x^U(6+7)\right] \tag{46}$$

and $s_{1b}(i)$ as defined in Equations (19)–(21).

The third quasi-cycle convolution is given by:

$$T_{1c}^{U} = \begin{bmatrix} x^{U}(2+11) & -x^{U}(5+8) & -x^{U}(6+7) \\ -x^{U}(6+7) & -x^{U}(2+11) & -x^{U}(5+8) \\ -x^{U}(5+8) & x^{U}(6+7) & -x^{U}(2+11) \end{bmatrix} \cdot \begin{bmatrix} s_c(1) \\ s_c(2) \\ s_c(3) \end{bmatrix} \tag{47}$$

with $s_{1c}(i)$ as defined in Equations (23)–(25).

Finally, we obtain the even part of the sequence $T^{U}(k)$ using the outputs of the computational structures from above as follows:

$$\begin{bmatrix} T^{U}(4) \\ T^{U}(8) \\ T^{U}(10) \\ T^{U}(6) \\ T^{U}(12) \\ T^{U}(2) \end{bmatrix} = \begin{bmatrix} T_{1a}^{U}(1) + T_{1b}^{U}(1) \\ T_{1c}^{U}(1) - T_{1b}^{U}(2) \\ -T_{1a}^{U}(2) - T_{1b}^{U}(2) \\ T_{1c}^{U}(2) - T_{1b}^{U}(3) \\ T_{1a}^{U}(3) + T_{1b}^{U}(3) \\ -T_{1c}^{U}(3) - T_{1b}^{U}(1) \end{bmatrix} \tag{48}$$

Then, we are computing the 4th quasi-cycle convolution as:

$$T_{2a}^{U} = \begin{bmatrix} x^{U}(4-9) & x^{U}(3-10) & x^{U}(1-12) \\ x^{U}(1-12) & x^{U}(4-9) & -x^{U}(3-10) \\ -x^{U}(3-10) & x^{U}(1-12) & -x^{U}(4-9) \end{bmatrix} \cdot \begin{bmatrix} s_a(1) \\ s_a(2) \\ s_a(3) \end{bmatrix} \tag{49}$$

with:

$$x^{U}(i-j) = x^{U}(i) - x^{U}(j) \tag{50}$$

and the 5th quasi-cycle convolution as follows:

$$T_{2b}^{U} = \begin{bmatrix} x_r^{U}(2,4) & -x_r^{U}(3,5) & -x_r^{U}(1,6) \\ -x_r^{U}(1,6) & x_r^{U}(2,4) & x_r^{U}(3,5) \\ x_r^{U}(3,5) & -x_r^{U}(1,6) & -x_r^{U}(2,4) \end{bmatrix} \cdot \begin{bmatrix} s_b(1) \\ s_b(2) \\ s_b(3) \end{bmatrix} \tag{51}$$

with:

$$x_r^{U}(2,4) = x^{U}(2-11) - x^{U}(4-9) \tag{52}$$

$$x_r^{U}(3,5) = x^{U}(3-10) + x^{U}(5-8) \tag{53}$$

$$x_r^{U}(1,6) = x^{U}(1-12) + x^{U}(6-7) \tag{54}$$

The 6th quasi-cycle convolution is computed as follows:

$$T_{2c}^{U} = \begin{bmatrix} x^{U}(2-11) & x^{U}(5-8) & -x^{U}(6-7) \\ -x^{U}(6-7) & -x^{U}(2-11) & x^{U}(5-8) \\ x^{U}(5-8) & x^{U}(6-7) & -x^{U}(2-11) \end{bmatrix} \cdot \begin{bmatrix} s_c(1) \\ s_c(2) \\ s_c(3) \end{bmatrix} \tag{55}$$

And finally, we obtain the odd part of the sequence $T^{U}(k)$ by combining the outputs of the above computational structures as follows:

$$\begin{bmatrix} T^{U}(9) \\ T^{U}(5) \\ T^{U}(3) \\ T^{U}(7) \\ T^{U}(1) \\ T^{U}(11) \end{bmatrix} = \begin{bmatrix} -T_{2a}^{U}(1) - T_{2b}^{U}(1) \\ -T_{2c}^{U}(1) + T_{2b}^{U}(2) \\ T_{2a}^{U}(2) + T_{2b}^{U}(2) \\ -T_{2c}^{U}(2) + T_{2b}^{U}(3) \\ -T_{2a}^{U}(3) - T_{2b}^{U}(3) \\ T_{2c}^{U}(3) + T_{2b}^{U}(1) \end{bmatrix} \tag{56}$$

A further optimization of the computations performed in Equations (48) and (56) can be performed by rearranging the order of the $T_{1c}^{U}(i)$ and $T_{2c}^{U}(i)$ such that the expressions $\pm T_{1c}^{U}(i) \pm T_{1b}^{U}(j)$ from Equation (48) and the expressions $\pm T_{2c}^{U}(i) \pm T_{2b}^{U}(j)$ can be

computed in order. This can be achieved by considering the permutation $\pi(l)$ described in Equation (57):

$$\pi = \begin{pmatrix} 1 & 2 & 3 \\ 3 & 1 & 2 \end{pmatrix} \tag{57}$$

and by denoting $P_{1c}^{U}(i) = T_{1c}^{U} \circ \pi(i)$ and $P_{2c}^{U}(i) = T_{2c}^{U} \circ \pi(i)$. Equations (48) and (56) can be re-written as:

$$\begin{bmatrix} T^U(4) \\ T^U(8) \\ T^U(10) \\ T^U(6) \\ T^U(12) \\ T^U(2) \end{bmatrix} = \begin{bmatrix} T_{1a}^U(1) + T_{1b}^U(1) \\ P_{1c}^U(2) - T_{1b}^U(2) \\ -T_{1a}^U(2) - T_{1b}^U(2) \\ P_{1c}^U(3) - T_{1b}^U(3) \\ T_{1a}^U(3) + T_{1b}^U(3) \\ -P_{1c}^U(1) - T_{1b}^U(1) \end{bmatrix} \tag{58}$$

$$\begin{bmatrix} T^U(9) \\ T^U(5) \\ T^U(3) \\ T^U(7) \\ T^U(1) \\ T^U(11) \end{bmatrix} = \begin{bmatrix} -T_{2a}^U(1) - T_{2b}^U(1) \\ -P_{2c}^U(2) + T_{2b}^U(2) \\ T_{2a}^U(2) + T_{2b}^U(2) \\ -P_{2c}^U(3) + T_{2b}^U(3) \\ -T_{2a}^U(3) - T_{2b}^U(3) \\ P_{2c}^U(1) + T_{2b}^U(1) \end{bmatrix} \tag{59}$$

Since $\pi(l)$ is a circular permutation, $P_{1c}^{U}(i)$ and $P_{2c}^{U}(i)$ can be obtained from Equations (47) and (55) by circularly permuting the lines of the matrices, as shown in Equations (60) and (61). It is important to observe that these permutations do not alter the property that all the elements along parallels to the main diagonal of the matrix are equal in absolute value, a property specific to quasi-cycle convolutions.

$$P_{1c}^{U} = \begin{bmatrix} -x^U(5+8) & x^U(6+7) & -x^U(2+11) \\ x^U(2+11) & -x^U(5+8) & -x^U(6+7) \\ -x^U(6+7) & -x^U(2+11) & -x^U(5+8) \end{bmatrix} \cdot \begin{bmatrix} s_c(1) \\ s_c(2) \\ s_c(3) \end{bmatrix} \tag{60}$$

$$P_{2c}^{U} = \begin{bmatrix} x^U(5-8) & x^U(6-7) & -x^U(2-11) \\ x^U(2-11) & x^U(5-8) & -x^U(6-7) \\ -x^U(6-7) & -x^U(2-11) & x^U(5-8) \end{bmatrix} \cdot \begin{bmatrix} s_c(1) \\ s_c(2) \\ s_c(3) \end{bmatrix} \tag{61}$$

3. The Proposed Unified VLSI Architecture for DCT/DST IV

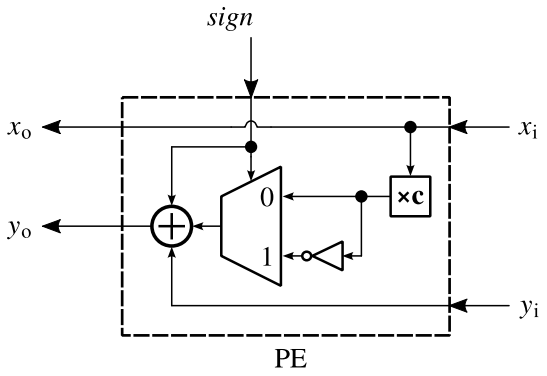
3.1. Designing the VLSI Architecture

The proposed architecture is derived from a typical systolic array that implements a quasi-cycle convolution, as in [28]. Figure 1 describes the interface and the operation of a processing element (PE) from a systolic array that is used to implement a quasi-cycle convolution. The processing element has a multiplier with a constant at its core and an adder/subtractor that implements the addition/subtraction based on the sign input. The multiplier with a constant is represented as a "×c" block and drives the "0" input of the multiplexer. When the sign input is high, the multiplexer selects the inverted output of the multiplier, and an input carry bit is applied to the adder to obtain the two's complement of the multiplier output.

Since the matrix–vector products in Equations (42) and (49), (43) and (51), (60) and (61) have, each two at a time, identical coefficient vectors, we can take advantage of the fact that the processing elements will have the same multipliers with a constant to reduce the area of the VLSI implementation.

Instead of using two different systolic arrays to compute the T_{1a} and T_{2a} partial results samples, one can re-use the same systolic array to compute sequentially the samples of T_{1a} and the samples of T_{2a} (and the same considerations apply for T_{1b} and T_{2b} and P_{1c} and P_{2c}, respectively). To obtain the partial results samples of T_{2a} at almost the same time as the samples of T_{1a}, we propose applying the input samples for the computation of T_{2a} interleaved with the input samples for the computation of T_{1a}, as illustrated in Figure 2, at

the right side of the systolic array, the x_i input of PE_1. To achieve the interleaved operation, compared to a typical quasi-cycle convolution implementation with systolic arrays, we had to double the number of delay elements at the output of each PE (except for the last PE along the chain), as represented by the thick vertical bars in Figure 2. The same considerations apply for the systolic arrays that implement Equations (43) and (51), (60), and (61), as shown in Figures 3 and 4.



Processing element operation:

$$x_o = x_i$$
$$y_o = \begin{cases} y_i + c \cdot x_i, & sign = 0 \\ y_i - c \cdot x_i, & sign = 1 \end{cases}$$

Figure 1. The function of a processing element (PE) used in the systolic array.

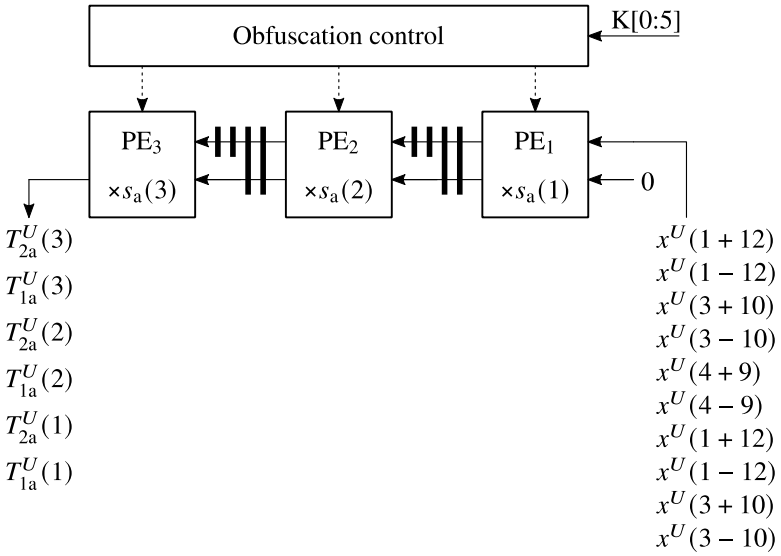


Figure 2. Systolic array with interleaved operation for Equations (42) and (49). The vertical bars along the data flow arrows represent delay elements. The input samples of Equations (42) and (49) are applied in an interleaved manner, and the output samples of Equations (42) and (49) are obtained in an interleaved manner. $K[0:5]$ is part of the de-obfuscation key that controls the selection of the applied sign bits to the processing elements.

The computation of the first output sample begins after the first four input samples have been loaded in the holding registers at the output of PE_1 and $x^U(4+9)$ reaches the input of the first PE. Due to the unequal number of delay elements along the input samples path ($x_i \rightarrow x_o$) and partial results path ($y_i \rightarrow y_o$), the partial results "travel" at twice the speed along the systolic array. By the time the first partial result $x^U(4+9)\cdot s_a(1)$ reaches PE_2 after two clock cycles, it "catches up" with the previously applied $x^U(3+10)$ input sample and the partial result accumulates the term $-x^U(3+10)\cdot s_a(2)$. After two more clock cycles, the partial result accumulates the final term of the $T^U_{1a}(1)$ computation, $-x^U(1+12)\cdot s_a(3)$. It can be observed that the computation of the partial results of the T^U_{1a} samples starts at even numbers of clock cycles. On the other hand, the computation of $T^U_{2a}(1)$ starts when $x^U(4-9)$ reaches the input of the first processing element after five clock

cycles and continues by accumulating partial results every other clock cycle. Therefore, the computation of the partial results of the T_{2a}^{U} samples starts at odd numbers of clock cycles. Similar considerations apply to the operation of the systolic arrays in Figures 3 and 4.

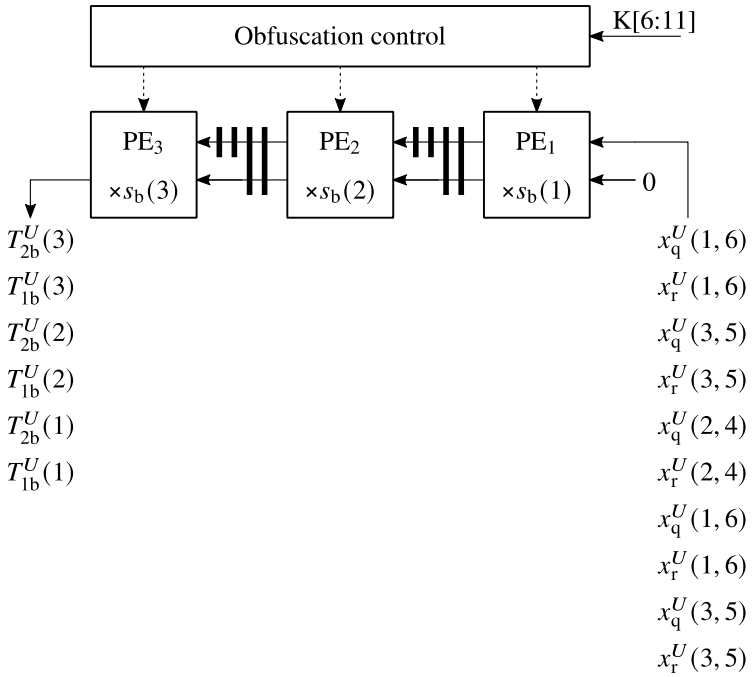


Figure 3. Systolic array with interleaved operation for Equations (43) and (51). $K[6:11]$ is part of the de-obfuscation key that controls the selection of the applied sign bits to the processing elements.

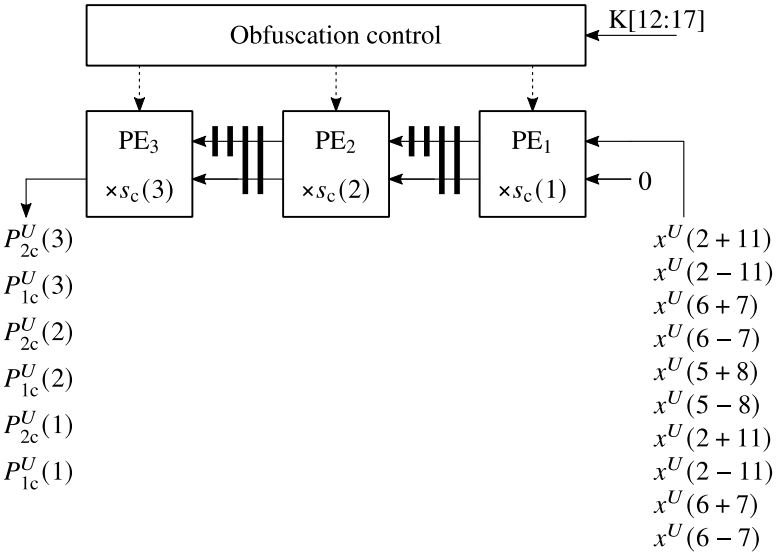


Figure 4. Systolic array with interleaved operation for Equations (60) and (61), resulting from Equations (47) and (55) by conveniently re-arranging the input samples. $K[12:17]$ is part of the de-obfuscation key that controls the selection of the applied sign bits to the processing elements.

3.2. The Obfuscation Technique Used in the Proposed Design

The obfuscation technique employed in this work is a mode-based obfuscation technique [17,18]. A correct key, representing an 18-bit binary code, must be applied at the input of the obfuscation control blocks in Figures 2–4 for the unified DCT/DST core to produce

the correct results. To obfuscate the operation of the systolic arrays, we have chosen three random permutations, π_1, π_2, and π_3 of the $\{1, 2, \ldots, 9\}$ set:

$$\pi_1 = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 4 & 6 & 7 & 5 & 2 & 8 & 3 & 9 & 1 \end{pmatrix} \tag{62}$$

$$\pi_2 = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 8 & 1 & 6 & 3 & 7 & 4 & 9 & 5 & 2 \end{pmatrix} \tag{63}$$

$$\pi_3 = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 5 & 3 & 6 & 8 & 7 & 1 & 9 & 2 & 4 \end{pmatrix} \tag{64}$$

A four-way multiplexer is used to generate the obfuscated sign bit, $\hat{sgn}(i)$, for a certain PE. Each multiplexer's selection bits are driven by two bits of the de-obfuscation key, $K[2i+1:2i]$, where i is the number of the multiplexer, with $i \in 0, 8$, corresponding to one PE of the 3 systolic arrays. Figure 5 represents the four possible instances of a multiplexer based on the chosen combination of the key bits. For this work, we have considered a de-obfuscation key $K =$ 0x27E91 (base 16 representation). In the case the wrong key is applied at the input of the obfuscation control block, a different sign input (one of the other 8 possible sign bits, based on the particular permutation corresponding to the selected MUX input) will be applied to the processing element.

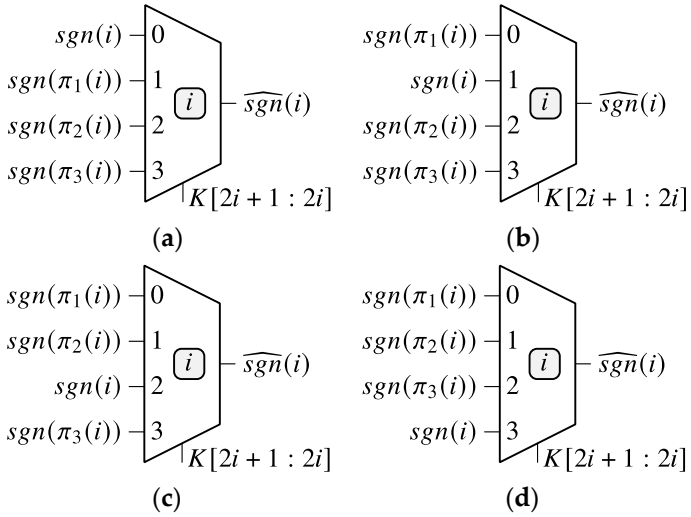


Figure 5. Four possible connections of the four-way multiplexer inputs depending on the obfuscation key values: (**a**) $K[2i+1:2i] = 0$, for which the correct sign bit is applied at input 0; (**b**) $K[2i+1:2i] = 1$, for which the correct sign bit is applied at input 1; (**c**) $K[2i+1:2i] = 2$, for which the correct sign bit is applied at input 2; and (**d**) $K[2i+1:2i] = 3$, for which the correct sign bit is applied at input 3.

The suggested obfuscation method is straightforward and comes with minimal overhead, which is especially important for mass production consumer goods. Moreover, it is difficult to distinguish the correct sign bits from the wrong ones because all the sign signals are selected from the same pool of signals (which are the sign signals corresponding to all processing elements). This amplifies the level of confusion and, consequently, the effectiveness of obfuscation, which can be highly advantageous, particularly in scenarios where reverse engineering is employed.

4. Results

As it can be seen from Section 2.1, we have designed a new VLSI algorithm for type IV DST that can be unified in a straightforward manner with a previous algorithm proposed by us in [29]. The obtained algorithm allows the decomposition of the computation of type IV DST using six computational structures that can operate in parallel. Compared with the best unified VLSI implementation of DCT/DST IV, where eight such computational

structures are used, we have obtained a considerable reduction of the hardware complexity. Since the general multipliers have been replaced with multipliers where one operand is constant, a further reduction of the hardware complexity has been achieved.

As compared with the algorithm for DCT IV proposed in [29], our algorithm for DST IV has the following differences:

- the input sequences are replaced from $x_S(i-j) = x_S(i) - x_S(j)$, where $x_S(i) = x_a \cdot \sin i\alpha$ as in [29], to $x^C(i-j) = x^C(i) - x^C(j)$, where $x^C(i) = x_a \cdot \cos i\alpha$, as in Equation (9);
- certain constants involved in the multiplications in the post-processing stage change from $\cos i\alpha$ in [29] to $\sin i\alpha$, in Equation (4), and from $\cos[(2k+1)\alpha/2]$ in [29] to $\sin[(2k+1)\alpha/2]$ in Equation (3).

Thus, we have obtained the unified VLSI algorithm for type IV DCT/DST from Section 2.2 that can be used to implement an efficient VLSI unified architecture with minor differences in the pre-processing and post-processing stages.

Moreover, since the first matrix–vector product has the same constants as the 4th one and the 2nd one with the 5th one and the 3rd one, with the 6th one we can further considerably reduce the hardware complexity using an interleaving technique while maintaining high-speed performance because the delay on the critical path is very small ($3T_a$, where T_a is the delay of an adder). Thus, two matrix–vector products having the same constant vector are executed on the same linear systolic array in an interleaving manner.

The constants involved can be represented in a fixed-point format using a signed-digit (SD) representation as in Table 1 and implemented using only adders/subtracters and shift operations that are implemented without any hardware cost by appropriate hardware interconnection.

Table 1. Signed-digit representation of multiplier coefficients $s_a(i)$, $s_b(i)$ and $s_c(i)$ from the processing elements of the systolic arrays implementing Equations (42), (43), (47), (49), (51), and (55).

Multiplier Coefficient (C)	Representation	Approximate Fixed-Point Value ($\hat{C}$)	$\log_2 \lvert C-\hat{C} \rvert$	Number of Adders/ Subtractors
$s_a(1) = s(4) - s(5) = 2\cdot(\sin 16\alpha - \sin 20\alpha)$	$2^{-1} + 2^{-4} - 2^{-6} - 2^{-8}$	0.54296875000	−10.3	3
$s_a(2) = s(3) + s(6) = 2\cdot(\sin 12\alpha + \sin 24\alpha)$	$2^{1} + 2^{-1} - 2^{-5} - 2^{-8}$	2.46484375000	−10.3	3
$s_a(3) = s(1) + s(2) = 2\cdot(\sin 4\alpha + \sin 8\alpha)$	$2^{1} + 2^{-1} + 2^{-4} + 2^{-6} - 2^{-9}$	2.57617187500	−10.4	4
$s_b(1) = s(4) = 2\cdot\sin 16\alpha$	$2^{1} - 2^{-3} - 2^{-8} - 2^{-10}$	1.87011718750	−13.5	3
$s_b(2) = s(3) = 2\cdot\sin 12\alpha$	$2^{1} - 2^{-6} + 2^{-10}$	1.98535156250	−13.9	2
$s_b(3) = s(1) = 2\cdot\sin 4\alpha$	$2^{0} - 2^{-4} - 2^{-7}$	0.92968750000	−12.0	2
$s_c(1) = s(3) - s(5) = 2\cdot(\sin 12\alpha - \sin 20\alpha)$	$2^{-1} + 2^{-3} + 2^{-5} + 2^{-8} - 2^{-10}$	0.65917968750	−17.0	4
$s_c(2) = s(1) - s(6) = 2\cdot(\sin 4\alpha - \sin 24\alpha)$	$2^{-1} - 2^{-4} + 2^{-6} - 2^{-9}$	0.45117187500	−11.5	3
$s_c(3) = s(2) + s(4) = 2\cdot(\sin 8\alpha + \sin 16\alpha)$	$2^{2} - 2^{-1} + 2^{-6} + 2^{-11}$	3.51611328125	−13.1	3

Table 2 summarizes the synthesis results of the unified DCT IV/DST IV core using Cadence's Genus synthesis tools and NCSU's 15nm FreePDK [30] process design kit with Nangate's standard cell library [31] under different clock period constraints. The parameterized input data width and output data width has been set to 8 bits.

The first five lines have almost identical results in terms of area and static power due to the synthesis being under-constrained and the tool finding almost identical solutions (which is also reflected in almost the same delay along the critical path). We have presented these five results as distinct due to the dynamic power being evaluated at the constrained clock frequency, which is different for the five netlists, to better emphasize the dependence of the dynamic power on the clock frequency.

The last entry in the table represents the highest clock frequency for which the constraints have still been met. It can be observed that the type IV DCT/DST hardware accelerator core can potentially operate at a frequency of 7.29 GHz (pre-place and route

estimation) while dissipating 16.7 mW of dynamic power. For more stringent power constraints (e.g., in mobile devices) one can choose a lower operating frequency.

Table 2. Summary of synthesis results for the unified DCT IV/DST IV core.

Constrained Clock Period/ Frequency	Critical Path Delay + Setup Time [ps]	Interconnect Area [μm^2]	Combinational Area [μm^2]	Flop Area [μm^2]	Total Area [μm^2]	Equivalent Gates Count	Static Power [μW]	Dynamic Power at Constrained Clock Frequency [mW]
50 ns/20 MHz	218	277.8	443.1	505.4	1226.2	60,831	34.9	0.04 at 20 MHz
10 ns/100 MHz	218	277.8	443.1	505.4	1226.2	60,831	34.9	0.2 at 100 MHz
1 ns/1 GHz	218	280.1	445.2	505.4	1230.7	61,053	35.1	1.8 at 1 GHz
300 ps/3.33 GHz	219	281.5	446.1	505.2	1232.7	61,154	35.2	6.0 at 3.33 GHz
250 ps/4 GHz	219	281.9	446.4	505.2	1233.4	61,190	35.2	7.2 at 4 GHz
200 ps/5 GHz	196	304.1	461.2	505.2	1270.5	63,029	36.2	9.5 at 5 GHz
175 ps/5.71 GHz	175	326.6	484.3	505.2	1316.1	65,291	37.7	11.3 at 5.71 GHz
150 ps/6.67 GHz	150	351.9	533.4	505.2	1390.5	68,984	41.1	14.2 at 6.67 GHz
137 ps/7.29 GHz	137	379.0	597.0	505.2	1481.1	73,479	45.3	16.7 at 7.29 GHz

From Table 2, it can be seen that our unified VLSI architecture for type IV DCT and DST has a reduced hardware complexity of about 1250 μm^2 and a low power of 0.04 mW at 20 MHz, but due to the fact that the delay on the critical path is under 200 ps, we can also obtain high-speed performance using pipelining and increasing the clock frequency (up to 7 GHz). Due to the fact that the main advantage of the unified VLSI architecture is its low hardware complexity/power, it can be used in such applications with restricted resources.

Table 3 summarizes the post place and route results for the unified DCT IV/DST IV core. The place and route (PnR) has been performed using Cadence's Innovus PnR tool for the most constrained five netlists obtained at the synthesis step. Out of the five considered netlists, for the last three netlists, the PnR tool has found a solution with a slightly higher critical path delay for a similar area reported by the synthesis tool. Analyzing the PnR results, one can observe the penalty incurred in terms of operating frequency and dynamic power compared to the synthesis results for a similar area. Still, even for the most constrained design, the achievable operating frequency is close to the one predicted by the synthesis tool: 6.89 GHz vs 7.29 GHz, with a relative difference of 5.5%. One can observe that the reported dynamic power is more significantly underestimated by the synthesis tool: 16.7 mW at 7.29 GHz reported from synthesis compared to 22.9 mW at 6.89 GHz, equivalent to a 45.5% relative difference (extrapolating the PnR dynamic power consumption to the same 7.29 GHz operating frequency).

Table 3. Post place and route results for the unified DCT IV/DST IV core.

Constrained Clock Period [ps]	Slack [ps]	Critical Path Delay + Setup Time [ps]	Operating Frequency [GHz]	Interconnect Area [μm^2]	Cell Area [μm^2]	Total Area [μm^2]	Static Power [μW]	Dynamic Power at Operating Frequency [mW]
250	30.12	219.88	4.55	191.26	1083.80	1275.1	41.9	12.1
200	0.29	199.71	5.01	194.72	1103.41	1298.1	43.2	14.0
175	−4.51	179.51	5.57	199.53	1130.64	1330.2	45.1	16.0
150	−9.19	159.19	6.28	212.87	1206.29	1419.2	49.9	19.4
137	−8.14	145.14	6.89	225.88	1279.97	1505.8	55.5	22.9

5. Discussion

5.1. Discussion about the Main Features of the Proposed Solution

Using a new VLSI algorithm for type IV DST, we have obtained an area-efficient unified VLSI implementation for type IV DCT/DST where most of the chip is used in common by the two transforms. The hardware complexity has been considerably reduced as compared with the best unified solution reported in the literature, presented in [30].

Moreover, the resulting unified VLSI architecture can efficiently incorporate the mode-based obfuscation technique with very low overheads. Using a parallel reformulation and a systolic array architecture paradigm, we have obtained a significant reduction of the hardware complexity and high-speed performances by exploiting concurrency both as parallelism and pipelining. The hardware complexity has been reduced by an efficient use of interleaving and by reducing the number of computational structures from eight to only six. High-speed performance has been obtained not only by using concurrency but also by reducing the delay on the critical path to as low as 137 ps. This allows the increase in the clock frequency until 7.29 GHz, as can be seen from Table 2. In the applications where high-speed performances are not mandatory, we can reduce the clock frequency, and due to its low hardware complexity, we can obtain a low power implementation with a power consumption of only 0.04 mW.

Equations (42), (43), (49), (51), (60), and (61) can be mapped to only six special arithmetic structures (a matrix–vector product with a specific form) where all the elements of the vectors are constant. This allows an efficient implementation with a low hardware complexity and high-speed performance since we are using only multipliers with a constant that can be implemented with only a few adders and subtracters. This feature has been used to further reduce the hardware complexity and, at the same time, to obtain high-speed performance through a significant reduction of the critical path delay.

The interleaving technique has been efficiently used to reduce the hardware complexity by half because three computational structures out of six use the same multiplier constants as the other three. Because the six systolic arrays can be separated into three groups of arrays having processing elements with multipliers with the same constants, we have applied a hardware sharing technique that allows us to replace of six systolic arrays with only three and process the input samples in an interleaved manner.

Therefore, as it has been shown, the proposed VLSI algorithm can be mapped to only three systolic arrays resulting from the merge of the six due to the possibility of processing the input samples from two computational structures in an interleaved manner. Due to its good topology, the proposed design is well adapted to the VLSI technology, allowing for an efficient implementation.

Also, as shown in Section 3.2, the obfuscation method has been introduced with very low overheads and consists of only nine one-bit MUXs with four inputs and one output.

5.2. Comparison with Similar Solutions

When comparing to existing unified VLSI architectures for DCT/DST IV, we can see that, in [22], the throughput is significantly lower due to the fact that we have three shorter systolic arrays operating in parallel, in contrast with two longer systolic arrays in [22]. The hardware core in [22] has N general multipliers and N adders as compared with $3(N-1)/4$ multipliers, with a constant and $3(N-1)/4$ adders in the proposed solution. Moreover, the solution proposed in [22] does not incorporate the obfuscation technique.

As compared with the unified VLSI architecture presented in [28], which is the best reported in the literature, we have a significant reduction in the hardware complexity from $2(N-1)$ general multipliers and $2(N-1)$ adders in the hardware core at only $3(N-1)/4$ multipliers and $3(N-1)/4$ adders.

As compared with [32], we have a significant reduction of the hardware complexity from $2N$ general multipliers and $2N$ adders and, at the same time, a significant reduction of the clock period, given by the iteration bound, from $T_{mul} + T_a$, where T_{mul} is the delay of a general multiplier. Also, the proposed solution cannot lead to an efficient unified VLSI architecture for type IV DCT/DST.

In Table 4, we have included the number of additions and multipliers for a recently proposed fast algorithm for DST IV, but there was no VLSI implementation reported for it. We can appreciate that if the SFG graph from [33] is implemented, one will obtain $2N$ multipliers and $2N$ adders, which would lead to a higher hardware complexity and, with the SFG graph not being regular and modular, one cannot possibly hope to obtain a

very efficient VLSI implementation. Also, it is not possible to obtain an efficient unified architecture and to include the obfuscation technique.

Table 4. Comparison in terms of main resources utilization (adders, multipliers, registers) for DCT IV/DST IV algorithms.

	This Work (Hardware Core)	**[22]**	**[28]**	**[32]**	**[33] ***
Type of transform	Unified DCT/DST IV	Unified DCT/DST IV	Unified DCT/DST IV	DCT IV	DST IV
No. of adders	$3(N-1)/4$	N	$2(N-1)$	$2N$	$\frac{3}{2}Nlog_2N$
No. of multipliers	$3(N-1)/4$	N	$2(N-1)$	$2N$	$\frac{1}{2}Nlog_2N+n$
No. of registers	24	N/A	12·8	$3N$	-

* Note: the reference [33] describes a fast algorithm for DST IV, and, in the paper, only the numbers of adders and multipliers are reported.

Due to the fact that, in our solutions, the general multipliers have been replaced with multipliers with a constant, the hardware complexity of the implementation has been further reduced as compared with the VLSI implementation in [22,28] where a general multiplier with a significant greater hardware complexity and latency has been used. Since the delay on the critical path has been considerably reduced and we have applied a hardware sharing technique by means of interleaving, we have succeeded in maintaining a low hardware complexity and achieved high-speed performance.

The above results in terms of the main resources used can be summarized in Table 4.

6. Conclusions

This paper has presented an efficient approach to obtain a unified VLSI architecture for type IV DCT and type IV DST based on a new VLSI algorithm for type IV DST specially designed for this purpose. The algorithms and architectures for the two transforms have been formulated in such a fashion that they can fully exploit hardware sharing of the core accelerator with minor differences in the pre-processing and post-processing stages. Compared to similar VLSI implementations, the obtained solution has a low hardware complexity that favors low power consumption. At the same time, we have solved an important problem in the design of VLSI integrated circuits for consumer applications by efficiently incorporating the hardware security in the design with very low overheads, without prejudice to the high-speed performance of the chip. The proposed method uses the regular and modular computational structures that have been called quasi-cycle convolutions, and the obtained architecture is inspired by the paradigm of the systolic array architecture. The obtained implementation has all the advantages of the VLSI architectures based on cycle convolution or circular correlation as a regularity, modularity, and local interconnections, and it is well suited for an efficient implementation using the VLSI technology and achieves high-speed operation due to exploiting the concurrency specific to systolic array architectures.

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

Funding: This work was supported by a grant of the Romanian Ministry of Education and Research, CNCS—UEFISCDI, project number PCE 172/2021 (PN-III-P4-ID-PCE2020-0713), within PNCDI III.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

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

Appendix A

Proof of the Equation (3)

Using an input auxiliary sequence $\{x_a(i) : i = 0, \ldots, N-1\}$ as in [29] that can be recursively computed as follows:

$$x_a(N-1) = x(N-1)$$

$$x_a(i) = \left(-1\right)^i x(i) + x_a(i+1)$$

for $i = N-2, \ldots, 0$, we can write:

$$Y(k) = x_a(0) \cdot \sin[(2k+1)\alpha/2] + 2\left\{\sum_{i=0}^{N-1}\left(-1\right)^i x_a(i) \cdot \sin[(2k+1)\alpha]\right\} \cdot \cos[(2k+1)\alpha/2]$$

for $k = 1, \ldots, N-1$.

We are introducing the auxiliary output sequence $\left\{Y_a^S(k) : k = 1, 2, \ldots, N-1\right\}$ as

$$Y_a^S(k) = \sum_{i=0}^{N-1}\left(-1\right)^i x_a(i) \sin(2k+1)\, i\alpha]$$

Thus, the output sequence $\{Y(k) : k = 1, 2, \ldots, N-1\}$ can be computed using Equation (3):

$$Y(k) = x_a(0) \cdot \sin[(2k+1)\alpha/2] + 2Y_a^S(k) \cdot \cos[(2k+1)\alpha/2]$$

References

1. Jain, A.K. A Sinusoidal Family of Unitary Transforms. *IEEE Trans. Pattern Anal. Mach. Intell.* **1979**, *PAMI-1*, 356–365. [CrossRef] [PubMed]
2. Malvar, H.S. Lapped Transforms for Efficient Transform/Subband Coding. *IEEE Trans. Acoust. Speech Signal Process.* **1990**, *38*, 969–978. [CrossRef]
3. Malvar, H. A Modulated Complex Lapped Transform and Its Applications to Audio Processing. In Proceedings of the 1999 IEEE International Conference on Acoustics, Speech, and Signal Processing, Proceedings. ICASSP99 (Cat. No.99CH36258), Phoenix, AZ, USA, 15–19 March 1999; Volume 3, pp. 1421–1424.
4. Jing, C.; Tai, H.-M. Fast Algorithm for Computing Modulated Lapped Transform. *Electron. Lett.* **2001**, *37*, 796–797. [CrossRef]
5. Davidson, G.A.; Isnardi, M.A.; Fielder, L.D.; Goldman, M.S.; Todd, C.C. ATSC Video and Audio Coding. *Proc. IEEE* **2006**, *94*, 60–76. [CrossRef]
6. Chan, Y.-H.; Siu, W.-C. On the Realization of Discrete Cosine Transform Using the Distributed Arithmetic. *IEEE Trans. Circuits Syst. Fundam. Theory Appl.* **1992**, *39*, 705–712. [CrossRef]
7. Guo, J.-I.; Liu, C.-M.; Jen, C.-W. A New Array Architecture for Prime-Length Discrete Cosine Transform. *IEEE Trans. Signal Process.* **1993**, *41*, 436. [CrossRef]
8. Cheng, C.; Parhi, K.K. A Novel Systolic Array Structure for DCT. *IEEE Trans. Circuits Syst. II Express Briefs* **2005**, *52*, 366–369. [CrossRef]
9. Meher, P.K. Systolic Designs for DCT Using a Low-Complexity Concurrent Convolutional Formulation. *IEEE Trans. Circuits Syst. Video Technol.* **2006**, *16*, 1041–1050. [CrossRef]
10. Meher, P.K.; Swamy, M.N.S. New Systolic Algorithm and Array Architecture for Prime-Length Discrete Sine Transform. *IEEE Trans. Circuits Syst. II Express Briefs* **2007**, *54*, 262–266. [CrossRef]
11. Xie, J.; Meher, P.K.; He, J. Hardware-Efficient Realization of Prime-Length DCT Based on Distributed Arithmetic. *IEEE Trans. Comput.* **2013**, *62*, 1170–1178. [CrossRef]
12. Kung Why Systolic Architectures? *Computer* **1982**, *15*, 37–46. [CrossRef]
13. White, S.A. Applications of Distributed Arithmetic to Digital Signal Processing: A Tutorial Review. *IEEE ASSP Mag.* **1989**, *6*, 4–19. [CrossRef]
14. Pilato, C.; Garg, S.; Wu, K.; Karri, R.; Regazzoni, F. Securing Hardware Accelerators: A New Challenge for High-Level Synthesis. *IEEE Embed. Syst. Lett.* **2018**, *10*, 77–80. [CrossRef]

15. Knechtel, J.; Patnaik, S.; Sinanoglu, O. Protect Your Chip Design Intellectual Property: An Overview. In Proceedings of the International Conference on Omni-Layer Intelligent Systems, Crete, Greece, 5–7 May 2019; Association for Computing Machinery: New York, NY, USA, 2019; pp. 211–216.
16. Zhang, J. A Practical Logic Obfuscation Technique for Hardware Security. *IEEE Trans. Very Large Scale Integr. VLSI Syst.* **2016**, *24*, 1193–1197. [CrossRef]
17. Koteshwara, S.; Kim, C.H.; Parhi, K.K. Hierarchical Functional Obfuscation of Integrated Circuits Using a Mode-Based Approach. In Proceedings of the 2017 IEEE International Symposium on Circuits and Systems (ISCAS), Baltimore, MD, USA, 28–31 May 2017; pp. 1–4.
18. Koteshwara, S.; Kim, C.H.; Parhi, K.K. Key-Based Dynamic Functional Obfuscation of Integrated Circuits Using Sequentially Triggered Mode-Based Design. *IEEE Trans. Inf. Forensics Secur.* **2018**, *13*, 79–93. [CrossRef]
19. Parhi, K.K.; Koteshwara, S. Dynamic Functional Obfuscation. U.S. Patent US11061997B2, 3 August 2017.
20. Murthy, N.R.; Swamy, M.N.S. On the On-Line Computation of DCT-IV and DST-IV Transforms. *IEEE Trans. Signal Process.* **1995**, *43*, 1249–1251. [CrossRef]
21. Kidambi, S.S. Recursive Implementation of the DCT-IV and DST-IV. In Proceedings of the 1998 IEEE Symposium on Advances in Digital Filtering and Signal Processing, Symposium Proceedings (Cat. No.98EX185), Victoria, BC, Canada, 5–6 June 1998; pp. 106–110.
22. Chiper, D.F.; Ahmad, M.O.; Swamy, M.N.S. A Unified VLSI Algorithm for a High Performance Systolic Array Implementation of Type IV DCT/DST. In Proceedings of the International Symposium on Signals, Circuits and Systems ISSCS2013, Iasi, Romania, 11–12 July 2013; pp. 1–4.
23. Lai, S.-C.; Chien, W.-C.; Lan, C.-S.; Lee, M.-K.; Luo, C.-H.; Lei, S.-F. An Efficient DCT-IV-Based ECG Compression Algorithm and Its Hardware Accelerator Design. In Proceedings of the 2013 IEEE International Symposium on Circuits and Systems (ISCAS), Beijing, China, 19–23 May 2013; pp. 1296–1299.
24. Chiper, D.F. A New VLSI Algorithm for a High-Throughput Implementation of Type IV DCT. In Proceedings of the 2016 International Conference on Communications (COMM), Bremen, Germany, 13–15 June 2016; pp. 17–20.
25. Perera, S.M. Signal Flow Graph Approach to Efficient and Forward Stable DST Algorithms. *Linear Algebra Its Appl.* **2018**, *542*, 360–390. [CrossRef]
26. Chiper, D.F.; Cotorobai, L.T. A New VLSI Algorithm for an Efficient VLSI Implementation of Type IV DST Based on Short Band-Correlation Structures. In Proceedings of the 2020 13th International Conference on Communications (COMM), Bucharest, Romania, 18–20 June 2020; pp. 69–72.
27. Perera, S.M.; Liu, J. Complexity Reduction, Self/Completely Recursive, Radix-2 DCT I/IV Algorithms. *J. Comput. Appl. Math.* **2020**, *379*, 112936. [CrossRef]
28. Chiper, D.F.; Cotorobai, L.-T. A New Approach for a Unified Architecture for Type IV DCT/DST with an Efficient Incorporation of Obfuscation Technique. *Electronics* **2021**, *10*, 1656. [CrossRef]
29. Chiper, D.F. An Improved VLSI Algorithm for an Efficient VLSI Implementation of a Type IV DCT That Allows an Efficient Incorporation of Hardware Security with a Low Overhead. *Electronics* **2023**, *12*, 243. [CrossRef]
30. FreePDK15 | NC State EDA. Available online: https://eda.ncsu.edu/freepdk15/ (accessed on 3 October 2023).
31. Open-Cell Library. Available online: https://si2.org/open-cell-library/ (accessed on 3 October 2023).
32. Luo, C.-H.; Ma, W.-J.; Juang, W.-H.; Kuo, S.-H.; Chen, C.-Y.; Tai, P.-C.; Lai, S.-C. An ECG Acquisition System Prototype Design With Flexible PDMS Dry Electrodes and Variable Transform Length DCT-IV Based Compression Algorithm. *IEEE Sens. J.* **2016**, *16*, 8244–8254. [CrossRef]
33. Perera, S.M.; Lingsch, L.E. Sparse Matrix Based Low-Complexity, Recursive, and Radix-2 Algorithms for Discrete Sine Transforms. *IEEE Access* **2021**, *9*, 141181–141198. [CrossRef]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.