J Supercomput (2019) 75:1336–1349
https://doi.org/10.1007/s11227-018-2436-0

High-performance architecture for digital transform processing

H. Mora[1] · M. T. Signes-Pont[1] · A. Jimeno-Morenilla[1] · J. L. Sánchez-Romero[1]

Published online: 25 May 2018

Abstract The digital transforms are intensive in multiplication and accumulation operations which have a high computational cost. Advances in computer arithmetic and digital technologies allow simplifying the processing of complex algorithms when they are implemented in modern circuits. New computation techniques can be explored to provide efficient operational methods for implementing algorithms that avoid much of the complex and costly mathematical operations. This work aims to design a high-performance architecture for computing some common digital transforms. The proposed architecture has been compared to other methods. The transform used as example in this work is the discrete cosine transform. The results show that the proposal offers high-performance results comparable or better than best-known methods.

Keywords Digital transform implementation · Computational techniques design · Computer arithmetic · DCT

✉ H. Mora
hmora@ua.es; hmora@dtic.ua.es

M. T. Signes-Pont
teresa@dtic.ua.es

A. Jimeno-Morenilla
jimeno@dtic.ua.es

J. L. Sánchez-Romero
sanchez@dtic.ua.es

1 Department of Computer Science Technology and Computation, University of Alicante, 03690 Alicante, Spain

1 Introduction

The digital signal processing is a powerful tool to better understand the real world around us. These operations analyze the information to make it more accessible to our understanding and handling. However, their calculation requires in many cases, costly mathematical computing. Much research effort is currently engaged in reducing the computational costs of their implementation in order to meet the constraints of new applications and devices.

The new capacities in circuit manufacture provided by VLSI and ULSI high-density integration methods open up new possibilities in the design of digital transform operators. Regular circuit modeling allows compact structures and provides efficiency advantages in VLSI integration over other methods which require complex control stages. Therefore, it is proposed to take advantage of better integration techniques for designing direct calculation of the arithmetic expressions to facilitate and simplify its implementation in signal processors. This idea could lay the basis of designing a processor core for a set of transforms to build a high-performance digital computing system.

We are based on the DCT transform analyzed in previous research [1] as example to describe how the calculation architecture works. The intensive arithmetic processing requirements of the compression methods based on DCT or in other transforms justify the development of recent computing models and specialized hardware architectures that can meet time constraints while complying with the limitations of being embedded in new consumer devices such as media players, mobile phones or tablets.

The remainder of this paper is organized as follows: The problem is defined in Sect. 2; in Sect 3, the main design techniques and hardware implementations are reviewed. Section 4 exposes the proposed architecture, and Sect. 5 evaluates its performance in area and delay cost. Section 6 provides a comparison with other techniques, and conclusions are drawn in Sect. 7.

2 Problem definition

The efficient calculation of the transforms is made using the separability property. That is, the 2-D transform is organized in two consecutive 1-D transforms. The separable transforms adopt a simpler formulation that reduces complexity and allows the calculation of this matrix separately for rows and columns. Their general expression is shown in (1).

$$T\left(u\right)=\sum_{x=0}^{N-1} f\left(x\right) \mathrm{g}\left(x,u\right), \tag{1}$$

where $g(x, u)$ is the *kernel* of the transform and $u \in 0\ldots N-1$.

As representative of this computation problem, the DCT method is analyzed. The 1-D DCT of eight samples $\{f(x), x = 0, 1, \ldots, 7\}$ can be expressed as:

$$F\,(u) = \frac{C\,(u)}{2}\sum_{x=0}^{7} f\,(x)\cos\left[\frac{(2x+1)\,u\pi}{16}\right], \tag{2}$$

where the kernel of the transform is:

$$g\,(x,\,u) = \cos\left[\frac{(2x+1)\,u\pi}{16}\right] \text{and}\, C\,(u) = \begin{cases} \frac{1}{\sqrt{2}}, & \text{if} u = 0 \\ 1. & \text{otherwise} \end{cases}$$

The scaling factor (1/2) can be eliminated without loss of generality to facilitate calculation. In a deployed way, the above (2) formula can be written as:

$$F\,(u) = C_{uo} f\,(0) + C_{u1} f\,(1) + \cdots + C_{u7} f\,(7) \tag{3}$$

where: $C_{0\mathrm{j}}$, $\forall\, j \in \{0..7\}$, is $C_{0\mathrm{j}} = \cos\left[\frac{\pi}{4}\right]$, and C_{uj}, with $u \in \{1..7\}$ and $j \in \{0..7\}$, is $C_{uj} = \cos\left[\frac{(2j+1)u\pi}{16}\right]$

The 2-D DCT of an 8 × 8 block sample $\{f(x, y), x = 0, 1, \ldots, 7, \text{and } y = 0, 1, \ldots, 7\}$ is generally expressed as follows:

$$F\,(u, v) = \frac{C\,(u)\,C\,(v)}{4}\sum_{x=0}^{7}\sum_{y=0}^{7} f\,(x, y)\cos\left[\frac{(2x+1)\,k\pi}{16}\right]\cos\left[\frac{(2y+1)\,k\pi}{16}\right], \tag{4}$$

where $C(u)$ and $C(v)$ are defined with the same (2) equation.

Other mentioned transforms have similar computing needs. For example, the 1-D discrete wavelet transform (DWT) of N samples:

$$W_{\Phi}\,(u) = \sum_{x=0}^{N} f\,(x)\,\Phi\,(x, u)\,, \tag{5}$$

where $\Phi\,(x, u)$ defines the type of wavelet transform.

In the case of the 1-D fast Fourier transform (FFT) of eight samples $\{f(x), x = 0, \ldots, 7\}$, it can be expressed as:

$$F\,(u) = \sum_{x=0}^{7} f\,(x)\,W_8^{ux}, \tag{6}$$

where $W_8 = e^{-2\pi i/8}$. In this case, complex arithmetic is needed in the calculations.

3 Related work

For each transform, many calculation methods and implementation designs have been developed. The proposals consider performance issues such as resource requirements, power consumption and execution delay of the calculation. In addition, the hardware architectures and the VLSI implementation of the methods must take into consideration the critical aspects of the applications and/or devices in which the transforms will be executed; for example, in the case of DCT, the multimedia apps running on consumer electronics devices. So that, the methods try to avoid or reduce the costly operations in the calculation process, such as multiplications which require much more hardware than addition operations. There are two main groups of proposals based on different equations of the transform which will be detailed below.

3.1 Vectorial methods

In first place, distributed arithmetic (DA) implementations are based on the vectorial formulation of expression (3). This method is an efficient technique to process these operations when one of the vectors is fixed due to the regularity of its implementation. Recently, extensive research for this method of calculation has been developed. There are two broad approaches to implementing a DA computation unit: (a) ROM based and (b) adder based. (a) The ROM based or conventional DA approach accelerates multiplication operations precalculating them for each input value. Then, the results are produced by *shift-and-add* procedures. This method requires a ROM or lookup tables (LUTs) to store these precalculated products [2]. (b) The second type, adder based, does not require LUTs and can exploit the distribution of binary value patterns of constant operands [3, 4]. Nevertheless, these methods need more addition operations and are slower than previous ROM-based approaches.

Other transforms can be processed similarly using these methods. As example, there are proposals for FFT and for DWT (discrete wavelet transform) [5].

3.2 Flow-graph methods

Flow-graph algorithms (FGA) take advantage of symmetries of the calculations derived from Eq. (1) to reduce the necessary operations. Numerous designs have been proposed to minimize the necessary multiplication [6–8] methods. They are characterized by calculation flow-graph results and butterfly configurations. Figure 1 depicts Loeffler configuration as representative of these methods.

Hardware implementations of flow-graph algorithms for DCT calculation try to minimize the impact of involved floating-point operations, especially multiplication operations. Some works have been proposed to replace them by additions and shifts [9]. Other proposals perform the calculations without multiplications by approximating these operations with integer fractions with power of two denominators. So, each multiplication is replaced by a series of integer additions and shifts [10]. Although this technique avoids multiplications, slightly lower quality of the compressed image is obtained. In the same way, other techniques replace expensive multiplications by means of the CORDIC method and perform the butterfly configuration calculations by rotating angles [11]. In these methods, the necessary final compensation of the CORDIC method is included in the quantization stage of the image compression.

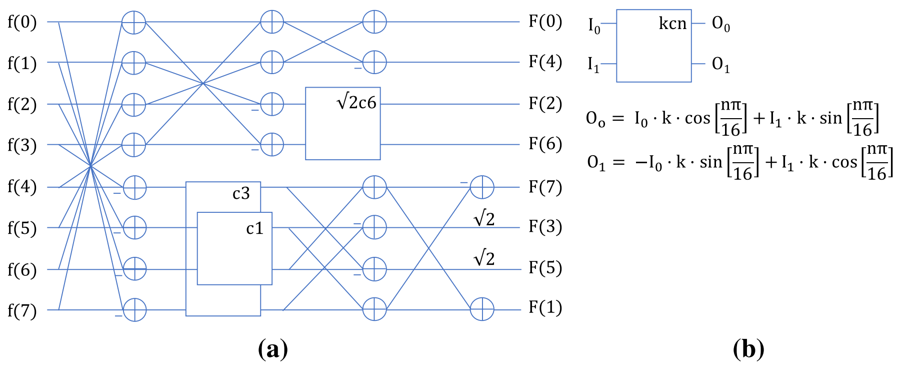
f(0)
f(1)
f(2)
f(3)
f(4)
f(5)
f(6)
f(7)
F(0)
F(4)
F(2)
F(6)
F(7)
F(3)
F(5)
F(1)
√2c6
c3
c1
√2
√2
(a)
kcn
I_0
I_1
O_0
O_1
$O_0 = I_0 \cdot k \cdot \cos\left[\frac{n\pi}{16}\right] + I_1 \cdot k \cdot \sin\left[\frac{n\pi}{16}\right]$
$O_1 = -I_0 \cdot k \cdot \sin\left[\frac{n\pi}{16}\right] + I_1 \cdot k \cdot \cos\left[\frac{n\pi}{16}\right]$
(b)

Fig. 1 One-dimensional Loeffler algorithm. **a** Flow graph; **b** the rotator block with equations

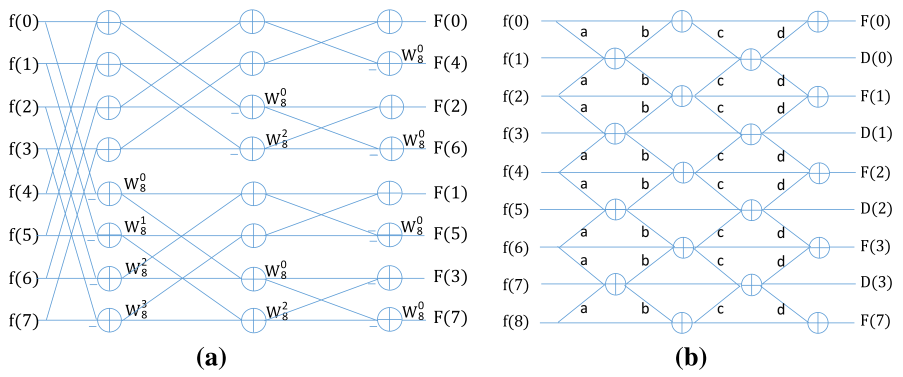

Fig. 2 Transforms flow graphs. **a** FFT (decimation-in-frequency), where $W_8^k = e^{-2\pi ik/8}$; **b** 9/7 DWT, where a, b, c, d are constants

Other transforms can be calculated following similar schemes. For example, Fig. 2 shows flow graphs for 1-D FFT and DWT (used in JPEG 2000 standard [12]).

3.3 Other methods

There are other proposals of calculation methods based on formulation which try to adapt the implementation to specific applications; for example, proposing a new recursive formulation based on weighted sum operations [13], prime length [14] and custom VLSI implementations [15, 16].

4 Architecture

The main idea of the proposed architecture consists in integrating arithmetic processing techniques into the transform's design by replacing the multiplication and addition operations by regular calculation trees. This approach can be generalized for other separable transforms. As scientific contribution, the proposed method will allow to

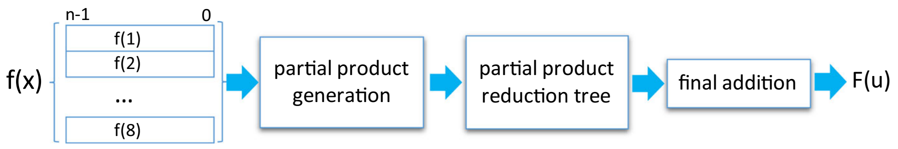

Fig. 3 General scheme of the architecture

build high-performance architectures for digital signal transform processing based on the same core function. As technical contribution, this approach does not need ROM modules and implements a direct integration of the operations which avoids irregular routing paths. In the next section, these ideas are described.

The methodology uses the basic mathematical formulation of the DCT described by (2) and (3). The input $n = 8$ is the most common input size for this transform since it corresponds with the vector size specifications for the standard JPEG compression method [17].

According to the previous expressions, the calculation of each $F(u)$ contains eight multiplications and seven additions. The operands of each of these multiplications are an $f(x)$ input value and a constant C_{uj}. The set of constants are depending on the kernel $g(x, u)$ of the transform. The key design features of the proposed architecture are two: (a) how multiplication and addition operations required for each $F(u)$ are composed and (b) how the constants C_{uj} are included in the processing of each multiplication.

(a) The proposed idea to design this stage consists in computing the eight multiplications and seven additions of each $F(u)$ in a combined way, so that the partial product reductions are integrated with the additions required in each calculation. This approach allows for accelerating a variety of arithmetic circuits where additions and multiplications are present. This technique should improve overall computation of the transform, where former operations are very frequent, instead of performing complicated designs for reducing the complexity.

In general terms, the proposed architecture implements a design for each $F(u)$ composed of the three multiplication stages: partial product generation, partial product reduction and final addition [18]. Figure 3 shows the general scheme of calculating each $F(u)$. This proposal does not have to compute symmetries nor decomposition operations for transform calculation to reduce the processing delay because all addition and multiplication operations involved in each $F(u)$ are considered in an integrated way.

(b) The calculation of partial products of the multiplications by the constants is performed by a *direct partial product generation* method. This proposed method uses neither single direct generation due to the large number of partial products generated nor Booth's method because of the additional delay that incorporates. Therefore, the proposal is to make a direct generation of only nonzero digits. For this, the distribution of bits of each constant represented in fixed point is analyzed, and then a custom circuit is created for each case. To improve the generation of partial products, we use signed-digit format to represent those constants and so minimize the number of nonzero digits while keeping the error within the p bits

Table 1 CSD coded calculation constants of DCT

Ci	Decimal value	Binary value	CSD binary value	Partial products
$\cos[\pi/16]$	0.9807852	0.111110110001	1.0000 $\bar{1}$ 0110001	5
$\cos[\pi/8]$	0.9238795	0.111011001000	1.00 $\bar{1}$ 011001000	5
$\cos[3\pi/16]$	0.8314696	0.110101001101	0.1101010100 $\bar{1}$ 0	6
$\cos[\pi/4]$	0.7071067	0.101101010000	0.101101010000	5
$\cos[5\pi/16]$	0.5555702	0.100011100011	0.100100 $\bar{1}$ 00100	4
$\cos[3\pi/8]$	0.3828125	0.011000100000	0.011000100000	3
$\cos[7\pi/16]$	0.1953125	0.001100100000	0.001100100000	3

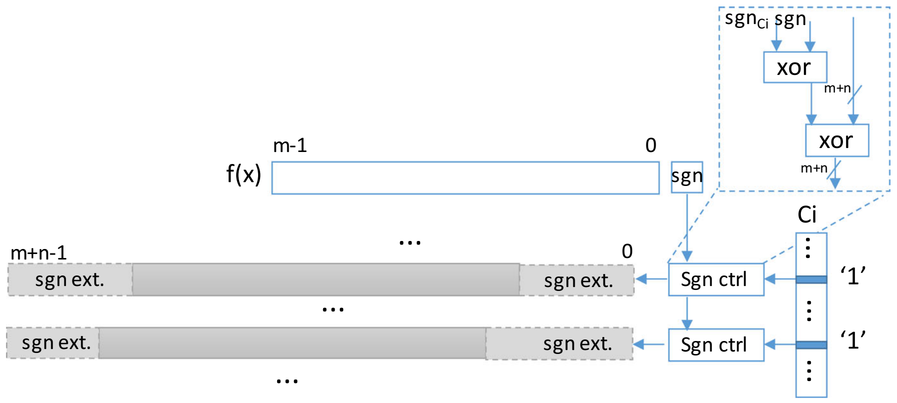

Fig. 4 Direct partial product generation with error compensation

length. The signed-digit format used is *Canonic Signed Digit* (CSD) because the number of nonzero digits is minimal [19]. Table 1 shows the constants Ci of DCT with their decimal and binary values, the signed-digit representation and the resulting partial products in each case. As shown in this table, the number of partial products generated is between 3 and 6.

The generator circuits directly produce direct or inverse partial products according to the sign of each digit. The next figure (Fig. 4) shows the production scheme with this system for a constant from the above table. This error compensation approach from the two complementations is performed in the same way as before.

Note that the same strategies can be designed for obtaining multiplications by constants of FFT and constants of DWT. At final stage, the set of partial products to be combined is obtained. From here, these generated partial products are merged by means of a reduction tree consisting of different counters and compressor elements. The first-generation method for DCT operation produces $8 \times \lceil n/k \rceil$ partial products for each $F(u)$. The detailed configuration of the tree can be determined from the precision given by n and k. Then, from the value of n and k, several configurations can be given with different numbers of partial products generated.

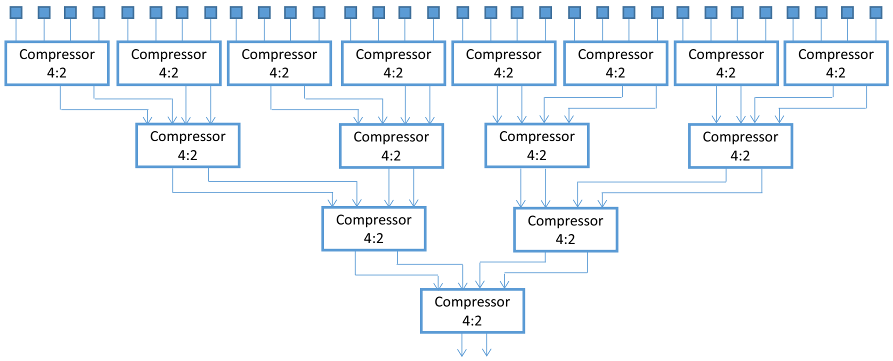

Fig. 5 Reduction tree design. For 32 partial products

The direct generation method produces between 32 and 40 partial products for n = 12 bit length and between 24 and 32 for n = 8 bit length. The reduction trees can be configured with the outputs from each F_i generator as well. As in the previous case, the signed numbers are coded in two's complement, so this representation uses 24 bits including sign. Figure 5 shows an example for 32 partial products. As shown, trees composed of 4:2 compressor elements have a more regular structure than an ordinary CSA tree made of (3,2) counters because the partial products are added up in the form of a binary tree. This is a very good feature for regular VLSI implementations [20]. Another approach is to use high-order column compressors, such as 5:2 compressors, instead of 3:2 counters to increase the reduction ratio [1, 21].

The final addition of the operation can be made by any addition known method. This is one important reason why it is more efficient, in terms of computational cost, to compute all collected multiplications and additions of each $F(u)$ as a whole instead of alternating additions, multiplications or shifts operations as proposed in FGA-based methods. Another reason is that the method can be easily hardwired in VLSI integrated circuit, whereas in vector-based methods, there are necessary ROM modules with precalculated data (ROM based) or complex control stages to perform the additions (adder based).

5 Evaluation of the performance of architecture

In first place, analytical tests comparing the accuracy results to those from other methods (the comparison reference used is MATLAB "*dct*" function with simple precision operands) have been made. We have obtained an average error $< 2^{-6}$ for inputs of 12-bit length and an error $< 2^{-2}$ for inputs of 8-bit length. The evaluation experiment consists of using the results produced within computing compression JPEG algorithm of an image. We have obtained an SNR = 20 dB and SNR = 24 dB for 8- and 12-bit length of the inputs, respectively, using a compression rate = 10. The resulting images are shown in Fig. 6 (Lena image. Picture taken from http://sipi.usc.edu/database/).

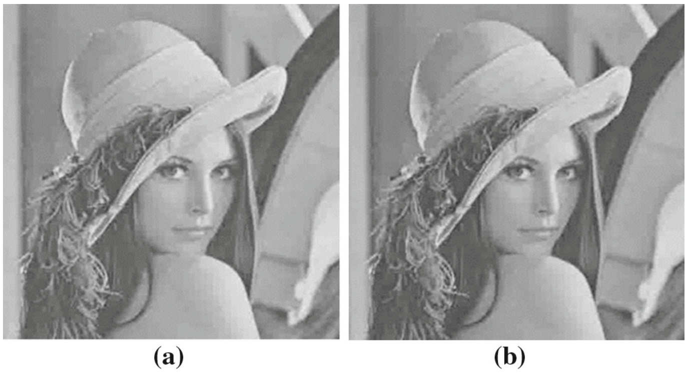

Fig. 6 Subjective quality of JPEG algorithm using direct DCT calculation method. **a** 8-bit input length; **b** 12-bit input length

Next, the performance of the proposed technique is analyzed in terms of area and computing delay. The precomputed constants are stored in lookup tables (LUTs) near the transform operator inside the arithmetic unit [22]. For performance evaluation, we want to keep the highest precision recommended by the IEEE 1180-1990 standard [23] for DCT implementation. Therefore, the input signals have been set to 12-bit length and the paths between stages to 16-bit wide. These precision characteristics set n and m to 12 bits in length. An additional one bit will be added in the multiplicand to encode the sign. The partial results of the additions and reductions will be set to the most significant 16 bits, so the loss of precision in the results will be insignificant. A block size of $k = 4$ is established for addressing each entry.

A homogeneous convention in calculating area cost and response times of the elements involved is assumed to facilitate comparison with other designs or proposals. Let τ_a be the area occupied by a complex gate, for example an *xor* gate. Thus, it is easy to translate the performance results into another scale of comparison, such as number of transistors or occupied area in μm^2 [24]. In the same way, let τ_t be the delay of a complex gate. The following table (Table 2) summarizes the estimated area and delay cost of the main elements of the architecture in homogeneous terms. These data came from well-known calculation methods in the computer arithmetic literature.

The constants registers bank and their addressing by partial product generation by multiplexors do not have a delay because they are not stored in memory and it is not necessary to read them. The addressing of the registers toward the inputs of multiplexers is straightforward, and then, the right constant values are directly selected. The next subsections analyze the performance of the proposed architecture in terms of time delay and hardware resources. The results are summarized in Table 3.

5.1 Area cost

The contributions to the required area for implementing the $F(u)$ calculation come from (i) processing the stages depicted in Fig. 1, that are: partial product generation stage;

Table 2 Area and delay cost estimation

Device	Area (τ_a)	Delay (τ_t)
3:2 counter	3	2
4:2 compressor	6	3
5:2 compressor	9	4
12-bit adder (CLA)	36	6
16-bit adder (CLA)	48	8
12-bit multiplier	109	26
Up to 8-bit input LUT	30 Γa /K-bit	3.5
16:1 multiplexor	16	2
k-bit RAC (ROM accumulator)	$2^k \cdot$ precision	
k-bit register	k	
12-bit shifter	48	4

Table 3 Area resources and delay cost estimations of the proposed designs

	Area (τ_a)	Delay (τ_t)
(i) Partial product generation	960	2
(ii) Partial product reduction	2544	13
(iii) Final addition	48	8
(iv) Constants registers bank	84	–
$F(u)$ total area cost	3636	23

(ii) partial product reduction stage; (iii) final addition; and (iv) storing the constants needed. Next, each contribution is detailed.

(i) The estimation of area of this stage is obtained only for the combinational circuit described by Fig. 2. This configuration requires 24 τ_a to compute each partial product and, consequently, 120 τ_a to process each multiplication. The $F(u)$ generation requires a total estimation of 960 τ_a.

(ii) Regarding the partial product reduction tree, the delay cost is related to the number of addends to reduce. With full precision, the result of each multiplication is 24-bit long, where each partial product also has a length of 24 bits. Using the estimations shown in Table 2, the partial product reduction tree requires a total area 2544 τ_a to simultaneously produce results of 24-bit length.

(iii) The last stage is the final addition of the two addends produced by the tree. Although the reduction tree works with addends of 24-bit long, the final addition will be made to the most significant 16 bits according to the recommendations of accuracy that set the standard [24]. The known carry look-ahead addition method requires an area cost of 48 τa to perform a 16-bit addition.

(iv) One LUT module can store all the necessary constants of the transform. Moreover, in the case of register-based generation, implementation will also be necessary to store the odd multiples of these constants. In the case of DCT calculation, it gives a total estimation of area to store that information of 7×12 bits = 84 bits for the constants and $7 \times 7 \times 12 = 588$ bits for their multiples. The assumed correspondence is a complex gate to store a bit in a flip-flop [25], so the encoding information requires 84 τ_a for constants and 588 τ_a for their multiples.

5.2 Delay cost

The delay cost of the proposed method is obtained by adding the response times of each stage: partial product generation stage (i), partial product reduction stage (ii) and the final addition (iii).

(i) The partial product generation stage is very fast due to the proposed configurations. For direct generation, the delay is only 2 τ_t.
(ii) The partial product reduction tree spends 13 τ_t for 40 addends according to the reduction tree structure described.
(iii) Finally, the estimated delay of CLA addition for 16 bits length is 8 τ_t.

6 Discussion and comparisons

6.1 Area comparisons

At first place, the basic design based on the direct implementation of a DCT algorithm is taken from Eq. (2). According to it, the elements required to calculate each $F(u)$ consist of a LUT for storing the constants of C, a multiplier and an adder operator. These components need a combined area of 237 τ_a.

Secondly, the designs according to the vectorial methods have two main types of designs: (a) DA ROM based and (b) DA adder based:

(a) DA ROM-based technique mainly consists of addition-accumulation iterations of results read from memory (RAC—ROM accumulator). The main contributors to required hardware for computing the transform by means of these methods are the memory components. In addition, the necessary hardware to build the accumulation registers and parallel adders of the first stage must be added. Consequently, the design requires a combined area between 25 Kτ_a for the standard method and 2.56 Kτ_a for efficient hardware implementations with smaller ROMs.
(b) DA adder-based architectures do not need ROM components. However, they need more addition operations to compute the inner products [2]. An area cost of 1.5 Kτ_a can be estimated [3] by reusing adder operators as the specification shows.

And finally, the architectures based on the FGA algorithms minimize the number of addition and multiplication operations needed for DCT calculation. For example, the implementations based on Loeffler method require only 11 multiplications and 29 additions [8]. Considering just the resources to implement these elements, we obtain an area estimation of 237 τ_a to compute 1D DCT. In a parallel architecture, eight adder and six multiplier units would be necessary to perform all computations required. This arithmetic processing has an area cost of 1118 τ_a, apart from necessary control circuitry. For the FGA-based proposals that replace multiplications by CORDIC rotations, we consider only the smallest number of iterations to obtain the required accuracy. According to the designs outlined in the literature [11], between 48 and 56 additions with the corresponding shift instructions are needed to perform the calculations. If execution is carried out in a serial way, a CORDIC processor and a multiplier for the final compensation will be necessary. This configuration requires 349 τ_a to compute a whole DCT. A parallel configuration needs eight adders, three CORDIC processors and six multipliers. The resulting hardware cost is 1614 τ_a.

6.2 Delay comparisons

The basic scheme based on Eq. (3) requires eight LUT accesses, eight multiplications and seven additions to compute each $F(u)$. These operations have a combined delay cost of 1208 τ_t. This cost justifies the need to spend effort to reduce it, especially in applications with hard time constraints.

The path delay of the DA designs lies in the accumulated iterations with data obtained from the LUTs [15]. The number of iterations is linear with operands precision (12 bits); in each iteration, a data access and an addition are processed. The delay costs of these components for calculating each $F(u)$ without considering the control iteration circuitry and shifts are 94 τ_t. Thus, in the case of adder-based DA architectures, nearly three times more addition operations are required than in the previous method. Since the adder operator is the most significant contributor to the final delay, we can estimate a time cost of 282 τ_t to compute each $F(u)$ of DCT.

Implementations based on the FGA Loeffler algorithm require 518 τ_t for serial execution and 84 τ_t in a parallel implementation. The CORDIC implementations of the FGA algorithm require a runtime estimation between 620 τ_t and 648 τ_t. Finally, the optimized design with hardware resources for concurrent execution would require only 146 τ_t time delay.

The results of the estimations shown in Table 4 demonstrate that the proposal described in this paper significantly reduces the execution time of DA-based designs and maintains comparable times to those achieved by FGA-based methods.

7 Conclusions

In this work, a new calculation method is proposed for digital separable transforms based on their direct mathematical expressions. The architecture provides a compact structure for performing operations that do not require ROMs to perform multiplications.

This approach does not need to insert precomputed stages to reduce the operations involved. It has proven that a simple approach can provide good performance results

Table 4 Area and delay cost estimation comparison

Method	Area (τ_a)	Delay (τ_t)
Basic direct design (Eq. 3)	237	1208
DA ROM-based [2]	2.5 K	94
DA adder based [3]	1.5 K	282
Serial Loeffler FGA[a] [8]	237	518
Parallel Loeffler FGA[a] [8]	1.1 K	84
Serial Loeffler CORDIC FGA[a] [11]	349	648
Parallel improved Loeffler CORDIC FGA[a] [11]	1.6 K	146
Proposed direct calculation	3.6 K	23

[a]Full 1-D DCT

keeping in mind its subsequent VLSI implementation. In comparison with the existing designs, the approach offers some advantages that can be explored for high-speed calculators. The results of the tests have shown that the implementation's performance is comparable to the best-known methods based on both vectorial and FGA designs.

Currently, we are developing an arithmetic unit prototype to exploit the potential of using the same calculation core for processing digital separable transforms. In addition, we are exploring to apply these ideas in other convolutional calculations where sequences of arithmetic operations (multiplications and additions) are needed.

Acknowledgements This work was supported by the Spanish Research Agency (AEI) and the European Regional Development Fund (FEDER) under project "CloudDriver4Industry" TIN2017-89266-R.

References

1. Mora H, Signes-Pont MT, Azorín-López J, Corral Sánchez L (2015) High-speed architecture for direct computation of DCT In: International Conference on systems, control, signal processing and informatics, pp 176–183
2. Sungwook Y, Swartziander EE (2001) DCT implementation with distributed arithmetic. IEEE Trans Comput 50(9):985–991. https://doi.org/10.1109/12.954513
3. Shams AM, Chidanandan A, Pan A, Bayoumi MA (2006) NEDA: a low-power high-performance DCT architecture. IEEE Trans Signal Process 54(3):955–964. https://doi.org/10.1109/TSP.2005.862755
4. Sharma VK, Mahapatra KK, Pati UC (2011) An efficient distributed arithmetic based VLSI architecture for DCT. In: International conference on devices and communications, pp 1–5. https://doi.org/10.1109/icdecom.2011.5738484
5. Bernabé G, Hernández R, Acacio ME (2016) Parallel implementations of the 3D fast wavelet transform on a Raspberry Pi 2 cluster. J Supercomput. https://doi.org/10.1007/s11227-016-1933-2
6. Chen WH, Smith C, Fralick S (1977) A fast computational algorithm for the Discrete Cosine Transform. IEEE Trans Commun 25(9):1004–1009. https://doi.org/10.1109/TCOM.1977.1093941
7. Vetterli M, Kovacevic J (2013) Wavelets and subband coding. CreateSpace Independent Publishing Platform. ISBN: 978-1484886991

8. Loeffler C, Lightenberg A, Moschytz GS (1989) Practical fast 1-D DCT algorithms with 11-multiplications. Proc of ICASSP Glagow 2:988–991. https://doi.org/10.1109/ICASSP.1989.266596
9. El Aakif M, Belkouch S, Chabini N, Hassani MM (2011) Low power and fast DCT architecture using multiplier-less method. Faible Tension Faible Consomm. https://doi.org/10.1109/FTFC.2011.5948920
10. Liang J, Tran TD (2001) Fast multiplierless approximations of the DCT with the lifting scheme. IEEE Trans Signal Process 49(12):3032–3044. https://doi.org/10.1109/78.969511
11. Huang H, Xiao L (2013) CORDIC based fast Radix-2 DCT algorithm. IEEE Signal Process Lett 20(5):483–486. https://doi.org/10.1109/LSP.2013.2252616
12. Ghodhbani R, Saidani T, Horrigue L et al (2017) An efficient pass-parallel architecture for embedded block coder in JPEG 2000. Real-Time Image Proc. https://doi.org/10.1007/s11554-017-0666-7
13. Signes MT et al (2009) Improvement of the Discrete Cosine Transform calculation by means of a recursive method. Math Comput Model 50:750–764. https://doi.org/10.1016/j.mcm.2009.05.004
14. Xie J, Meher PK, He J (2013) Hardware-efficient realization of prime-length DCT based on distributed arithmetic. IEEE Trans Comput 62(6):1170–1178. https://doi.org/10.1109/TC.2012.64
15. Coutinho VA et al (2016) A multiplierless pruned DCT-like transformation for image and video compression that requires ten additions only. J Real-Time Image Process 12(2):247–255. https://doi.org/10.1007/s11554-015-0492-8
16. Revathi KG, Reeja Malar J (2016) Efficient diagonal data mapping for large size 2D DCT/IDCT using single port SRAM based transpose memory. Conf Electr Electron Optim Tech, Int. https://doi.org/10.1109/ICEEOT.2016.7755651
17. ISO/IEC (1994) Information technology-digital compression and coding of continuous-tone still images-requirements and guidelines. ISO 81: 09-92, 1994
18. Mora-Mora H, Mora-Pascual J, Sánchez-Romero JL, García-Chamizo JM (2008) Partial product reduction by using look-up tables for M $\times$ N multiplier. Integr VLSI J 41(4):557–571. https://doi.org/10.1016/j.vlsi.2008.01.005
19. Tanaka Y (2016) Efficient signed-digit-to-canonical-signed-digit recoding circuits. Microelectron J 57:21–25. https://doi.org/10.1016/j.mejo.2016.09.001
20. Mora H et al (2017) Mathematical model and implementation of rational processing. J Comput Appl Math 309:575–586. https://doi.org/10.1016/j.cam.2016.05.001
21. Ghasemzadeh M, Mahdavi S, Zokaei A, Hadidi K (2016) A new ultra high speed 5-2 compressor with a new structure. In: International conference on mixed design of integrated circuits and systems, pp 151–154. https://doi.org/10.1109/mixdes.2016.7529721
22. Mora H et al (2010) Mathematical model of stored logic based computation. Math Comput Model 52(7–8):1243–1250. https://doi.org/10.1016/j.mcm.2010.02.034
23. IEEE Std. 1180-1990 (1990) IEEE standard specification for the implementation of 8 $\times$ 8 inverse cosine transform. Institute of Electrical and Electronics Engineers, International Standard, New York, USA
24. Joshi A et al (2017) A comparative performance analysis of various CMOS design techniques for XOR and XNOR circuits. Int J Res Appl Sci Eng. https://doi.org/10.22214/ijraset.2017.4241
25. Bernardi P, Restifo M, Sánchez E, Sonza Reorda M (2017) On the in-field test of embedded memories. International symposium on on-line testing and robust system design (IOLTS). https://doi.org/10.1109/IOLTS.2017.8046236