Hindawi Publishing Corporation
EURASIP Journal on Advances in Signal Processing
Volume 2007, Article ID 45321, 17 pages
doi:10.1155/2007/45321

Research Article

Calculation Scheme Based on a Weighted Primitive: Application to Image Processing Transforms

María Teresa Signes Pont, Juan Manuel García Chamizo, Higinio Mora Mora, and Gregorio de Miguel Casado

Departamento de Tecnología Informática y Computación, Universidad de Alicante, 03690 San Vicente del Raspeig, 03071 Alicante, Spain

Received 29 September 2006; Accepted 6 March 2007

Recommended by Nicola Mastronardi

This paper presents a method to improve the calculation of functions which specially demand a great amount of computing resources. The method is based on the choice of a weighted primitive which enables the calculation of function values under the scope of a recursive operation. When tackling the design level, the method shows suitable for developing a processor which achieves a satisfying trade-off between time delay, area costs, and stability. The method is particularly suitable for the mathematical transforms used in signal processing applications. A generic calculation scheme is developed for the discrete fast Fourier transform (DFT) and then applied to other integral transforms such as the discrete Hartley transform (DHT), the discrete cosine transform (DCT), and the discrete sine transform (DST). Some comparisons with other well-known proposals are also provided.

1. INTRODUCTION

Mathematical notation aside, the motivation behind integral transforms is easy to understand. There are many classes of problems that are extremely difficult to solve or, at least, quite unwieldy from the algebraic standpoint in their original domains. An integral transform maps an equation from its original domain (time or space domain) into another domain (frequency domain). Manipulating and solving the equation in the target domain is, ideally, easier than manipulating and solving it in the original domain. The solution is then mapped back into the original domain. Integral transforms work because they are based upon the concept of spectral factorization over orthonormal bases. Equation (1) shows the generic formulation of a discrete integral transform where $f(x)$, $0 \le x < N$, and $F(u)$, $0 \le u < N$, are the original and the transformed sequences, respectively. Both have $N = 2^n$ values, $n \in \mathbb{N}$ and $T(x,u)$ is the kernel of the transform

$$F(u) = \sum_{x=0}^{N-1} T(x,u) f(x). \tag{1}$$

The inverse transform can be defined in a similar way. Table 1 shows some integral transforms ($j = \sqrt{-1}$ as usual).

The Fourier transform (FT) is a reference tool in image filtering [1, 2] and reconstruction [3]. A fast Fourier transform (FFT) scheme has been used in OFDM modulation (orthogonal frequency division multiplexing) and has shown to be a valuable tool in the scope of communications [4, 5]. The most relevant algorithm for FFT calculation was developed in 1965 by Cooley and Tukey [6]. It is based on a successive folding scheme and its main contribution is a computational complexity reduction that decreases from $O(N^2)$ to $O(N \cdot \log_2 N)$. The variants of FFT algorithms follow different ways to perform the calculations and to store the intervening results [7]. These differences give rise to different improvements such as memory saving in the case of in-place algorithms, high speed for self-sorting algorithms [8] or regular architectures in the case of constant geometry algorithms [9]. These improvements can be extended if combinations of the different schemes are envisaged [10]. The features of the different algorithms point to different hardware trends. The in-place algorithms are generally implemented by pipelined architectures that minimize the latency between stages and the memory [11] whereas the constant geometry algorithms

Table 1: Some integral transforms.

Transform	*Kernel* $T(x,u)$	*Remarks*
Fourier	$\frac{1}{N}\exp\left(\frac{-2j\pi ux}{N}\right)$	Trigonometric kernel
Hartley	$\cos\left(\frac{2\pi ux}{N}\right)+\sin\left(\frac{2\pi ux}{N}\right)$	Trigonometric kernel
Cosine	$e(k)\cos\frac{(2x+1)\pi u}{2N}$	Trigonometric kernel with $e(0)=\frac{1}{\sqrt{2}}$, $e(k)=1, 0<k<N$
Sine	$e(k)\sin\frac{(2x+1)\pi u}{2N}$	Trigonometric kernel with $e(0)=\frac{1}{\sqrt{2}}$, $e(k)=1, 0<k<N$

have an easier control because of their regular structure based on a constant indexation through all the stages. This allows parallel data processing by a column of processors with a fixed interconnecting net [12, 13].

The Hartley transform is a Fourier-related transform which was introduced in 1942 by Hartley [14] and is very similar to the discrete Fourier transform (DFT), with analogous applications in signal processing and related fields. Its main distinction from the DFT is that it transforms real inputs into real outputs, with no intrinsic involvement of complex numbers. The discrete Hartley transform (DHT) analogue of the Cooley-Tukey algorithm is commonly known as the fast Hartley transform (FHT) algorithm, and was first described in 1984 by Bracewell [15–17]. The transform can be interpreted as the multiplication of the vector $(x_0,\ldots,x_{N-1})$ by an $N\times N$ matrix; therefore, the discrete Hartley transform is a linear operator. The matrix is invertible and the DHT is its own inverse up to an overall scale factor. This FHT algorithm, at least when applied to power-of-two sizes N, is the subject of a patent issued in 1987 to the University of Stanford. The University of Stanford placed this patent in the public domain in 1994 [18]. The DHT algorithms are typically slightly less efficient (in terms of the number of floating-point operations) than the corresponding FFT specialized for real inputs or outputs [19, 20]. The latter authors published the algorithm which achieves the lowest operation count for the DHT of power-of-two sizes by employing a split-radix algorithm, similar to that of the FFT. This scheme splits a DHT of length N into a DHT of length $N/2$ and two real-input DFTs (not DHTs) of length $N/4$. A priori, since the FHT and the real-input FFT algorithms have similar computational structures, none of them appears to have a substantial speed advantage [21]. As a practical matter, highly optimized real-input FFT libraries are available from many sources whereas highly optimized DHT libraries are less common. On the other hand, the redundant computations in FFTs due to real inputs are much more difficult to eliminate for large prime N, despite the existence of $O(N\cdot\log_2 N)$ complex-data algorithms for that cases. This is because the redundancies are hidden behind intricate permutations and/or phase rotations in those algorithms. In contrast, a standard prime-size FFT algorithm such as Rader's algorithm can be directly applied to the DHT of real data for roughly a factor of two less computation than that of the equivalent complex FFT. This DHT approach currently appears to be the only way known to obtain such factor-of-two savings for large prime-size FFTs of real data [22]. A detailed analysis of the computational cost and specially of the numerical stability constants for DHT of types I–IV and the related matrix algebras is presented by Arico et al. [23]. The authors prove that any of these DHTs of length $N=2^t$ can be factorized by means of a divide–and–conquer strategy into a product of sparse, orthogonal matrices where in this context sparse means at most two nonzero entries per row and column. The sparsity joint with orthogonality of the matrix factors is the key for proving that these new algorithms have low arithmetic costs and an excellent normwise numerical stability.

DCT is often used in signal and image processing, especially for lossy data compression, because it has a strong "energy compaction" property: most of the signal information tends to be concentrated in a few low-frequency components of the DCT [24, 25]. For example, the DCT is used in JPEG image compression, MJPEG, MPEG [26], and DV video compression. The DCT is also widely employed in solving partial differential equations by spectral methods [27] and fast DCT algorithms are used in Chebyshev approximation of arbitrary functions by series of Chebyshev polynomials [28]. Although the direct application of these formulas would require $O(N^2)$ operations, it is possible to compute them with a complexity of only $O(N\cdot\log_2 N)$ by factorizing the computation in the same way as in the fast Fourier transform (FFT). One can also compute DCTs via FFTs combined with $O(N)$ pre- and post-processing steps. In principle, the most efficient algorithms are usually those that are directly specialized for the DCT [29, 30]. For example, particular DCT algorithms resemble to have a widespread use for transforms of small, fixed sizes such as the 8×8 DCT used in JPEG compression, or the small DCTs (or MDCTs) typically used in audio compression. Reduced code size may also be a reason for using a specialized DCT for embedded-device applications. However, even specialized DCT algorithms are typically closely related to FFT algorithms [22]. Therefore, any improvement in algorithms for one transform will theoretically lead to immediate gains for the other transforms too [31]. On the other hand, highly optimized FFT programs are widely available. Thus, in practice it is often easier to obtain high performance for generalized lengths of N with FFT-based algorithms. Performance on modern hardware is typically not simply dominated by arithmetic counts and optimization requires substantial engineering effort.

As DCT which is equivalent to a DFT of real and even functions, the discrete sine transform (DST) is a Fourier-related transform using a purely real matrix [25]. It is equivalent to the imaginary parts of a DFT of roughly twice the length, operating on real data with odd symmetry. As for DCT, four main types of DST can be presented. The boundary conditions relate the various DCT and DST types.

Table 2: Definition of the operation $\oplus$ for $k = 1$.

$a \oplus b$	$01 = 1$	$10 = 00 = 0$	$11 = -1$
$01 = 1$	$\alpha + \beta$	α	$\alpha - \beta$
$10 = 00 = 0$	β	0	$-\beta$
$11 = -1$	$-\alpha + \beta$	$-\alpha$	$-\alpha - \beta$

The applications of DST are similar to those for DCT as well as its computational complexity. The problem of reflecting boundary conditions (BCs) for blurring models that lead to fast algorithms for both deblurring and detecting the regularization parameters in the presence of noise is improved by Serra-Capizzano in a recent work [32]. The key point is that Neumann BC matrices can be simultaneously diagonalized by the fast cosine transform DCT III and Serra-Capizzano introduces antireflective BCs that can be related to the algebra of the matrices that can be simultaneously diagonalized by the fast sine transform DST I. He shows that, in the generic case, this is a more natural modeling whose features are both, on one hand a reduced analytical error, since the zero (Dirichlet) BCs lead to discontinuity at the boundaries, the reflecting (Neumann) BCs lead to C° continuity at the boundaries, while his proposal leads to C^1 continuity at the boundaries, and on the other hand fast numerical algorithms in real arithmetic for deblurring and estimating regularization parameters.

This paper presents a method that performs function evaluation by means of successive iterations on a recursive formula. This formula is a weighted sum of two operands and it can be considered as a primitive operation just as computational usual primitives such as addition and shift. The generic definition of the new primitive can be achieved by a two-dimensional table in which the cells store combinations of the weighting parameters. This evaluation method is suitable for a great amount of functions, particularly when the evaluation needs a lot of computing resources, and allows implementation schemes that offer a good balance between speed, area saving, and error containing. This paper is focused on the application of the method for the discrete fast Fourier transform with the purpose to extend the application to other related integral transforms, namely DHT, DCT, and DST.

The paper is structured in seven parts. Following the introduction, Section 2 defines the weighted primitive. Section 3 presents the fundamental concepts of the evaluation method based on the use of the weighted primitive, outlining its computational relevance. Some examples are presented for illustration. In Section 4, an implementation based on look-up tables is discussed and an estimation of the time delay, area occupation, and calculation error is developed. Section 5 is entirely devoted to the applications of our method for digital signal processing transforms. The calculation of the DFT is developed as a generic scheme and other transforms, namely the DHT, the DCT, and the DST are considered under the scope of the DFT. In Section 6 some comparisons with other well-known proposals considering operation counts, area, time delay, and stability estimations are presented. Finally, Section 7 summarizes results and presents the concluding remarks.

2. DEFINITION OF A WEIGHTED PRIMITIVE

The weighted primitive is denoted as $\oplus$ and its formal definition is as follows:

$$\begin{aligned} &\oplus : \mathbb{R} \times \mathbb{R} \bullet \mathbb{R}, \\ &(a, b) \bullet a \oplus b = \alpha a + \beta b, \\ &(\alpha, \beta) \in \mathbb{R}^2. \end{aligned} \tag{2}$$

The operation $\oplus$ can also be defined by means of a two-input table. Table 2 defines the operation for integer values in binary sign-magnitude representation; k stands for the number of significant bits in the representation.

In Table 2 the arguments have been represented in binary and decimal notation and the results are referred to in a generic way as combinations of the parameters α and β. The operation $\oplus$ is performed when the arguments (a, b) address the table and the result is picked up from the corresponding cell. The first argument (a) addresses the row whereas the second (b) addresses the column.

The same operation can be represented for greater values of k (see Table 3, for $k = 2$). Central cells are equivalent to those of Table 2.

The amount of cells in a table is $(2^{(k+1)} - 1)^2$ and it only depends on k. These cells are organized as concentric rings centred in 0. It can be noticed that increasing k causes a growth in the table and therefore the addition of more peripheral rings. The number of rings increases 2^k when k increases one unit. The smallest table is defined for $k = 1$ but the same information about the operation $\oplus$ is provided for any k value. When the precision of the arguments n is greater than k, these must be fragmented in k-sized fragments in order to perform the operation. So, t double accesses are necessary to complete t cycles of a single operation (if $n = k \cdot t$). A single operation requires picking up from a table so many partial results as fragments are contained in the argument. The overall result is obtained by adding t partial results according to their position.

As the primitive involves the sum of two products, the arithmetic properties of the operation $\oplus$ have been studied with respect to those of the addition and multiplication.

Commutative

$$\begin{aligned} &\forall (a, b) \in \mathbb{R}^2,\ a \oplus b = b \oplus a \\ &\iff \alpha a + \beta b = \alpha b + \beta a \iff (a - b)(\alpha - \beta) = 0 \\ &\iff a = b \text{ (trivial case)} \\ &\iff \alpha = \beta \text{ (usual sum).} \end{aligned} \tag{3}$$

As shown, the commutative property is only verified when $a = b$ or when $\alpha = \beta$.

Table 3: Definition of the operation $\oplus$ for $k = 2$.

$a \oplus b$	$011 = 3$	$010 = 2$	$001 = 1$	$100 = 000 = 0$	$101 = -1$	$110 = -2$	$111 = -3$
$011 = 3$	$3\alpha + 3\beta$	$3\alpha + 2\beta$	$3\alpha + \beta$	3α	$3\alpha - \beta$	$3\alpha - 2\beta$	$3\alpha - 3\beta$
$010 = 2$	$2\alpha + 3\beta$	$2\alpha + 2\beta$	$2\alpha + \beta$	2α	$2\alpha - \beta$	$2\alpha - 2\beta$	$2\alpha - 3\beta$
$001 = 1$	$\alpha + 3\beta$	$\alpha + 2\beta$	$\alpha + \beta$	α	$\alpha - \beta$	$\alpha - 2\beta$	$\alpha - 3\beta$
$100 = 000 = 0$	3β	2β	β	0	$-\beta$	-2β	-3β
$101 = -1$	$-\alpha + 3\beta$	$-\alpha + 2\beta$	$-\alpha + \beta$	$-\alpha$	$-\alpha - \beta$	$-\alpha - 2\beta$	$-\alpha - 3\beta$
$110 = -2$	$-2\alpha + 3\beta$	$-2\alpha + 2\beta$	$-2\alpha + \beta$	-2α	$-2\alpha - \beta$	$-2\alpha - 2\beta$	$-2\alpha - 3\beta$
$111 = -3$	$-3\alpha + 3\beta$	$-3\alpha + 2\beta$	$-3\alpha + \beta$	-3α	$-3\alpha - \beta$	$-3\alpha - 2\beta$	$-3\alpha - 3\beta$

Associative

$$\begin{aligned} &\forall (a,b,c) \in \mathbb{R}^3, \\ &a \oplus (b \oplus c) = \alpha a + \beta(\alpha b + \beta c) = \alpha a + \beta\alpha b + \beta\beta c, \\ &(a \oplus b) \oplus c = \alpha(\alpha a + \beta b) + \beta c = \alpha\alpha a + \alpha\beta b + \beta c. \end{aligned} \tag{4}$$

As noticed, the operation $\oplus$ is not associative except for a particular case given by $\alpha a(1-\alpha) = \beta c(1-\beta)$.

The lack of associative property obliges to fix arbitrarily an order in calculations execution. We assume that the operations are performed from left to right:

$$\begin{aligned} &a_1 \oplus a_2 \oplus a_3 \oplus a_4 \cdots \oplus a_q \\ &\quad = (\cdots(((a_1 \oplus a_2) \oplus a_3) \oplus a_4) \cdots \oplus a_q). \end{aligned} \tag{5}$$

Neutral element

$$\begin{aligned} &\forall a \in R,\ \exists e \in \mathbb{R},\ a \oplus e = e \oplus a = a \\ &\iff \alpha a + \beta e = a \\ &\iff \alpha e + \beta a = a. \end{aligned} \tag{6}$$

No neutral element can be identified for this operation.

Symmetry

Spherical symmetry can be proved by looking at the table:

$$\forall (a,b) \in \mathbb{R}^2, \quad -[a \oplus b] = -a \oplus -b. \tag{7}$$

Proof

$$\begin{aligned} -[a \oplus b] &= -(\alpha a + \beta b) = -\alpha a - \beta b \\ &= \alpha(-a) + \beta(-b) = -a \oplus -b. \end{aligned} \tag{8}$$

So, $a \oplus b$ and $-[a \oplus b]$ are stored in diametrically opposite cells.

The primitive $\oplus$ does not fulfill the properties that allow the definition of a set structure. □

3. A FUNCTION EVALUATION METHOD BASED ON THE USE OF A WEIGHTED PRIMITIVE

This section presents the motivation and the fundamental concepts of the evaluation method based on the use of the weighted primitive, outlining its computational relevance.

3.1. Motivation

In order to improve the calculation of functions which demand a great amount of computing resources, the approach developed in this paper aims for balancing the number of computing levels with the computing power of the corresponding primitive. That is to say, the same calculation may get the advantages steaming from the calculation at a lower computing level by other primitives than the usual ones whenever the new primitives assume intrinsically part of the complexity. This approach is considered as far as it may be a way to perform a calculation of functions with both algorithmic and architectural benefits.

Our inquiry for a primitive operation that bears more computing power than the usual primitive sum points towards the operation $\oplus$. This new primitive is more generic (usual sum is a particular case of weighted sum) and, as it will be shown, the recursive application of $\oplus$ achieves quite different features that mean much more than the formal combination of sum and multiplication. This issue has crucial consequences because function evaluation is performed with no more difficulty than applying iteratively a simple operation defined by a two-input table.

3.2. Fundamental concepts of the evaluation method

In order to carry out the evaluation of a given function Ψ we propose to approximate it through a discrete function F defined as follows:

$$\begin{aligned} &F_0 \in \mathbb{R}, \\ &F_{i+1} = F_i \oplus G_i, \quad \forall i,\ i \in \mathbb{N},\ F_i \in \mathbb{R},\ G_i \in \mathbb{R}. \end{aligned} \tag{9}$$

The first value of the function F is given by (F_0) and the next values are calculated by iterative application of the recursive equation (9). The approximation capabilities of function F can be understood as the equivalence between two sets of real values: on one hand $\{F_i\}$ and on the other hand $\{\Psi(i)\}$ which is generated by the quantization of the function Ψ. The independent variable in function Ψ is denoted by $z = x + ih$, where $x \in \mathbb{R}$ is the initial value, $h \in \mathbb{R}$ is the quantization step, and $i \in \mathbb{N}$ can take successive increasing values. The mapping implies three initial conditions to be fulfilled. They are

(a) x (initial Ψ value) is mapped to 0 (index of the first F value), that is to say $\Psi(x) \equiv F_0$;

Table 4: Approximation of some usual generic functions by the recursive function F.

Usual function Ψ	*Mapping parameters for* F			
	F_0	α	β	$\mathbf{G_i}$
Linear $\Psi(z) = mz$	$F_0 = 0$	$\alpha = 1$	$\beta = h$	$G_i = m$
Trigonometric $\Psi(z) = \cos(z)$	$F_0 = 1$	$\alpha = \cos(h)$	$\beta = -\sin(h)$	$G_i = -\sin(i-1)h$
$\Psi(z) = \sin(z)$	$F_0 = 0$	$\alpha = \cos(h)$	$\beta = \sin(h)$	$G_i = \cos(i-1)h$
Hyperbolic $\Psi(z) = \cosh(z)$	$F_0 = 1$	$\alpha = \cosh(h)$	$\beta = \sinh(h)$	$G_i = \sinh(i-1)h$
$\Psi(z) = \sinh(z)$	$F_0 = 0$	$\alpha = \cosh(h)$	$\beta = \sinh(h)$	$G_i = \cosh(i-1)h$
Exponential $\Psi(z) = e^z$	$F_0 = 1$	$\alpha = \cosh(h)$	$\beta = \sinh(h)$	$G_i = F_{i-1}$

(b) the successive samples of function Ψ are mapped to successive F_i values irrespectively to the value of the quantization step, h;

(c) the two previous assumptions allow not having to discern between i (index belonging to the independent variable of Ψ) and i (iteration number of F), that is to say:

$$\Psi(z) = \Psi(x + ih) \equiv F_i. \tag{10}$$

The mapping of function Ψ by the recursive function F succeeds in approximating it through the normalization defined in (a), (b), and (c). It can be noticed that the function F is not unique. Since different mappings related to different values of the quantization step h can be achieved to approximate the same function Ψ, different parameters α and β can be suited.

Table 4 shows the approximation of some usual generic functions. The first column shows different functions Ψ that have been quantized. The next four columns present the mapping parameters of the corresponding recursive functions F. All cases are shown for $x = 0$.

Any calculation of $\{F_i\}$ is performed with a computational complexity $O(N)$ whenever $\{G_i\}$ is known or whenever it can be carried out with the same (or less) complexity. It can be outlined that the interest of the mapping by the function F is concerned with the fulfillment of this condition. This fact draws at least two different computing issues. The first develops new function evaluation upon the previous; that is to say, when function F has been calculated, it can play the role of G in order to generate a new function F. This spreading scheme provides a lot of increasing computing power, always with linear cost. The second scheme deals with the crossed paired calculation of functions F and G; that is to say, G is the auxiliary function involved in the calculation of F as well as F is the auxiliary function for calculation of G. In addition to the linear cost, the crossed calculation scheme provides time delay saving as both functions can be calculated simultaneously.

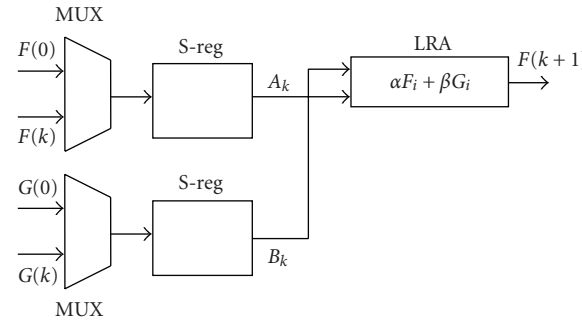


Figure 1: Arithmetic processor for the spreading calculation scheme.

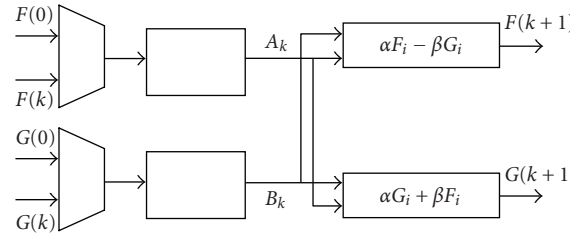


Figure 2: Arithmetic processor for the crossed paired evaluation.

4. PROCESSOR IMPLEMENTATION

As mentioned in Section 3, the two main computing issues lead to different architectural counterparts. The development of a new function evaluation upon the previous one in a spreading calculation scheme is carried out by the processor presented in Figure 1 that requires function G to be known. The second scheme deals with the crossed paired calculation of the F and G functions. The corresponding processor is shown in Figure 2.

The implementation proposed uses an LRA (acronym for look-up table (LUT), register, reduction structure, and adder). The LUT contains all partial products $\alpha A_k + \beta B_k$; A_k, B_k are portions of few bits of the current input data F_i and G_i.

Table 5: Arithmetic processor estimations of area cost and time delay for 16 bits and one-bit fragmented data.

Hardware devices		*Occupied area*	*Time delay*
Multiplexer		$0.25 \cdot \times 2 \times 16\tau_a = 8\tau_a$	$0,5\tau_t$
Shift register		$0.5 \times 16\tau_a = 8\tau_a$	$15 \times 0,5\tau_t = 7,5\tau_t$
LRA	LUT	$40\ \tau_a$/Kbit $\times 16$ bits $\times$ 16 cell $= 10\tau_a$	$3.5\tau_t \times 16$ accesses $= 56\tau_t$
	Register	$0.5 \times 16 \cdot \tau_a = 8\tau_a$	$1\tau_t$
	Reduction structure 4 : 2 + adder	$4\tau_a + 16\tau_a = 20\tau_a$	3 red. $\times\ 3\tau_t + \lg 16\tau_t = 13\tau_t$
Arithmetic processor (Figure 1)		$70\tau_a$	$78\tau_t$
Arithmetic processor (Figure 2)		$108\tau_a$	$78\tau_t$

Table 6: Relationship between area, time delay, and fragment length k, for 16 bits data for processor 2.

	$k = 1$	$k = 2$	$k = 4$	$k = 8$	$k = 16$
LUT area	$20\tau_a$	$80\tau_a$	$2048\tau_a$	$524288\tau_a$	$34359738368\tau_a$
LUT area versus overall area	$\frac{20\tau_a}{108\tau_a} = 0.18$	$\frac{80\tau_a}{168\tau_a} = 0.47$	$\frac{2048\tau_a}{2136\tau_a} = 0.96$	> 0.99	> 0.99
LUT time access	$56\tau_t$	$28\tau_t$	$14\tau_t$	$7\tau_t$	$3\tau_t$
LUT time access versus overall processing time	$\frac{56\tau_t}{78\tau_t} = 0.72$	$\frac{28\tau_t}{50\tau_t} = 0.56$	$\frac{14\tau_t}{36\tau_t} = 0.39$	$\frac{7\tau_t}{29\tau_t} = 0.24$	$\frac{3\tau_t}{25\tau_t} = 0.12$

On every cycle, the LUT is respectively accessed by A_k and B_k coming from the shift registers. Then, the partial products are taken out of the cells (partial products in the LUT are the hardware counterpart of the weighted primitives presented in Tables 1 and 2). The overall partial product $\alpha F_i + \beta G_i$ is obtained by adding all the shifted partial products corresponding to all fragment inputs A_k, B_k of F_i and G_i, respectively. In the following iteration, both the new calculated F_{i+1} value and the next G_{i+1} value are multiplexed and shifted before accessing the LUT in order to repeat the addressing process. The processor in Figure 2 is different from Figure 1 in what concerns function G. The G values are obtained in the same way as for F but the LUT for G is different from the LUT for F.

4.1. Area costs and time delay estimation

In order to have the capability to make a comparison of computing resources, an estimation of the area cost and time delay of the proposed architectures is presented here. The model we use for the estimations is taken from the references [33, 34]. The unit τ_a represents the area of a complex gate. The complex gate is defined as the pair (AND, XOR) that provides a meaningful unit, as these two gates implement the most basic computing device: the one bit full-adder. The unit τ_t is the delay of this complex gate. This model is very useful because it provides a direct way to compare different architectures, without depending on their implementation features. As an example, the area cost and time delay for 16 bits one-bit fragmented data are estimated for both processors, as shown in Table 5.

If the fragments of the input data are greater than one bit, then the occupied area and the time delay access of the LUT vary. The relationship between area, time delay, and fragment length k for 16 bits data is shown in Table 6 for processor 2.

Table 6 outlines that the LUT area increases exponentially with k, and represents an increasing portion of the overall area as k increases. The access time for the LUT decreases as $1/k$. The percentage of access time versus overall processing time decreases slowly as $1/k$. The trade-off between area and time has to be defined depending on the application.

The proposed architecture has also been tested in the XS4010XL-PC84 FPGA. Time delay estimation in usual time units can also be provided assuming $\tau_t \approx 1$ ns.

4.2. Algorithmic stability

A complete study of the error is still under consideration and numerical results are not yet available except for particular cases [35]. Nevertheless, two main considerations are presented: on one hand, the recursive calculation accumulates the absolute error caused by the successive round-off which is performed as the number of iterations increases, on the other hand, if round-off is not performed, the error can become lower as the length in bits of the result increases, but the occupied area as well as the time delay increase too. In what follows, both trends are analyzed.

Round-off is performed

The drawback of the increasing absolute error can be faced by decreasing the number of iterations, that is to say the number of calculated values, with the corresponding loss of

accuracy of the mapping. A trade-off between the accuracy of the approximation (related to the number of calculated values) and the increasing calculation error must be found. Parallelization provides a mean to deal with this problem by defining more computing levels. The N values of function F that are to be calculated can be assigned to different computing levels (therefore different computing processors) in a tree-structured architecture, by spreading N into a product as follows:

$$N = N_1 \cdot N_2 \cdots N_P. \tag{11}$$

- 1st computing level: F_0 is the seed value that initializes the calculation of N_1 new values,
- 2nd computing level: the N_1 obtained values are the seeds that initialize the calculation of $N_1 \cdot N_2$ new values (N_2 values per each N_1).

And so on until achieving the

- pth computing level: the N_{p-1} obtained values are the seeds that complete the calculation of $N = N_1 \cdot N_2 \cdots N_p$ new values (N_p values per each N_{p-1}).

If the error for one value calculation is assumed to be ε, the overall error after N values calculation is

- for sequential calculation $= N\varepsilon = N_1 \cdot N_2 \cdot \cdots \cdot N_p\varepsilon$,
- for calculation by a tree structured architecture $= (N_1 + N_2 + \cdots + N_p)\varepsilon$.

The parallelized calculation decreases the overall error without having to decrease the number of points. The minimum value for the overall error is obtained when the sum $(N_1 + N_2 + \cdots + N_p)$ is minimized, that is to say when all N_i in the sum are relatively prime factors.

It can be mentioned that the time delay calculation follows a similar evolution scheme as the error. Considering T as the time delay for one value calculation, the overall time delay is

- for sequential calculation $= NT = N_1 \cdot N_2 \cdot \cdots \cdot N_pT$,
- for calculation by a tree structured architecture $= (N_1 + N_2 + \cdots + N_p)T$.

The minimization of the time delay is also obtained when the N_i are relatively prime factors.

For the occupied area, the precise structure of the tree in what concerns the depth (number of computing levels) and the number of branches (number of calculated values per processor) is quite relevant for the result. The distribution of the N_i is crucial in the definition of some improving tendencies. The number of processors P in the tree-structure can be bounded as follows:

$$P = 1 + N_1 + N_1 \cdot N_2 + N_1 \cdot N_2 \cdot N_3 + \cdots + N_1 \cdot N_2 \cdot N_3 \cdot \cdots \cdot N_{p-1} < 1 + (p-1)\frac{N}{Np}. \tag{12}$$

P increases at the same rate as the number of computing levels p, but the growth can be contained if N_p is the maximum value of all N_i, that is to say in the last computing level $p - 1$, the number of calculated values per processor is the highest. It can be observed that the parallel calculation involves much more processors than sequential one processor.

Summarizing the main ideas

(i) The parallel calculation provides benefits on error bound and time delay whereas sequential calculation performs better in what concerns area saving.

(ii) A trade-off must be established between the time delay, the occupied area, and the approximation accuracy (through the definition of the computing levels).

Round-off is not performed

As explained in Section 2, we assume the first input data length is n, the data have been fragmented ($n = kt$), and the partial products in the cells are p bits long. If t accesses have been performed to the table and t partial products have to be added, the first result will be $p + t + 1$ bits long (t bits represent the increase caused by the corresponding shifts plus one bit for the last carry). The second value has to be calculated in the same way so that the $p + t + 1$ bits of the feedback data is k-fragmented and the process goes on. This recursive algorithm can be formalized as follows:

$$\begin{aligned}
&\text{Initial value} && n\,\text{bits} = A_0\,\text{bits}\\
& && p + t + 1\,\text{bits} = p + \frac{n}{k} + 1\,\text{bits}\\
&\text{1st calculated value} && = p + 1 + \frac{A_0}{k}\,\text{bits}\\
& && = A_1\,\text{bits}\\
&\text{2nd calculated value} && p + 1 + \frac{A_1}{k}\,\text{bits}\\
&\quad\cdots && \quad\cdots
\end{aligned}$$

and so on.

Table 7 presents the data length evolution and the corresponding error for $n = p = 16, 32$, and 64 bits data, as well as the number of calculated values that lead to the maximum data length achievement.

It can be noticed that the increase of the number of bits is bounded after a finite and rather low number of calculated values that decreases as k grows. As usual, the error decreases as the number of the data bits increases and the results are improved in any case by small fragmentation ($k = 2$). When round-off is not performed, time delay and area occupation increase because of the higher number of bits involved, so Tables 5 and 6 should be modified. It can be outlined that small fragmentation makes error to decrease, but time delay would increase too much. By increasing the fragment length value, time delay improves but the error and the area cost would make this issue infeasible. The trade-off between area, time delay, and error must be set regarding to the application.

Table 7: Data length evolution and error versus number of calculated values for $n = p = 16, 32$, and 64 bits.

Initial data length (bits)	*Fragment length*	*Final data length (bits)*	*Length increase rate*	*Number of calculated values*	*Error*
16	$k = 2$	34	112%	9	2^{-34}
	$k = 4$	23	44%	4	2^{-23}
	$k = 8$	19	19%	2	2^{-19}
	$k = 16$	18	12.5%	2	2^{-16}
32	$k = 2$	66	106%	10	2^{-66}
	$k = 4$	44	37.5%	5	2^{-44}
	$k = 8$	38	18.8%	4	2^{-38}
	$k = 16$	35	9.4%	3	2^{-35}
	$k = 32$	34	6.2%	2	2^{-34}
64	$k = 2$	130	103%	11	2^{-130}
	$k = 4$	86	34.3%	6	2^{-86}
	$k = 8$	74	15.6%	4	2^{-74}
	$k = 16$	69	7.8%	4	2^{-69}
	$k = 32$	67	4.7%	2	2^{-67}
	$k = 64$	66	3.1%	2	2^{-66}

5. GENERIC CALCULATION SCHEME FOR INTEGRAL TRANSFORMS

In this section, a generic calculation scheme for integral transforms is presented. The DFT is taken as a paradigm and some other transforms are developed as applications of the DFT calculation.

5.1. The DFT as paradigm

Equation (13) is the expression of the one-dimensional discrete Fourier transform. Let us have $N = 2M = 2^n$,

$$F(u) = \frac{1}{N}\sum_{x=0}^{N-1} f(x)W_{2M}^{ux}, \quad \text{where } W_N = \exp\frac{-2j\pi}{N}. \tag{13}$$

The Cooley and Tukey algorithm segregates the FT in even and odd fragments in order to perform the successive folding scheme, as shown in (14):

$$\begin{aligned} F(u) &= \frac{1}{2}\left(F_{\text{even}}(u) + F_{\text{odd}}(u)W_{2M}^{u}\right),\\ F(u+M) &= \frac{1}{2}\left(F_{\text{even}}(u) - F_{\text{odd}}(u)W_{2M}^{u}\right),\\ F_{\text{even}}(u) &= \frac{1}{M}\sum_{x=0}^{M-1} f(2x)W_{M}^{ux},\\ F_{\text{odd}}(u) &= \frac{1}{M}\sum_{x=0}^{M-1} f(2x+1)W_{M}^{ux}. \end{aligned} \tag{14}$$

For any $u \in [0, M[$, the Cooley and Tukey algorithm starts by setting the M initial two-point transforms. In the second step $M/2$ four-point transforms are carried out by combining the former transforms and so on till to reach the last step, where one M-point transform is finally obtained. For values of $u \in [M, N[$ no more extra calculations are required as the corresponding transforms can be obtained by changing the sign, as shown by the second row in (14).

Our method enhances this process by adding a new segregation held by both real (R) and imaginary (I) parts in order to allow the crossed evaluation presented at the end of Section 3. Due to the fact that two segretations are considered (even/odd, real/imaginary) there will be, for each u, four transforms, which are $R_{p,q\,\text{even}}$, $R_{p,q\,\text{odd}}$, $I_{p,q\,\text{even}}$, and $I_{p,q\,\text{odd}}$ where p, q denote the step of the process and the number of the transform in the step, respectively, $p \in [0, n-1]$, and $q \in [0, 2^{n-1} - 1]$.

Equations (15), (16), and (17) show the first, the second, and the last steps of our process, respectively, for any $u \in [0, M[$. Parameters $\alpha_p(u) = \cos p\pi u/M$ and $\beta_p(u) = \sin p\pi u/M$ define the step p. The u argument has been omitted in (16) and (17) in order to clarify the expansion. In the first step, M two-point real and imaginary transforms are set in order to start the process. In the second step $M/2$ real and imaginary transforms are carried out following the calculation scheme shown in (9). At the end of the process, one real and one imaginary M-point transform are achieved and, without any more calculation, the result is deduced for $u \in [M, N[$. As observed in (16) and (17), each step involves the results of R and I obtained in the two previous steps; therefore, in each step the number of equations is halved. After the first step, a sum is added to the weighted primitive. This could have an effect on the LUT as the parameter set becomes $(\alpha, \beta, 1)$,

$$\begin{aligned} &u \in [0, M[\\ &R_{0,0\,\text{even}}(u) = f(0) + \alpha_0(u) f\left(2^{n-1}\right),\\ &R_{0,1\,\text{odd}}(u) = f\left(2^{n-2}\right) + \alpha_0(u) f\left(2^{n-2} + 2^{n-1}\right),\\ &\qquad\qquad \cdots\\ &R_{0,M-1\,\text{odd}}(u) = f\left(2 + 2^2 + \cdots + 2^{n-2}\right)\\ &\qquad\qquad + \alpha_0(u) f\left(2 + 2^2 + \cdots + 2^{n-2} + 2^{n-1}\right), \end{aligned}$$

$$
\begin{aligned}
&I_{0,0\,\text{even}}(u) = -\beta_0(u) f(2^{n-1}),\\
&I_{0,1\,\text{odd}}(u) = -\beta_0(u) f(2^{n-2} + 2^{n-1}),\\
&\qquad\qquad \cdots\\
&I_{0,M-1\,\text{odd}}(u) = -\beta_0(u) f(2 + 2^2 + \cdots + 2^{n-2} + 2^{n-1}),
\end{aligned}
\tag{15}
$$

$$
\begin{aligned}
R_{1,0\,\text{even}} &= R_{0,0\,\text{even}} + \alpha_1 R_{0,1\,\text{odd}} - \beta_1 I_{0,1\,\text{odd}}\\
&= R_{0,0\,\text{even}} + R_{0,1\,\text{odd}} \oplus I_{0,1\,\text{odd}},\\
I_{1,0\,\text{even}} &= I_{0,0\,\text{even}} + \beta_1 R_{0,1\,\text{odd}} + \alpha_1 I_{0,1\,\text{odd}}\\
&= I_{0,0\,\text{even}} + R_{0,1\,\text{odd}} \oplus I_{0,1\,\text{odd}},\\
R_{1,1\,\text{odd}} &= R_{0,2\,\text{even}} + \alpha_1 R_{0,3\,\text{odd}} - \beta_1 I_{0,3\,\text{odd}}\\
&= R_{0,2\,\text{even}} + R_{0,3\,\text{odd}} \oplus I_{0,3\,\text{odd}},\\
I_{1,1\,\text{odd}} &= I_{0,2\,\text{even}} + \beta_1 R_{0,3\,\text{odd}} + \alpha_1 I_{0,3\,\text{odd}}\\
&= I_{0,2\,\text{even}} + R_{0,3\,\text{odd}} \oplus I_{0,3\,\text{odd}},\\
&\qquad\qquad \cdots\\
R_{1,M/2-1\,\text{odd}} &= R_{0,M/2\,\text{even}} + \alpha_1 R_{0,M/2+1\,\text{odd}} - \beta_1 I_{0,M/2+1\,\text{odd}}\\
&= R_{0,M/2\,\text{even}} + R_{0,M/2+1\,\text{odd}} \oplus I_{0,M/2+1\,\text{odd}},\\
I_{1,M/2-1\,\text{odd}} &= I_{0,M/2\,\text{even}} + \beta_1 R_{0,M/2+1\,\text{odd}} + \alpha_1 I_{0,M/2+1\,\text{odd}}\\
&= I_{0,M/2\,\text{even}} + R_{0,M/2+1\,\text{odd}} \oplus I_{0,M/2+1\,\text{odd}},
\end{aligned}
\tag{16}
$$

$$
\begin{aligned}
\mathbf{R} = R_{n-1,0} &= R_{n-2,0\,\text{even}} + \alpha_{n-1} R_{n-2,1\,\text{odd}}\\
&\quad - \beta_{n-1} I_{n-2,1\,\text{odd}}\\
&= R_{n-2,0\,\text{even}} + R_{n-2,1\,\text{odd}} \oplus I_{n-2,1\,\text{odd}},\\
\mathbf{I} = I_{n-1,0} &= I_{n-2,0\,\text{even}} + \beta_{n-1} R_{n-2,1\,\text{odd}}\\
&\quad + \alpha_{n-1} I_{n-2,1\,\text{odd}}\\
&= I_{n-2,0\,\text{even}} + R_{n-2,1\,\text{odd}} \oplus I_{n-2,1\,\text{odd}},
\end{aligned}
\tag{17}
$$

$$
\begin{aligned}
&u \in [M, N[\\
&\mathbf{R} = R_{n-1,0} = R_{n-2,0\,\text{even}} - \alpha_{n-1} R_{n-2,1\,\text{odd}}\\
&\quad + \beta_{n-1} I_{n-2,1\,\text{odd}}\\
&= R_{n-2,0\,\text{even}} - R_{n-2,1\,\text{odd}} \oplus I_{n-2,1\,\text{odd}},\\
&\mathbf{I} = I_{n-1,0} = I_{n-2,0\,\text{even}} - \beta_{n-1} R_{n-2,1\,\text{odd}}\\
&\quad - \alpha_{n-1} I_{n-2,1\,\text{odd}}\\
&= I_{n-2,0\,\text{even}} - R_{n-2,1\,\text{odd}} \oplus I_{n-2,1\,\text{odd}}.
\end{aligned}
\tag{18}
$$

The number of operations has been used as the main unit to measure the computational complexity of the proposal. The operation implemented by the weighted primitive has been denoted as weighted sum WS, and the simple sum as SS. The calculations take into account both real and imaginary parts for any u value. The initial two-point transforms are assumed to be calculated. An inductive scheme is used to carry out the complexity estimations.

(i) $N = 4, n = 2, M = 2$

$F(0)$: 1 SS
$F(1)$: $2 \times 3 = 6$ WS
$F(2)$: deduced from $F(0)$, 1 SS
$F(3)$: deduced from $F(1)$, $2 \times 1 = 2$ WS (change of sign)
Overall: **8 WS and 2 SS**.

(ii) $N = 8, n = 3, M = 4$

$F(0)$: 3 SS
$F(1)$, $F(2)$ and $F(3)$ = 14 WS
$F(4)$: 3 SS
$F(5)$, $F(6)$ and $F(7) = 2 \times 3$ = 6 WS (change of sign)
Overall: **20 WS and 6 SS**.

(iii) $N = 16, n = 4, M = 8$

$F(0)$: 7 SS
$F(1), F(2), F(3), \ldots, F(7)$ = 30 WS
$F(8)$: 7 SS
$F(9), \ldots, F(15) = 2 \times 7$ = 14 WS (change of sign)
Overall: 44 WS and 14 SS.

From these results two induced calculation formulas can be proposed referring to the count of needed weighted sums and simple sums,

$$
\begin{aligned}
\text{WS}(n) &= 2 \times \text{WS}(n-1) + 4,\\
\text{SS}(n) &= 2 \times \text{SS}(n-1) + 2.
\end{aligned}
\tag{19}
$$

Proof. Starting from $\text{WS}(1) = 2$ and $\text{SS}(1) = 0$, for any n, $n > 1$, it may be assumed that

$$
\begin{aligned}
\text{WS}(n) &= 2(2n-1) + (2n-2) = 2n + 1 + 2n - 4,\\
\text{SS}(n) &= 2n - 2.
\end{aligned}
\tag{20}
$$

By the application of the inductive scheme, after substituting n by $n + 1$ the formulas become

$$
\begin{aligned}
\text{WS}(n+1) &= 2n + 2 + 2n + 1 - 4,\\
\text{SS}(n+1) &= 2n + 1 - 2.
\end{aligned}
\tag{21}
$$

Comparing the expressions for n and $n + 1$, it can be noticed that

$$
\begin{aligned}
\text{WS}(n+1) &= 2 \times \text{WS}(n) + 4,\\
\text{SS}(n) &= 2 \times \text{SS}(n-1) + 2.
\end{aligned}
\tag{22}
$$

The proposed formulas (see (19)) have been validated by this proof. □

Comparing with the Cooley and Tukey algorithm, where $M(n)$ is the number of multiplications and $S(n)$ the number of sums, we have

$$
\begin{aligned}
M(n+1) &= 2 \times M(n) + 2^n,\\
S(n+1) &= 2 \times S(n) + 2^{n+1}.
\end{aligned}
\tag{23}
$$

The contribution of the weighted primitive is clear as we compare (19) and (23). The quotient $M(n)/\text{WS}(n)$ increases linearly versus n. The same occurs with the quotient $S(n)/\text{SS}(n)$ but with a steeper slope. So, the weighted primitive provides best results as n grows.

5.2. Other transforms

This calculation scheme can be applied to other transforms. As DHT and DCT/DST are DFT-related transforms, a common calculation scheme can be presented after we perform some mathematical manipulations.

Hartley transform

Let $H(u)$ be the discrete Hartley transform of a real function $f(x)$:

$$H(u) = \frac{1}{N}\sum_{x=0}^{N-1} f(x)\left(\cos\frac{2\pi ux}{N} + \sin\frac{2\pi ux}{N}\right),$$

$$\text{where}\quad R(u) = \frac{1}{N}\sum_{x=0}^{N-1} f(x)\cos\frac{2\pi ux}{N}, \tag{24}$$

$$I(u) = \frac{1}{N}\sum_{x=0}^{N-1} f(x)\sin\frac{2\pi ux}{N}.$$

$H(u)$ is the transformed sequence that can split into two fragments: $R(u)$ corresponds to the cosine part and $I(u)$ to the sine part. The whole previous development for the DFT can be applied but the last stage has to perform an additional sum of the two calculated fragments,

$$H(u) = R(u) + I(u). \tag{25}$$

The number of simple sums increases as one last sum must be performed per each u value. Nevertheless, (19) suits because only the initial value varies, $\mathrm{SS}(1) = 2$,

$$\begin{aligned}\mathrm{WS}(n) &= 2\times\mathrm{WS}(n-1)+4,\\ \mathrm{SS}(n) &= 2\times\mathrm{SS}(n-1)+2.\end{aligned} \tag{26}$$

Cosine/sine transforms

Let $C(u)$ be the discrete cosine transform of a real function $f(x)$:

$$C(u) = e(k)\sum_{x=0}^{N-1} f(x)\cos(2x+1)\frac{\pi u}{2N}. \tag{27}$$

$C(u)$ is the transformed sequence that can split into two fragments as follows:

$$\begin{aligned} f(x)\cos(2x+1)\frac{\pi u}{2N} &= f(x)\cos\left(\frac{\pi ux}{N}+\frac{\pi u}{2N}\right)\\ &= f(x)\left(\cos\frac{\pi ux}{N}\cos\frac{\pi u}{2N} - \sin\frac{\pi ux}{N}\sin\frac{\pi u}{2N}\right).\end{aligned} \tag{28}$$

So that (27) leads to (29)

$$C(u) = e(k)\sum_{x=0}^{N-1} f(x)\left(\cos\frac{\pi ux}{N}\cos\frac{\pi u}{2N} - \sin\frac{\pi ux}{N}\sin\frac{\pi u}{2N}\right). \tag{29}$$

Then, $\cos[\pi u/2N]$ and $-\sin[\pi u/2N]$ are constant values for each u value and can lay outside the summation:

$$C(u) = e(k)\left(\alpha_u\sum_{x=0}^{N-1} f(x)\cos\frac{\pi ux}{N} + \beta_u\sum_{x=0}^{N-1} f(x)\sin\frac{\pi ux}{N}\right),$$

$$\text{where}\ \cos\frac{\pi u}{2N} = \alpha_u,\ -\sin\frac{\pi u}{2N} = \beta_u. \tag{30}$$

Both fragments, $R(u)$ (for the cosine part) and $I(u)$ (for the sine part), can be carried out under the DFT calculation scheme and combined in the last stage by an additional weighted sum:

$$C(u) = \alpha_u R(u) + \beta_u I(u). \tag{31}$$

A similar result could be inferred for sine transform with the following parameter values: $\cos(\pi u/2N) = \alpha_u$, $\sin(\pi u/2N) = \beta_u$.

The number of weighted sums increases because of the last weighted sum that must be performed, see (31). The equation has been modified as the constant value in $\mathrm{WS}(n)$ varies. The reason is that the initial value $\mathrm{WS}(1) = 3$,

$$\begin{aligned}\mathrm{WS}(n) &= 2\times\mathrm{WS}(n-1)+3,\\ \mathrm{SS}(n) &= 2\times\mathrm{SS}(n-1)+2.\end{aligned} \tag{32}$$

Summarizing

The calculation based upon the DFT scheme leads to an easy approach for the calculation of the DHT and the DCT/DST, as expected. This scheme can be extended to other integral transforms with trigonometric kernel.

6. COMPARISON WITH OTHER PROPOSALS AND DISCUSSION

In this section, some hardware implementations for the calculation of the DFT, DHT, and DCT are presented in order to provide a comparison for the different performances in terms of area cost, time delay, and stability.

6.1. DFT

The BDA proposal presented by Chien-Chang et al. [36] carries out the DFT of variable length by controlling the architecture. The single processing element follows the Cooley and Tukey algorithm radix-4 and calculates 16/32/64 points transform. When the number of points N grows, it can split out into a product of two factors $N_1 \times N_2$ in order to process the transform in a row-column structure. Formally, the four terms of the butterfly are set as a cyclic convolution that allows performing the calculations by means of distributed arithmetic based on blocks. The memory is partitioned in blocks that store the set of coefficients involved in the multiplications of the butterfly. A rotator is added to control the sequence of use of the blocks and avoids storing all the combinations of the same elements as in conventional distributed arithmetic. This architecture improves memory saving in exchange for increasing the time delay and the hardware because of the extra rotator in the circuit. This proposal substitutes the ROM by a RAM in order to make more flexible the change of the set of coefficients when the length of the Fourier transform varies. The processing column consists of an input buffer, a CORDIC processor that runs the complex multiplications followed by a parallel-serial register and a rotator. Four RAM memories and sixteen accumulators implement the distributed arithmetic. At last, four buffers are

Table 8: Critical path of the basic calculation module in the BDA architecture.

	Preprocessor	*P/S RAM*	*Adder + Acc*	*Post-processor*	*4-point DFT*	*Overall*
Time per column	13.71 ns	12.45 ns	14.06 ns	17.7 ns	10.35 ns	68.27 ns
Critical path	17.7 ns	17.7 ns	17.7 ns	17.7 ns	17.7 ns	88.5 ns

Table 9: Comparison between the hardware needed by BDA and our architecture implementations.

N	*Devices implementing the DBA architecture*	*Devices implementing our proposal*
16	5 buffers, 1 CORDIC processor, P/S-R, 1 rotator, 4 (4×16) bits RAMs, 16 MAC	4 MUX, 4 S-R, 2 (64×16) bits LUTs 4 registers, 4 red-structures 4 adders
64	5 buffers, 1 CORDIC processor, P/S-R, 1 rotator, 4 (16×16) bits RAMs, 16 MAC	
512	9 buffers, 1 CORDIC processor, 2 P/S-R, 1 rotator, 8 (8×16) bits RAMs, 32 MAC 1 transposition memory	
4096	9 buffers, 1 CORDIC processor, 2 P/S-R, 1 rotator, 8 (16×16) bits RAMs, 32 MAC 1 transposition memory	

Table 10: Comparison between the BDA and our architecture implementations in terms of τ_a and τ_t.

N	*BDA architecture*		*Our proposal*	
	Area	*Time delay*	*Area*	*Time delay*
1116	$314\tau_a$	$3.3\,10^3\tau_t$	$336\tau_a$	$1.248\,10^3\tau_t$
1164	$344\tau_a$	$13.2\,10^3\tau_t$		$4.992\,10^3\tau_t$
1512	$632\tau_a$	$105.6\,10^3\tau_t$		$39.936\,10^3\tau_t$
4096	$672\tau_a$	$844.8\,10^3\tau_t$		$119.808\,10^3\tau_t$

needed to reorder the partial products that are involved in the basic four points operation. The number of operations of this proposal is $O((N_1/4M)W_L)$ where N_1 is the length of the transform, $M = 4$ in the design, and W_L is the data length. When the transform is longer as 64 points, N_1 is substituted by the $N_1 \times N_2$. Table 8 shows the results obtained by the synopsis implementation of the circuit that has been described in Verilog HDL.

In order to compare the performance of our architecture and that of the BDA, an estimation of the occupied area and time delay is provided. The devices for both implementations are listed in Table 9 and evaluated in terms of τ_t and τ_a in Table 10. For the crossed evaluation scheme, the architecture is double because of the two segregations (even/odd and real/imaginary); 64 cells LUTs are assumed as the parameter set is $(\alpha, \beta, 1)$. Data is 16 bits long for any proposal. In Table 10, neither the rotator nor the CORDIC processor has been considered in the BDA implementation because the reference does not facilitate any detail upon their structure. The estimations of the time delay are based on the author's indications and presented in terms of τ_a and τ_t units.

It can be observed that the BDA architecture is worse than the crossed one in what concerns the occupied area because the BDA hardware needs to be increased stepwise when the number of points of the transforms increases. The time delay is lower for the crossed architecture than for the BDA for the values of N that have been considered and will remain lower for any N, because it achieves a linear growing in both implementations.

Table 11 summarizes the hardware cost as well as the time delay of proposals for the Fourier transform calculation presented by different authors [13, 37–40]. The four proposals in the beginning of the list have based their design on systolic matrices, the following one on adders and the others on distributed arithmetic (the DA is a generic distributed arithmetic approach). At the end of the list appears our proposal. Average computation time is indicated as

$$\left(\left(\frac{N_1}{4}\right)W_L\right)(T_{\mathrm{ROM}} + 2T_{\mathrm{ADD}} + T_{\mathrm{LATCH}}). \tag{33}$$

It appears that our proposal is the best in what concerns the hardware resources but time delay has a linear growth with respect to N (number of points of the transform) and with the data precision. It can be remembered that parallel architecture may present a better performance for this case.

6.2. DHT

As mentioned in Section 1, the DHT algorithms are typically less efficient (in terms of the number of floating-point

Table 11: Comparison between our proposal and other ones.

	Memory	*Adders*	*Multipliers*	*Shift registers*	*P/S registers*	*CORDIC*	*Average calculation time*
Chang and Chen [37]	0	N	N	$6N$	0	0	$N \times (2T_{\text{mult}} + 2T_{\text{add}} + T_{\text{latch}})$
Fang and Wu [38]	0	$2N + 6$	$N + 4$	$6N$	0	0	$N \times (2T_{\text{mult}} + 2T_{\text{add}} + T_{\text{latch}})$
Murthy and Swamy [39]	0	N	N	$10N$	0	0	$N \times (2T_{\text{mult}} + 2T_{\text{add}} + T_{\text{latch}})$
Chan and Panchanathan [13]	0	N	N	$8N$	0	0	$N \times (2T_{\text{mult}} + 2T_{\text{add}} + T_{\text{latch}})$
Chang et al. [40]	$4N - 4$ (RAM)	$6N + 7$	0	$4N - 2$	0	0	$N/2\times (T_{\text{sum}} + T_{\text{latch}} + T_{\text{add}})$
DA design	$\frac{N}{4}x2$ (ROM)	$\frac{N^2}{4}$	0	$5N$	N	0	$W_L \times (T_{\text{ROM}} + 2T_{\text{add}} + T_{\text{latch}})$
BDA design	$\frac{N}{4}x2$ (ROM)	$\frac{N}{4} + 4$	0	$3N$	$\frac{N}{4}$	$\frac{N}{4} + 4$	$N\times W_L/4\times (T_{\text{ROM}} + 2T_{\text{add}} + T_{\text{latch}})$
Our proposal	$2 \times W_L \times 2^3$ (ROM)	$2 + 2$	0	2	0	0	$(3N/2-2)\times W_L T_{\text{ROM}} + (N-1)W_L \times T_{\text{add}}$

Table 12: Lowest known operation counts (real multiplications + additions) for power-of-two DHT and corresponding DFT algorithms versus our proposal (weighted sums + simple sums).

Size N	*DHT (split-radix FHT)*	*DFT (split-radix FFT)*	*Our proposal*
4	0 + 8 = 8	0 + 6 = 6	8 + 6 = 14
8	2 + 22 = 24	2 + 20 = 22	20 + 14 = 34
16	12 + 64 = 76	10 + 60 = 70	44 + 30 = 74
32	42 + 166 = 208	34 + 164 = 198	92 + 62 = 154
64	124 + 416 = 540	98 + 420 = 518	188 + 126 = 314
128	330 + 998 = 1328	258 + 1028 = 1286	380 + 254 = 634
256	828 + 2336 = 3164	642 + 2436 = 3078	764 + 510 = 1274
512	1994 + 5350 = 7344	1538 + 5636 = 7174	1532 + 1022 = 2554
1024	4668 + 12064 = 16732	3586 + 12804 = 16390	3068 + 2046 = 5114

operations) than the corresponding DFT algorithm specialized for real inputs (or outputs), as proved by Sorensen et al. in 1987 [19]. To illustrate this, Table 12 lists the lowest known operation counts (real multiplications + additions) for the DHT and the DFT for power-of-two sizes, as achieved by the split-radix Cooley-Tukey FHT/FFT algorithm in both cases. Notice that, depending on DFT and DHT implementation details, some of the multiplications can be traded for additions or vice versa. The third column of the table estimates the operation counts (weighted sums + simple sums) to be performed by our proposal, following (19).

As expected, our proposal behaves better in what concerns the operation counts than both the DHT algorithm and the corresponding DFT algorithm specialized for real inputs or outputs. With respect to the particular hardware implementations, as the DFT has already been compared above with our proposal, the concluding remarks related to the DHT have to be deduced.

A detailed analysis of the computational cost and especially of the numerical stability constants for DHT is presented by Arico et al. in [23]. The authors base their research on the close connection existing between fast DHT algorithms and factorizations of the corresponding orthogonal Hartley matrices of length N, H_N. They achieve a factorization of the matrix H_N into a product of sparse matrices (at most, two nonzero entries per row and column) that allows an iterative calculation of $H_N\mathbf{x}$, for any $\mathbf{x} \in \mathbb{R}^N$. Since the matrices are sparse and orthogonal, the factorization of H_N generates a fast and low arithmetic cost DHT algorithms. The intraconnection of Hartley matrices of types (II), (III), and (IV) is expressed by means of other Hartley matrix of type (I), $H_N(\text{I})$, is pursued by means of twiddle matrices T_N and T'_N (that are direct sums of 1 and of rotation-reflection matrices of order 2). Finally, factorization of $H_N(\text{I})$ is achieved requiring permutations, scaling operations, butterfly operations, and plane rotations with small angles.

Table 13: Normwise forward stability of DHT-I (N) for 16, 32, and 64 bits data.

N	$\log_2(N)$	$u = 2^{-16}$	$u = 2^{-32}$	$u = 2^{-64}$
16	4	$13.292163u = 2^{3.74}2^{-16} = 2^{-19.74}$	$13.292163u = 2^{3.74}2^{-32} = 2^{-35.74}$	$13.292163u = 2^{3.74}2^{-64} = 2^{-67.74}$
32	5	$17.722908u = 2^{4.16}2^{-16} = 2^{-20.16}$	$17.722908u = 2^{4.16}2^{-32} = 2^{-36.16}$	$17.722908u = 2^{4.16}2^{-64} = 2^{-68.16}$
64	6	$22.153605u = 2^{4.48}2^{-16} = 2^{-20.48}$	$22.153605u = 2^{4.48}2^{-32} = 2^{-36.48}$	$22.153605u = 2^{4.48}2^{-64} = 2^{-68.48}$
128	7	$2^{4.75}2^{-16} = 2^{-20.75}$	$2^{4.75}2^{-32} = 2^{-36.75}$	$2^{4.75}2^{-64} = 2^{-68.75}$
256	8	$2^{4.97}2^{-16} = 2^{-20.97}$	$2^{4.97}2^{-32} = 2^{-36.97}$	$2^{4.97}2^{-64} = 2^{-68.97}$
512	9	$2^{5.16}2^{-16} = 2^{-21.16}$	$2^{5.16}2^{-32} = 2^{-37.16}$	$2^{5.16}2^{-64} = 2^{-69.16}$
1024	10	$2^{5.33}2^{-16} = 2^{-21.33}$	$2^{5.33}2^{-32} = 2^{-37.33}$	$2^{5.33}2^{-64} = 2^{-69.33}$
2048	11	$2^{5.49}2^{-16} = 2^{-21.49}$	$2^{5.49}2^{-32} = 2^{-37.49}$	$2^{5.49}2^{-64} = 2^{-69.49}$
4096	12	$2^{5.63}2^{-16} = 2^{-21.63}$	$2^{5.63}2^{-32} = 2^{-37.63}$	$2^{5.63}2^{-64} = 2^{-69.63}$
8192	13	$2^{5.75}2^{-16} = 2^{-21.75}$	$2^{5.75}2^{-32} = 2^{-37.75}$	$2^{5.75}2^{-64} = 2^{-69.75}$
16384	14	$2^{5.87}2^{-16} = 2^{-21.87}$	$2^{5.87}2^{-32} = 2^{-37.87}$	$2^{5.87}2^{-64} = 2^{-69.87}$

The computational complexity is calculated for all types DHT-X, X = I, II, III, and IV but for comparison with our results we will consider the best result which is for X = I.

The number of additions is denoted by $\alpha(\text{DHT-I}, N)$ and the number of multiplications by $\mu(\text{DHT-I}, N)$:

$$\begin{aligned}(\text{DHT-I}, N) &= \frac{3}{2N}\log_2(N) - \frac{3}{2N} + 2,\\ \mu(\text{DHT-I}, N) &= N\log_2(N) - 3N + 4.\end{aligned} \tag{34}$$

As seen in the paper, the operation error follows the IEEE precision arithmetic $u = 2^{-24}$ or $u = 2^{-53}$ depending on the precision of the mantissa (24 or 53 bits, resp.). The round-off algorithmic errors are related to the structure of the involved matrices and for direct calculation the round-off error is evaluated as a squared distance bounded by an expression $\approx k_N u$. The numerical stability is measured by k_N that can be understood as the relative error on the output vector (of the mapping previously defined). For any X, a different expression for k_N is obtained for the corresponding DHT-X(N). All k_N expressions are similar and have linear dependence of $\log_2 N$. For example, the normwise forward stability for

$$\text{DHT-I}(N) \text{ is } \left(\left(\frac{4}{3}\sqrt{3} + \frac{3}{2}\sqrt{2}\right)(\log_2 N - 1) + O(u)\right)u. \tag{35}$$

As far as we can compare this very deep and strong theoretical approach with our method that is rather empirical, the results that can be taken into account are the computational cost and the stability of the algorithms. To make easier the comparison with our paper in what concerns the number of operations to be performed, a recursive formulation of (DHT-I, N) and $\mu(\text{DHT-I}, N) =$ for $N = 2^n$ has been deduced from (34):

$$\begin{aligned}\alpha(n) &= 2\alpha(n-1) + 3.2^n - 2,\\ \mu(n) &= 2\mu(n-1) + 2^n - 4.\end{aligned} \tag{36}$$

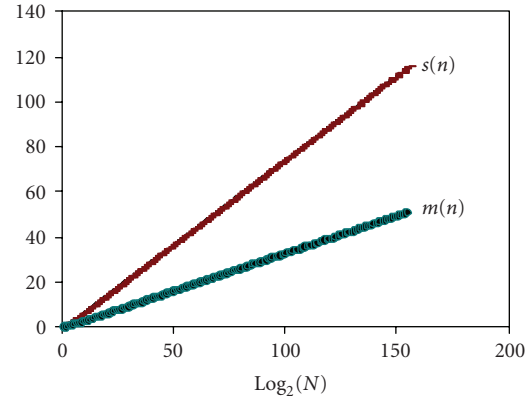


Figure 3: Growing rates $s(n)$ and $m(n)$ versus n.

Table 14: Number of multiplication and addition operations for different 4×4 DCTs.

Operation	*Fast algorithms*			*Our proposal*
	[48]	[49]	[47]	
Multiplication	512	256	172	45
Addition	496	480	963	14

The initial values are for n =1, following (34):

$$\begin{aligned}\alpha(1) &= \frac{3}{2}\cdot 2.1 - \frac{3}{2}\cdot 2 + 2 = 2,\\ \mu(1) &= 2\cdot 1 - 3 + 4 = 3.\end{aligned} \tag{37}$$

The comparison between (19) and (36) (WS(n) versus $\mu(n)$ and SS(n) versus $\alpha(n)$) outlines that $\alpha(n)$ and $\mu(n)$ increase at a higher speed than WS(n) and SS(n), respectively,

(i) for all n, $\alpha(n) > \text{SS}(n)$,
(ii) for $n > 6$, $\mu(n) > \text{WS}(n)$.

Figure 3 represents the growing rates $s(n) = \alpha(n)/\text{SS}(n)$ and $m(n) = \mu(n)/\text{WS}(n)$ versus n.

The value of the normwise forward stability in the case of DHT-I (N) is $(((4/3)\sqrt{3} + (3/2)\sqrt{2})(\log_2 N - 1) + O(u))u = 4.430721(\log_2 N - 1)u$. In order to compare with our results

Table 15: Number of recursive cycles for different $N \times N$ DCT recursive structures.

$N \times N$		*Row-column method with transposition memory*				*Recursive algorithm*	*Our proposal*
		[42]	[43]	[45]	[46]	[47]	
Power of two	8×8	1024	1024	800	256	220	189
	16×16	8192	8192	5952	2048	1756	765
	32×32	65526	65536	45696	16384	14044	3069
	64×64	524288	524288	357632	131073	112348	12285
	128×128	4194304	4194304	2828800	1948567	898780	49149
Number of recursive kernels		1	1	1	2	2	0
Size of transposition memory		$O(N^2)$	$O(N^2)$	$O(N^2)$	$O(N^2)$	0	0

Table 16: Comparison between the hardware needed by the recursive architecture versus that of the implementation of our proposal for 4×4 DCT transform.

$N \times N$	Devices implementing the recursive architecture	Devices implementing our proposal
4×4	$1 \times$ Data memory buffer, $2 \times$ adders $2 \times$ 1–4 DEMUX $1 \times$ CMP $1 \times$ Condensed counter ($2 \times$ ripple connected mod-4 counters) $1 \times$ Condensed index generator (2 S-R, 2 shifters, 3 adders) $2 \times$ Recursive input buffer $2 \times$ 1D DCT/DST IIR	$4 \times$ MUX, 4 S-R, $2 \times (64 \times 16)$ bits LUTs $4 \times$ registers, $4 \times$ reduction structures $4 \times$ adders

of Table 7, the previous formula has been calculated for the cases $u = 2^{-16}$, 2^{-32}, and 2^{-64} bits and for different values of N.

The comparison between Tables 7 and 13 shows that for 16 bits (fragmentation lengths $k = 2$ and $k = 4$), for 32 bits data ($k = 2$, 4, and 8) and for 64 bits data ($k = 2$, 4, and 8) our algorithm behaves better.

6.3. DCT

The search for recursive algorithms with regular structure and less computation time remains an active research area. The recursive algorithms for computing 1D DCT are highly regular and modular [41–47]. However, a great number of cycles are required to compute the 2D transformation by using 1D recursive structures. For computing the 2D DCT by row-column approaches, the row (column) transforms of the input 2D data are first determined. A transposition memory is required to store those temporal results. Finally, the 2D DCT results are obtained by the column (row) transforms of the transposed data. The RAM is usually adopted as the transposition memory. This approach has disadvantages such as higher-power consumption and long access time. Chen et al. develop in 2004 a new recursive structure with fast and regular recursion to achieve fewer recursive cycles without using any transposition memory structure [48]. The 2D recursive DCT/IDCT algorithms are developed considering that the data with the same transform base can be pre-added such that the recursive cycles can be reduced. First, the 2D DCT/IDCT is decomposed into four portions which can be carried out either by 1D DCT or 1D DST (discrete sine transform). Based on the use of Chebyshev polynomials, efficient transform kernels are obtained for the 1D DCT and the DST. A reduction on the number or recursive cycles is achieved by a further folding on the inputs of the transform kernels. Considering other fast algorithms, the $N \times N$ DCT which maps the 2D index of the input sequence into the new 1D index is decomposed into N length-N 1D DCTs [49, 50]. Table 14 presents the number of multiplication and addition operations for these fast algorithms, for the case of 4×4 DCTs. Our proposal can be compared by assimilating the weighted sums and the multiplications (see (32)).

The number of operations required for our proposal is lower than those required for the existing methods. Table 15 shows the number of recursive cycles for different $N \times N$ DCT recursive structures in five different algorithms [43, 44, 46–48]. In [48], a recursive cycle represents the time delay needed for computing the 2D DCT cosine transform for a pair of frequency indexes. The circuit involves two parallel identical block diagrams, both with a condensed 1D DCT/DST IIR filter which obtains the corresponding input data from a recursive input buffer in order to perform the partial calculation of the transform. In the last stage, the transform is recombined by a sum of the two partial results.

So, the overall time delay for the 2D may be the same as for the 1D and the comparison with our proposal can be done as we assimilate the number of recursive cycles with the number of weighted sums to be performed following (32).

It can be outlined that our proposal has a better performance than the other ones, namely fast and recursive algorithms, in what concerns the number of recursive cycles. In [48] the chip area can be estimated as we depict the hardware recursive circuitry. Table 16 summarizes the hardware devices of the recursive architecture compared with our proposal for 4×4 DCT transform.

It can be observed that the devices for the implementation of the recursive architecture are numerous. Therefore, greater values for $N \times N$ may imply an increase of the chip area; the reason is the growth of the storing memory required for the buffers and for the number of outputs of the demultiplexer. Reference [48] does not offer any estimation of the time delay of the calculation. Our proposal implementation is very simple and has no variation related to the amount of devices when the number of calculated values varies. With respect to the time delay of the calculation in [48], as far as we can suppose, it can be estimated by analyzing the critical path of the depicted circuit. It seems to be higher than our proposal's one.

7. CONCLUSIONS

This paper has presented an approach to the scalability problem caused by the exploding requirements of computing resources in function calculation methods. The fundamentals of our proposal claim that the use of a more complete primitive, namely a weighted sum, converts the calculation of the function values into a recursive operation defined by a two-input table. The strength of the method is concerned with the fact that the operation to be performed is the same for the evaluation of different functions (elementary or not). Therefore, only the table must be changed because it holds the features of the concrete evaluated function in the parameter values. This method provides a linear computational cost when some conditions are fulfilled. Image processing transforms that involve combined trigonometric functions provide an interesting application field. A generic calculation scheme has been developed for the DFT as paradigm. Other image transforms namely the DHT and the DCT/DST are analyzed under the scope of the DFT. When comparing with other well-known proposals, it has been confirmed that our approach provides a good trade-off between hardware resource and time delay saving as well as encouraging partial results in what concerns error contention.

REFERENCES

[1] R. Chamberlain, E. Lord, and D. J. Shand, "Real-time 2D floating-point fast Fourier transforms for seeker simulation," in *Technologies for Synthetic Environments: Hardware-in-the-Loop Testing VII*, R. L. Murrer Jr., Ed., vol. 4717 of *Proceedings of SPIE*, pp. 15–23, Orlando, Fla, USA, July 2002.

[2] P. Yan, Y. L. Mo, and H. Liu, "Image restoration based on the discrete fraction Fourier transform," in *Image Matching and Analysis*, B. Bhanu, J. Shen, and T. Zhang, Eds., vol. 4552 of *Proceedings of SPIE*, pp. 280–285, Wuhan, China, September 2001.

[3] W. A. Rabadi, H. R. Myler, and A. R. Weeks, "Iterative multiresolution algorithm for image reconstruction from the magnitude of its Fourier transform," *Optical Engineering*, vol. 35, no. 4, pp. 1015–1024, 1996.

[4] C.-H. Chang, C.-L. Wang, and Y.-T. Chang, "Efficient VLSI architectures for fast computation of the discrete Fourier transform and its inverse," *IEEE Transactions on Signal Processing*, vol. 48, no. 11, pp. 3206–3216, 2000.

[5] S.-F. Hsiao and W.-R. Shiue, "Design of low-cost and high-throughput linear arrays for DFT computations: algorithms, architectures, and implementations," *IEEE Transactions on Circuits and Systems II*, vol. 47, no. 11, pp. 1188–1203, 2000.

[6] J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex Fourier series," *Mathematics of Computation*, vol. 19, no. 90, pp. 297–301, 1965.

[7] P. N. Swarztrauber, "Multiprocessor FFTs," *Parallel Computing*, vol. 5, no. 1-2, pp. 197–210, 1987.

[8] C. Temperton, "Self-sorting in-place fast Fourier transforms," *SIAM Journal on Scientific and Statistical Computing*, vol. 12, no. 4, pp. 808–823, 1991.

[9] M. C. Pease, "An adaptation of the fast Fourier transform for parallel processing," *Journal of the ACM*, vol. 15, no. 2, pp. 252–264, 1968.

[10] L. L. Hope, "A fast Gaussian method for Fourier transform evaluation," *Proceedings of the IEEE*, vol. 63, no. 9, pp. 1353–1354, 1975.

[11] C.-L. Wang and C.-H. Chang, "A DHT-based FFT/IFFT processor for VDSL transceivers," in *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP '01)*, vol. 2, pp. 1213–1216, Salt Lake, Utah, USA, May 2001.

[12] W.-H. Fang and M.-L. Wu, "An efficient unified systolic architecture for the computation of discrete trigonometric transforms," in *Proceedings of IEEE International Symposium on Circuits and Systems (ISCAS '97)*, vol. 3, pp. 2092–2095, Hong Kong, June 1997.

[13] E. Chan and S. Panchanathan, "A VLSI architecture for DFT," in *Proceedings of the 36th Midwest Symposium on Circuits and Systems*, vol. 1, pp. 292–295, Detroit, Mich, USA, August 1993.

[14] R. V. L. Hartley, "A more symmetrical Fourier analysis applied to transmission problems," *Proceedings of the IRE*, vol. 30, no. 3, pp. 144–150, 1942.

[15] R. N. Bracewell, "Discrete Hartley transform," *Journal of the Optical Society of America*, vol. 73, no. 12, pp. 1832–1835, 1983.

[16] R. N. Bracewell, "The fast Hartley transform," *Proceedings of the IEEE*, vol. 72, no. 8, pp. 1010–1018, 1984.

[17] R. N. Bracewell, *The Hartley Transform*, Oxford University Press, New York, NY, USA, 1986.

[18] R. N. Bracewell, "Computing with the Hartley transform," *Computers in Physics*, vol. 9, no. 4, pp. 373–379, 1995.

[19] H. V. Sorensen, D. L. Jones, M. T. Heideman, and C. S. Burrus, "Real-valued fast Fourier transfer algorithms," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 35, no. 6, pp. 849–863, 1987.

[20] P. Duhamel and M. Vetterli, "Improved Fourier and Hartley transform algorithms: application to cyclic convolution of real data," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 35, no. 6, pp. 818–824, 1987.

[21] M. Popović and D. Šević, "A new look at the comparison of the fast Hartley and Fourier transforms," *IEEE Transactions on Signal Processing*, vol. 42, no. 8, pp. 2178–2182, 1994.

[22] M. Frigo and S. G. Johnson, "The design and implementation of FFTW3," *Proceedings of the IEEE*, vol. 93, no. 2, pp. 216–231, 2005.

[23] A. Arico, S. Serra-Capizzano, and M. Tasche, "Fast and numerically stable algorithms for discrete Hartley transforms and applications to preconditioning," *Communications in Information Systems*, vol. 5, no. 1, pp. 21–68, 2005.

[24] K. R. Rao and P. Yip, *Discrete Cosine Transform: Algorithms, Advantages, Applications*, Academic Press, Boston, Mass, USA, 1990.

[25] S. A. Martucci, "Symmetric convolution and the discrete sine and cosine transforms," *IEEE Transactions on Signal Processing*, vol. 42, no. 5, pp. 1038–1051, 1994.

[26] W. B. Pennebaker and J. L. Mitchell, *JPEG Still Image Data Compression Standard*, Van Nostrand Reinhold, New York, NY, USA, 1993.

[27] Y. Q. Shi and H. Sun, *Image and Video Compression for Multimedia Engineering*, CRC Press, Boca Raton, Fla, USA, 2000.

[28] P. Duhamel and C. Guillemot, "Polynomial transform computation of the 2-D DCT," in *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP '90)*, vol. 3, pp. 1515–1518, Albuquerque, NM, USA, April 1990.

[29] E. Feig and S. Winograd, "Fast algorithms for the discrete cosine transform," *IEEE Transactions on Signal Processing*, vol. 40, no. 9, pp. 2174–2193, 1992.

[30] A. C. Hung and T. H.-Y. Meng, "A comparison of fast inverse discrete cosine transform algorithms," *Multimedia Systems*, vol. 2, no. 5, pp. 204–217, 1994.

[31] P. Duhamel and M. Vetterli, "Fast Fourier transforms: a tutorial review and a state of the art," *Signal Processing*, vol. 19, no. 4, pp. 259–299, 1990.

[32] S. Serra-Capizzano, "A note on antireflective boundary conditions and fast deblurring models," *SIAM Journal on Scientific Computing*, vol. 25, no. 4, pp. 1307–1325, 2003.

[33] M. Ercegovac and T. Lang, *Division and Square Root: Digit-Recurrence, Algorithms and Implementations*, Klüwer Academic Publishers, Boston, Mass, USA, 1994.

[34] J.-A. Piñeiro, M. D. Ercegovac, and J. D. Bruguera, "High-radix logarithm with selection by rounding," in *Proceedings of the 13th IEEE International Conference on Application-Specific Systems, Architectures and Processors (ASAP '02)*, pp. 101–110, San Jose, Calif, USA, July 2002.

[35] J. M. García Chamizo, M. T. Signes Pont, H. Mora Mora, and G. de Miguel Casado, "Parametrizable architecture for function recursive evaluation," in *Proceedings of the 18th Conference on Design of Circuits and Integrated Systems (DCIS '03)*, Ciudad Real, Spain, November 2003.

[36] L. Chien-Chang, Ch. Chih-Da, and J. I. Guo, "A parameterized hardware design for the variable length discrete Fourier transform," in *Proceedings of the 15th International Conference on VLSI Design (VLSID '02)*, Taiwan, China, August 2002.

[37] L. W. Chang and M. Y. Chen, "A new systolic array for discrete Fourier transform," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 36, no. 10, pp. 1665–1666, 1988.

[38] W.-H. Fang and M.-L. Wu, "An efficient unified systolic architecture for the computation of discrete trigonometric transforms," in *Proceedings of IEEE International Symposium on Circuits and Systems (ISCAS '97)*, vol. 3, pp. 2092–2095, Hong Kong, June 1997.

[39] N. R. Murthy and M. N. S. Swamy, "On the real-time computation of DFT and DCT through systolic architectures," *IEEE Transactions on Signal Processing*, vol. 42, no. 4, pp. 988–991, 1994.

[40] T.-S. Chang, J.-I. Guo, and C.-W. Jen, "Hardware-efficient DFT designs with cyclic convolution and subexpression sharing," *IEEE Transactions on Circuits and Systems II*, vol. 47, no. 9, pp. 886–892, 2000.

[41] V. Kober and G. Cristobal, "Fast recursive algorithms for short-time discrete cosine transform," *Electronics Letters*, vol. 35, no. 15, pp. 1236–1238, 1999.

[42] L.-P. Chau and W.-C. Siu, "Recursive algorithm for the discrete cosine transform with general lengths," *Electronics Letters*, vol. 30, no. 3, pp. 197–198, 1994.

[43] Z. Wang, G. A. Jullien, and W. C. Miller, "Recursive algorithms for the forward and inverse discrete cosine transform with arbitrary length," *IEEE Signal Processing Letters*, vol. 1, no. 7, pp. 101–102, 1994.

[44] M. F. Aburdene, J. Zheng, and R. J. Kosick, "Computation of discrete cosine transform using Clenshaw's recurrence formula," *IEEE Signal Processing Letters*, vol. 2, no. 8, pp. 155–156, 1995.

[45] Y.-H. Chan, L.-P. Chau, and W.-C. Siu, "Efficient implementation of discrete cosine transform using recursive filter structure," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 4, no. 6, pp. 550–552, 1994.

[46] J.-F. Yang and C.-P. Fan, "Compact recursive structures for discrete cosine transform," *IEEE Transactions on Circuits and Systems II*, vol. 47, no. 4, pp. 314–321, 2000.

[47] J. L. Wang, C. B. Wu, D.-B. Liu, and J.-F. Yang, "Recursive architecture for realizing modified discrete cosine transform and its inverse," in *Proceedings of IEEE Workshop on Signal Processing Systems (SIPS '99)*, pp. 120–130, Taipei, Taiwan, October 1999.

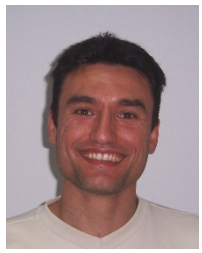
[48] C.-H. Chen, B.-D. Liu, and J.-F. Yang, "Direct recursive structures for computing radix-r two-dimensional DCT/IDCT/DST/IDST," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 51, no. 10, pp. 2017–2030, 2004.

[49] N. I. Cho and S. U. Lee, "A fast 4×4 DCT algorithm for the recursive 2-D DCT," *IEEE Transactions on Signal Processing*, vol. 40, no. 9, pp. 2166–2173, 1992.

[50] N. I. Cho and S. U. Lee, "Fast algorithm and implementation of 2-D discrete cosine transform," *IEEE Transactions on Circuits and Systems*, vol. 38, no. 3, pp. 297–305, 1991.

María Teresa Signes Pont received the B.S. degree in computer science from the Institut National des Sciences Appliquées de Toulouse (France) and in Physics Universidad Nacional de Educación a Distancia (Spain) in 1978 and 1987, respectively. She received the Ph.D. degree in computer science from the University of Alicante in 2005. Since 1996, she is a member of the Computer Technology and Computation Department at the same university where she is currently an Associate Professor and Researcher of Specialized Processors Architecture Laboratory. Her areas of research interest include computer arithmetic, computational biology and the design of floating points units, and approximation algorithms related to VLSI design.

Juan Manuel García Chamizo received his B.S. degree in physics at the University of Granada (Spain) in 1980, and the Ph.D. degree in computer science at the University of Alicante (Spain) in 1994. He is currently a Full Professor and Director of the Computer Technology and Computation Department at the University of Alicante. His current research interests are computer vision, reconfigurable hardware, biomedical applications, computer networks and architectures, and artificial neural networks. He has directed several research projects related to the above-mentioned interest areas. He is a member of a Spanish Consulting Commission on Electronics, Computer Science, and Communications. He is also member and editor of some program committee conferences.



Higinio Mora Mora received the B.S. degree in computer science engineering and the B.S. degree in business studies from University of Alicante, Spain, in 1996 and 1997, respectively. He received the Ph.D. degree in computer science from the University of Alicante in 2003. Since 2002, he is a member of the Computer Technology and Computation Department at the same university where he is currently an Associate Professor and Researcher of Specialized Processors Architecture Laboratory. His areas of research interest include computer arithmetic, the design of floating points units, and approximation algorithms related to VLSI design.

Gregorio de Miguel Casado received the B.S. degree in computer science engineering and a master degree in business administration from the University of Alicante, Spain, in 2001 and 2003, respectively. Since 2001, he is a member of the research group I2RC of the Computer Technology and Computation Department at the same university where he is currently a Researcher of the Specialized Processors Architecture Laboratory. His areas of research interest include formal VLSI design methods, computable analysis, and computer arithmetic for the development of arithmetic operators for scientific computing.