Hindawi Publishing Corporation
EURASIP Journal on Advances in Signal Processing
Volume 2011, Article ID 973806, 20 pages
doi:10.1155/2011/973806

Research Article

Evolutionary Approach to Improve Wavelet Transforms for Image Compression in Embedded Systems

Rubén Salvador,[1] Félix Moreno,[1] Teresa Riesgo,[1] and Lukáš Sekanina[2]

[1] *Centre of Industrial Electronics, Universidad Politécnica de Madrid, José Gutierrez Abascal 2, 28006 Madrid, Spain*
[2] *Faculty of Information Technology, Brno University of Technology, Bozetechova 2, 612 66 Brno, Czech Republic*

Correspondence should be addressed to Rubén Salvador, ruben.salvador@upm.es

Received 21 July 2010; Revised 19 October 2010; Accepted 30 November 2010

Academic Editor: Yannis Kopsinis

A bioinspired, evolutionary algorithm for optimizing wavelet transforms oriented to improve image compression in embedded systems is proposed, modelled, and validated here. A simplified version of an Evolution Strategy, using fixed point arithmetic and a hardware-friendly mutation operator, has been chosen as the search algorithm. Several cutdowns on the computing requirements have been done to the original algorithm, adapting it for an FPGA implementation. The work presented in this paper describes the algorithm as well as the test strategy developed to validate it, showing several results in the effort to find a suitable set of parameters that assure the success in the evolutionary search. The results show how high-quality transforms are evolved from scratch with limited precision arithmetic and a simplified algorithm. Since the intended deployment platform is an FPGA, HW/SW partitioning issues are also considered as well as code profiling accomplished to validate the proposal, showing some preliminary results of the proposed hardware architecture.

1. Introduction

Wavelet Transform (WT) brought a new way to look into a signal, allowing for a joint time-frequency analysis of information. Initially defined and applied through the Fourier Transform and computed with the subband filtering scheme, known as Fast Wavelet Transform (FWT), the Discrete Wavelet Transform (DWT) widened its possibilities with the proposal of the *Lifting Scheme* (LS) by Sweldens [1]. Custom construction of wavelets was made possible with this computation scheme.

Adaptation capabilities are increasingly being brought to embedded systems, and image processing is, by no means, the exception to the rule. Compression standard JPEG2000 [2] relies on wavelets for its transform stage. It is a very useful tool for (adaptive) image compression algorithms, since it provides a transform framework that can be adapted to the type of images being handled. This feature allows it to improve the performance of the transform according to each particular type of image so that improved compression (in terms of quality versus size) can be achieved, depending on the wavelet used.

Having a system able to adapt its compression performance, according to the type of images being handled, may help in, for example, the calibration of image processing systems. Such a system would be able to self-calibrate when it is deployed in different environments (even to adapt through its operational life) and has to deal with different types of images. Certain tunings to the transform coefficients may help in increasing the quality of the transform and, consequently, the quality of the compression.

This paper deals with the implementation of adaptive wavelet transforms in FPGA devices. The various approaches previously followed by other authors in the search for this transform adaptivity will be analysed. Most of these are based on the mathematical foundations of wavelets and multiresolution analysis (MRA). The knowledge domain of the authors of this paper does not lie within this theoretical point of view, but, in contrast, the author's team is composed of electronic engineers and Evolutionary Computation (EC)

experts. Therefore, what is being proposed here is the use of bio-inspired algorithms, such as Evolutionary Algorithms (EAs), as a design/optimization tool to help find new wavelet filters adapted to specific kind of images. For this reason, it is the whole system that is being adapted. No extra computing effort is added in the transform algorithm, such as what classical *adaptive lifting* techniques propose. In contrast, we are proposing *new* ways to design completely new wavelet filters.

The choice of an FPGA as the computing device for the embedded system comes from the restrictions imposed by the *embedded system* itself. The suitability of FPGAs for high-performance computing systems is nowadays generally accepted due to their inherent massive parallel processing capabilities. This reasoning can be extended to embedded vision systems as shown in [3]. Alternative processing devices like Graphics Processing Units (GPUs) have a comparable degree of parallelism producing similar throughput figures depending on the application at hand, but their power demands are too high for portable/mobile devices [4–7].

Therefore, the scope of this paper is directed at a generic artificial vision (embedded) system to be deployed in an unknown environment during design time, letting the calibration phase adjust the system parameters so that it performs efficient signal (image) compression. This allows the system to efficiently deal with images coming from very diverse sources such as visual inspections of a manufacturing line, a portable biometric data compression/analysis system, a terrestrial satellite image, and. Besides, the proposed algorithm will be mapped to an FPGA device, as opposed to other proposals, where these algorithms need to run on supercomputing machines or, at least, need such a computing power that makes them unfeasible for an implementation as an embedded *real-time* system.

The remainder of this paper is structured as follows. Sections 2 and 3 show a short introduction to WT and EAs. After an analysis of previously published works in Section 4, the proposed method is presented in Section 5. Obtained results are shown and discussed in Section 6, validating the proposed algorithm. Section 7 analyses the implementation in an FPGA device, together with the proposed architecture able to host this system and the preliminary results obtained. The paper is concluded in Section 8, featuring a short discussion and commenting on future work to be accomplished.

2. Overview of the Wavelet Transform

The DWT is a multiresolution analysis (MRA) tool widely used in signal processing for the analysis of the frequency content of a signal at different resolutions.

It concentrates the signal energy into fewer coefficients to increase the degree of compression when the data is encoded. The energy of the input signal is redistributed into a low-resolution trend subsignal (scaling coefficients) and high-resolution subsignals (wavelet coefficients; horizontal, vertical, and diagonal subsignals for image transforms). If the wavelet chosen for the transform is suited for the type of image being analysed, most of the information of the signal will be kept in the trend subsignal, while the wavelet coefficients (high-frequency details) will have a very low value. For this reason, the DWT can reduce the number of bits required to represent the input data.

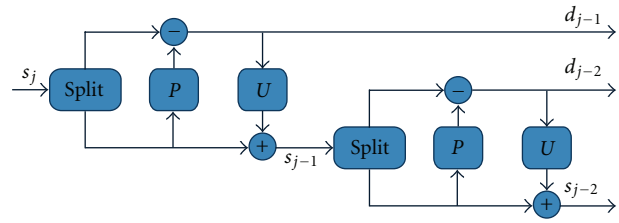

Figure 1: Lifting scheme.

For a general introduction to wavelet-based multiresolution analysis check [8], the *Fast Wavelet Transform* (FWT) algorithm computes the wavelet representation via a *subband filtering* scheme which recursively filters the input data with a pair of high-pass and low-pass digital filters, downsampling the results by a factor of two [9]. A widely known set of filters that build up the standard D9/7 wavelet (used in JPEG2000 for lossy compression) gets its name because its high-pass and low-pass filters have 9 and 7 coefficients, respectively.

The FWT algorithm was improved by the *Lifting Scheme* (LS), introduced by Sweldens [1], which reduces the computational cost of the transform. It does not rely on the Fourier Transform for its definition and application and has given rise to the so-called *Second Generation Wavelets* [10]. Besides, the research effort put on the LS has simplified the construction of custom wavelets adapted to specific and different types of data.

The basic LS, shown in Figure 1, consists of three stages: "*Split*", "*Predict*", and "*Update*", which try to exploit the correlation of the input data to obtain a more compact representation of the signal [11].

The *Split* stage divides the input data into two smaller subsets, s_{j-1} and d_{j-1}, which usually correspond with the even and odd samples. It is also called the *Lazy Wavelet*.

To obtain a more compact representation of the input data, the s_{j-1} subset is used to *predict* the d_{j-1} subset, called the wavelet subset, which is based on the correlation of the original data. The difference between the prediction and the actual samples is stored, also as d_{j-1}, overwriting its original value. If the prediction operator P is reasonably well designed, the difference will be very close to 0, so that the two subsets s_{j-1} and d_{j-1} produce a more compact representation of the original data set s_j.

In most cases, it is interesting to maintain some properties of the original signal after the transform, such as the mean value. For this reason, the LS proposes a third stage that not only reuses the computations already done in the previous stages but also defines an easily invertible scheme. This is accomplished by *updating* the s_{j-1} subset with the already computed wavelet set d_{j-1}. The wavelet representation of s_j is therefore given by the set of coefficients $\{s_{j-2}, d_{j-2}, d_{j-1}\}$.

This scheme can be iterated up to n levels, so that an original input data set s_0 will have been replaced with

the wavelet representation $\{s_{-n}, d_{-n}, \ldots, d_{-1}\}$. Therefore, the algorithm for the LS implementation is as follows

for $j \leftarrow 1, n$ **do**
  $\{s_j, d_j\} \leftarrow \text{Split}(s_{j+1})$
  $d_j = d_j - P(s_j)$
  $s_j = s_j + U(d_j)$
end for

where j stands for the decomposition level. There exists a different notation for the transform coefficients $\{s_{j-i}, d_{j-i}\}$; for a 2-level image decomposition, it can be expressed as $\{LL, LH, HL, HH\}$, where L stands for low-pass and H for high-pass coefficients, respectively.

3. Optimization Techniques Based on Bioinspired, Evolutionary Approaches

Evolutionary Computation (EC) [12] is a subfield of Artificial Intelligence (AI) that consists of a series of biologically inspired search and optimization algorithms that evolve iteratively better and better solutions. It involves techniques inspired by biological evolution mechanisms such as reproduction, mutation, recombination, natural selection, and survival of the fittest.

An Evolution Strategy (ES) [13] is one of the fundamental algorithms among Evolutionary Algorithms (EAs) that utilize a population of candidate solutions and bio-inspired operators to search for a target solution. ESs are primarily used for optimization of real-valued vectors. The algorithm operators are iteratively applied within a loop, where each run is called a *generation* (g), until a termination criterion is met. Variation is accomplished by the so-called *mutation* operator. For real-valued search spaces, mutation is normally performed by adding a normally (Gaussian) distributed random value to each component under variation (i.e., to each parameter encoded in the individuals). Algorithm 1 shows a pseudocode description of a typical ES.

One of the particular features of ESs is that the individual step sizes of the variation operator for each coordinate (or correlations between coordinates) is governed by self-adaptation (or by covariance matrix adaptation (CMA-ES) [14]). This self-adaptation of the step size σ, also known as *mutation strength* (i.e., standard deviation of the normal distribution), implies that σ is also included in the chromosomes, undergoing variation and selection itself (coevolving along with the solutions).

The canonical versions of the ES are denoted by $(\mu/\rho, \lambda)$-ES and $(\mu/\rho + \lambda)$-ES, where μ denotes the number of parents (parent population, P_μ), $\rho \leq \mu$ the mixing number (i.e., the number of parents involved in the procreation of an offspring), and λ the number of offspring (offspring population, P_λ). The parents are *deterministically selected* from the set of either the offspring, referred to as *comma selection* ($\mu < \lambda$), or both the parents and offspring, referred to as *plus selection*. This selection is based on the ranking of the individuals' fitness ($\mathcal{F}$) choosing the μ best individuals out of the whole pool of candidates. Once selected, ρ out of the μ parents ($\mathcal{R}$) are *recombined* to produce an offspring individual ($\mathbf{r}_l$) using *intermediate recombination*, where the parameters of the selected parents are averaged or randomly chosen if *discrete recombination* is used. Each ES individual $\mathbf{a} := (\mathbf{y}, \mathbf{s})$ comprises the *object parameter vector* $\mathbf{y}$ to be optimized and a set of strategy parameters $\mathbf{s}$ which coevolve along with the solution (and are therefore being adapted themselves). This is a particular feature of ES called self-adaptation. For a general description of the $(\mu/\rho \,\dagger\, \lambda)$-ES, see [13].

(1) $g \leftarrow 0$
(2) Initialize $P_\mu^{(g)} \leftarrow \{(\mathbf{y}_m, \mathbf{s}_m),\ m = 1, \ldots, \mu\}$
(3) Evaluate $P_\mu^{(g)}$
(4) **while** not_termination_condition **do**
(5) **for all** $l \in \lambda$ **do**
(6) $\mathcal{R} \leftarrow$ Draw ρ parents from $P_\mu^{(g)}$
(7) $\mathbf{r}_l \leftarrow$ recombine $(\mathcal{R})$
(8) $(\tilde{\mathbf{y}}_l, \tilde{\mathbf{s}}_l) \leftarrow$ mutate $(\mathbf{r}_l)$
(9) $\mathcal{F}_l \leftarrow$ evaluate $(\tilde{\mathbf{y}}_l)$
(10) **end for**
(11) $P_\lambda^{(g)} \leftarrow \{(\mathbf{y}_l, \mathbf{s}_l),\ l = 1, \ldots, \lambda\}$
(12) $P_\mu^{(g+1)} \leftarrow$ selection $(P_\lambda^{(g)}, P_\mu^{(g)}, \mu, \dagger)$
(13) $g \leftarrow g + 1$
(14) **end while**

Algorithm 1: $(\mu/\rho \,\dagger\, \lambda)$-ES.

4. Previous Work on Wavelets Adaptation

4.1. Introductory Notes. Research on adaptive wavelets has been taking place during the last two decades. At first, dictionary-based methods were used for the task. Coifman and Wickerhauser [15] select the best basis from a set of predefined functions, modulated waveforms called atoms, such as wavelet packets. Mallat and Zhang Matching Pursuit algorithm [16] uses a dictionary of Gabor functions by successive scalings, translations, and modulations of a Gaussian window function. It performs a search in the dictionary in order to find the best matching element (maximum inner product of the atom element with the signal). Afterwards, the signal is decomposed with this atom which leaves a residual vector of the signal. This algorithm is iteratively applied over the residual up to n elements. The Matching Pursuit algorithm is able to decompose a signal into a fixed, predefined number of atoms with arbitrary time-frequency windows. This allows for a higher degree of adaptation than wavelet packets. These dictionary-based methods do not produce new wavelets but just select the best combination of atoms to decompose the signal. In some cases, these methods were combined with EA for adaptive dictionary methods [17].

When the LS was proposed, new ways of constructing adaptive wavelets arose. One remarkable result is the one by Claypoole et al. [18] which used LS to adapt the *prediction* stage to minimize a data-based error criterion, so that this stage gets adapted to the signal structure. The *Update* stage is not adapted, so it is still used to preserve desirable properties of the wavelet transform. Another work which is focused on making perfect reconstruction possible without any overhead cost was proposed by Piella and Heijmans [19] that makes the update filter utilize local gradient information to adapt itself to the signal. In this work, a very interesting survey of the state of the art on the topic is covered.

These brief comments on the current literature proposals show the trend in the research community which has mainly involved the adaptation of the transform to the local properties of the signal *on the fly*. This implies an extra computational effort to detect the singularities of the signal and, afterwards, apply the proposed transform. Besides, a lot of work has been published on adaptive thresholding techniques for data compression.

The work being reported on in this paper deals with finding a complete new set of filters adapted to a given signal type which is equivalent to changing the whole wavelet transform itself. Therefore, the general lifting framework still applies. This has the advantage of keeping the computational complexity of the transform at a minimum (as defined by the LS) not being overloaded with extra filtering features to adapt to these local changes in the signal (as the transform is being performed).

Therefore, the review of the state of the art covered in this section will focus on bio-inspired techniques for the automatic design of new wavelets (or even the optimization of existing ones). This means that the classical meaning of adaptive lifting (as mentioned above) does not apply in this work. Adaptive, within the scope of this work, refers to the adaptivity of the system as a whole. As a consequence, this system does not adapt at run time to the signal being analysed, but, in contrast, it is optimized previously to the system operation (i.e., during a calibration routine or in a postfabrication adjustment phase).

4.2. Evolutionary Design of Wavelet Filters. The work described here gets its original idea from [20] by Grasemann and Miikkulainen. In their work, the authors proposed the original idea of combining the lifting technique with EA for designing wavelets. As it is drawn from [1, 10], the LS is really well suited for the task of using an EA to encode wavelets, since any random combination of lifting steps will encode a valid wavelet which guarantees perfect reconstruction.

The Grasemann and Miikkulainen method [20] is based on a coevolutionary Genetic Algorithm (GA) that encodes wavelets as a sequence of lifting steps. The evaluation run makes combinations of one individual, encoded as a lifting step, from each subpopulation until each individual had been evaluated an average of 10 times. Since this is a highly time-consuming process, in order to save time in the evaluation of the resulting wavelets, only a certain percentage of the largest coefficients was used for reconstruction, setting the rest to zero. A compression ratio of exactly 16 : 1 was used, which means that 6.25% of the coefficients are kept for reconstruction. A comparison between the idealized evaluation function and the performance on a real transform coder is shown in their work. Peak signal-to-noise ratio (PSNR) was the fitness figure used as a quality measure after applying the inverse transform. The fitness for each lifting step was accumulated each time it was used.

The most original contributions to the state of the art reported in this work [20] are two. First, they used a GA to encode wavelets as a sequence of lifting steps (specifically a coevolutionary GA with parallel evolving populations). Second, they proposed an idealized version of a transform coder to save time in the complex evaluation method that they used which involved computing the PSNR for one individual combining a number of times with other individuals from each subpopulation. This involves using only a certain percentage of the largest coefficients for reconstruction.

The evaluation consisted of 80 runs, each of which took approximately 45 minutes on a 3 GHz Xeon processor (total time $80 * 45$). The results obtained in this work outperformed the considered state-of-the-art wavelet for fingerprint image compression, the FBI standard based on the D9/7 wavelet, in 0.75 dB. The set of 80 images used was the same as the one used in this paper, as will be shown in Section 6.

Works reported by Babb et. al. [21–24] can be considered the current state of the art in the use of EC for image transform design. These algorithms are highly computationally intensive, so the training runs were done using supercomputing resources, available through the use of the Arctic Region Supercomputer Center (ARSC) in Fairbanks, Alaska. The milestones followed in their research, with references to their first published works, are summarized in the following list:

(1) evolve the inverse transform for digital photographs under conditions subject to quantization [25],

(2) evolve *matched* forward and inverse transform pairs [26],

(3) evolve coefficients for three- and four-level MRA transforms [27],

(4) evolve a different set of coefficients for each of level of MRA transforms [28].

Table 1 shows the most remarkable and up to date published results in the design of wavelet transforms using Evolutionary Computation (EC), and Table 2 shows the settings of the parameters for each reported work. The authors of these works state that in the cases of MRA the coefficients evolved for each level were different, since they obtained better results using this scheme with the exception of [20].

The use of supercomputing resources and the training times needed to obtain a solution gives an idea of the complexity of these algorithms. This issue makes their implementation as a hardware-embedded system highly unfeasible.

Table 1: State of the art in evolutionary wavelets design.

Reference	EA	Seed	Conditions	Image set	Improvement (dB)
[20]	Coevolutionary GA	Random Gaussian	MRA. 16 : 1 T[a]	Fingerprints	0.75
[21]	GA	D9/7 mutations	MRA (4). 16 : 1 T	Fingerprints	0.76
				Satellite	**1.79**
[22]	CMA-ES[b]	D9/7 mutations	64 : 1 Q[c]	Fingerprints	**3.00**
				Photographs	**2.39**
[23]	CMA-ES	0.2	MRA (3). 64 : 1 Q	Fingerprints	0.54

[a]Thresholding, [b]Covariance Matrix Adaptation-Evolution Strategy, [c]quantization.

Table 2: Parameter settings in reported work.

Reference	Parameters			Platform
[20]	$G^a = 500$	$M^b = 150(7)^c$	$N^d = 4 + 1^e$	Intel Xeon 3 GHz
[21]	$G = 15000$	$M = 800$	$N = 128$	ARSC[f]
[22]	$G = ?^g$	$M = ?$	$N = 16$	ARSC
[23]	$G = ?$	$M = ?$	$N = 96$	ARSC

[a]Generations, [b]population size, [c]parallel subpopulations, [d]individuals length (floating point coefficients), [e]integer for filter index, [f]Arctic Region Supercomputer Center, [g]unknown.

5. Proposed Simplified Evolution Strategy for an Embedded System Implementation

As proposed in the reports by Babb, et al. [22, 23], an ES was also considered within this paper scope to be the most suited algorithm to meet the requirements. However, a simpler one was chosen so that a viable hardware implementation was possible. Besides, this paper proposes, as Grasemann and Miikkulainen [20] did, the use of the LS to encode the wavelets. Therefore, it is being originally proposed here to combine both proposals from the literature so that

(i) "*search algorithm*" is set to be a *simplified* Evolution Strategy, and

(ii) "*encoding of individuals*" is done by using the Lifting Scheme.

Figure 2 shows a graphical representation of the whole idea of the paper: *let an evolutionary algorithm find an adequate set of parameters in order to maximize the wavelet transform performance from the compression point of view for a very specific type of images*.

To reduce the computational power requirements, the whole algorithm complexity must be downscaled. This involves changing not only the parameters of the evolution but the EA itself as well. In [29] the decisions made for simplifying the algorithm as compared to the previously reported state of the art are described. These proposals, which constitute the first step in the algorithm simplification, are summarized as follows:

(1) *single evolving population* opposed to the parallel populations of the coevolutionary genetic algorithm proposed in [20];

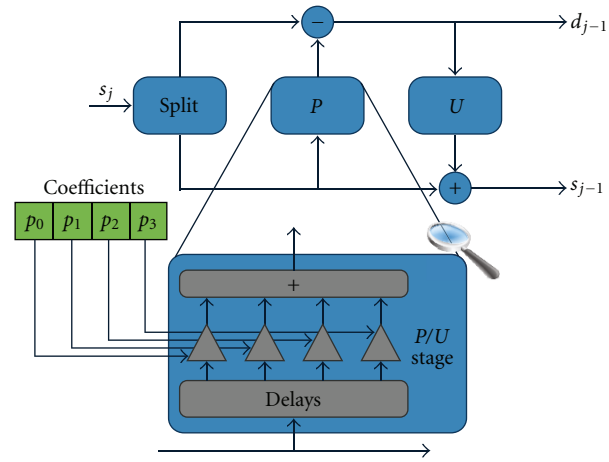

Figure 2: Idea of the algorithm.

(2) use of *uncorrelated mutations with one step size* [13] instead of the overcomplex CMA-ES method in [22, 23];

(3) evolution of *one single set of coefficients for all MRA levels*;

(4) *ideal evaluation of the transform*. Since doing a complete compression would turn out to be an unsustainable amount of computing time, the simplified evaluation method detailed in [20] was further improved. For this work, all wavelet coefficients d_j are zeroed, keeping only the trend level of the transform from the last iteration of the algorithm s_j, as suggested in [30]. Therefore, the evaluation of the individuals in the population is accomplished through the computation of the PSNR after setting entire bands of high-pass coefficients to 0. For 2 levels of decomposition, this is equivalent to an idealized 16 : 1 compression ratio.

These simplifications produced very positive results, but constraining the algorithm to evolve a single population of individuals and to use a simple mutation strategy could potentially result in a high loss of performance compared to other works. Since the evaluation of the transform performance is, by far, the most time-consuming task, this is the reason to propose the most radical simplification precisely for this task. Besides, this extreme simplification is expected to push the algorithm faster towards a reasonable

solution, which means, from a phenotypic point of view, to practically discard individuals who do not concentrate efficiently most of the signal energy in the LL bands.

There were still some complex operations pending in the algorithm so the complexity relaxation was taken even further, observing always a tradeoff between performance and size of the final circuit.

(1) *Uniform Random Distribution*. Instead of using a Gaussian distribution for the mutation of the object parameters, a uniform distribution was tested for being simpler in terms of the HW resources needed for its implementation.

(2) *Mean Absolute Error (MAE) as Evaluation Figure*. PSNR is the quality measure more widely used for image processing tasks. But, as previous works in image filter design via EC show [31], using MAE gives almost identical results because the interest lies in relative comparisons among population members.

5.1. Fixed Point Arithmetic. For the implementation of the algorithm in an FPGA device, special care with binary arithmetic has to be taken since floating point representation is not hardware (FPGA) friendly. Thanks to the LS, the Integer Wavelet Transform (IWT) [32] turns up as a good solution for wavelet transforms in embedded systems. But, since filter coefficients are still represented in floating point arithmetic, a fixed point implementation is needed.

As shown in [33, 34], for 8 bits per pixel (bpp) integer inputs from an image, a fixed point fractional format of Q2.10 for the lifting coefficients and a bit length in between 10 and 13 bits for a 2- to 5-level MRA transform for the partial results is enough to keep a rate-distortion performance almost equal to what is achieved with floating point arithmetic. This requires Multiply and Accumulate (MAC) units of 20–23 bits (10 bits for the fractional part of the coefficients + 10–13 bits for the partial transform results).

5.2. Modelling the Proposal. Prior to the hardware implementation, modelling and extensive simulations and tests of the algorithm were done using Python computing language together with its numerical and scientific extensions, NumPy and Scipy [35], as well as the plotting library MatPlotlib [36]. Fixed point arithmetic was modelled with integer types, defining the required quantization/dequantization and bit-alignment routines to mimic hardware behaviour. Figure 3 shows the flowgraph of the algorithm.

The standard "*representation*" of the individuals in ESs is composed of a set of *object parameters* to be optimized and a (set of) *strategy parameter(s)* which determines the extent to which the object parameters are modified by the mutation operator

$$\langle x_1, \ldots, x_n, \sigma \rangle \tag{1}$$

with x_i being the coefficients of the *predict* and *update* stages. Two versions were developed, one targeting floating point numbers for the first proposal [29] and another

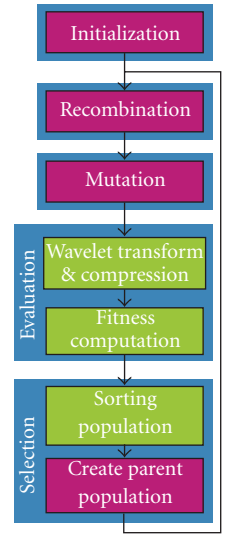

Figure 3: Flow graph of the algorithm.

one modelling fixed point behaviour in hardware. The individuals were seeded both randomly and with the D9/7 wavelet.

The "*encoding*" of each wavelet individual is of the form

$$\langle P_1, U_1, P_2, U_2, P_3, U_3, k_1, k_2 \rangle, \tag{2}$$

where each P_i and U_i consists of 4 coefficients and both k_i are single coefficients. Therefore, the total length of each chromosome is $n = 26$. As a comparison, the D9/7 wavelet is defined by $\langle P_1, U_1, P_2, U_2, k_1, k_2 \rangle$.

The "*mutation*" operator is defined as an *uncorrelated mutation with one step size*, σ. The formulae for the mutation mechanism is

$$\begin{aligned} \sigma' &= \sigma \cdot \exp^{\tau \cdot N(0,1)}, \\ x_i' &= x_i + \sigma' \cdot N_i(-\sigma', \sigma'), \\ x_i' &= x_i + \sigma' \cdot U_i(-\sigma', \sigma'), \end{aligned} \tag{3}$$

where $N(0,1)$ is a draw from the standard normal distribution and $N_i(-\sigma', \sigma')$ and $U_i(-\sigma', \sigma')$ a separate draw from the standard normal distribution and a separate draw from the discrete uniform distribution, respectively, for each variable i (for each object parameter). The parameter τ resembles the so-called *learning rate* of neural networks, and it is proportional to the square root of the object variable length n:

$$\tau \propto \frac{1}{\sqrt{\alpha n}}, \quad \alpha = \{1, 2\}. \tag{4}$$

The "*fitness function*" used to evaluate the offspring individuals, MAE, is defined as

$$\mathrm{MAE} = \frac{1}{RC} \sum_{i=0}^{R-1} \sum_{j=0}^{C-1} |I(i,j) - K(i,j)|, \tag{5}$$

Table 3: Proposed evolution strategy: summary.

Parameter/operator	Value
Individual encoding	$\langle P_1, U_1, P_2, U_2, P_3, U_3, k_1, k_2 \rangle$
Representation	$\langle x_1, \ldots, x_n, \sigma \rangle$ $n = 26$, floating/fixed point coefficients
Mutation	Strategy parameters: uncorrelated mutation, one σ Object parameters: Gaussian/uniform Initial σ: *variable*
Evaluation	MAE
Selection	Comma
Recombination	Intermediate
Parent population size	*Variable*
Offspring population size	*Variable*
Seed for initial population	Random

where R, C are the rows and columns of the image and I, K the original and transformed images, respectively. In previous works, the authors used PSNR for this task, but, as mentioned above, MAE produces the same results. However, for comparison purposes with other works, the evaluation of the best evolved individual against a standard image test set is reported as PSNR, computed as

$$\text{MSE} = \frac{1}{RC} \sum_{i=0}^{R-1} \sum_{j=0}^{C-1} |I(i,j) - K(i,j)|^2,$$
$$\text{PSNR} = 10 \log_{10} \left(\frac{I_{\max}}{\text{MSE}} \right), \tag{6}$$

where MSE stands for Mean Squared Error and $I_{\max}$ is the maximum possible value of a pixel, defined for B bpp as $I_{\max} = 2^{B-1}$.

For the "*survivor selection*", a *comma selection* mechanism has been chosen, which is generally preferred in ESs over *plus selection* for being, in principle, able to leave (small) local optima and not letting misadapted strategy parameters survive. Therefore, no *elitism* is allowed.

The "*recombination*" scheme chosen is *intermediate recombination* which averages the parameters (alleles) of the selected parents.

Table 3 gathers all the information related to the proposed ES.

5.3. Test Strategy to Validate the Algorithm. An incremental approach has been chosen as the strategy to successively build the proposed algorithm. First of all, the complete, software-friendly implementation of the ES in floating point arithmetic was accomplished. This validated the choice of a simple ES to design new lifting wavelet filters adapted to a specific type of signal. Since the target deployment platform is an FPGA, fixed point arithmetic is desired. Therefore, the next step was to test the performance of the fixed point implementation of the algorithm. The next great simplification to the algorithm was switching from a Gaussian-based mutation operator for the object parameters to a uniform-based one.

In order to find the best set of parameters, several tests for different combinations of them have been done in order to gather statistics of the evolutionary search performance for the *training image*, chosen randomly from the first set of 80 images of the FVC2000 fingerprint verification competition [37]. When changing parent population size, the offspring population size is modified accordingly to keep the selection pressure as suggested for ESs ($\mu/\lambda \approx 1/7$). Besides, the number of recombinants has been chosen to match approximately half of the population size.

The authors are aware that more tests can be performed for different settings of the parameters. Anyway, the results presented in the next section show how the proposed algorithm is widely validated within a reasonable number of computing hours (it has to be reminded here that the proposed deployment platform is an FPGA, so further tests have to be done in hardware). However, an extra test was run to check whether or not introducing *elitism* was good for the evolution. The successive *simplify, test, and validate* steps are summarized as follows:

(1) begin with the SW-friendly, full precision arithmetic, simplest ES. Find a suitable initial mutation strength. Perform several tests for different values of σ;

(2) HW-friendly arithmetic implementation. Compare with the result of (1) in fixed point arithmetic;

(3) HW-friendly mutation implementation. Compare with the result of (2) using uniform mutation;

(4) repeat (1) to check whether the same initial mutation strengths still apply after the simplifications proposed in (2) and (3);

(5) HW-friendly population size. Test the performance for different population sizes;

(6) test performance using *plus* selection operator.

Tables 4, 5, 6, 7, 8, and 9 compile the information regarding the five different tests mentioned above. Please note that when the test comprises variable parameters, the number of runs shown in the table is done for each parameter value so that different, independent runs of the algorithm are executed in order to have a statistical approximation to the repeatability of the results produced.

6. Results

6.1. Tests Results. The results obtained for each of the tests can be found in this section. All of them are compared with the D9/7 (JPEG2000 lossy and FBI fingerprint compression standard) and D5/3 (suitable for integer to integer transforms, JPEG2000 lossless standard) reference wavelets implemented in fixed or floating point arithmetic and evaluated with the proposed method. All the experiments reported in this paper have also used, as in [20], the first set of 80 images of the FVC2000 fingerprint verification

Table 4: Test no. 1. Initial mutation step σ.

Fixed parameters	Arithmetic	Floating point
	Mutation	Gaussian
	Population size	$(10/5, 70)$
Variable parameters	Mutation strength	$\sigma = \{0.1, \ldots, 2.0\}, \Delta\sigma = 0.1$
Runs	10 for each parameter variation step (total 200)	
Output	*Performance versus σ sweep*	
	Initial mutation strength σ_B ? for Gaussian mutation	

Table 5: Test no. 2. Fixed point arithmetic validation.

Fixed parameters	Arithmetic	Fixed point, Q_b bits
	Mutation	Gaussian
	Mutation strength	σ_B
	Population size	$(10/5, 70)$
Variable parameters	Fractional part bit length	$Q_b = \{8, 16, 20\}$ bits
Runs	50 for each parameter variation step (total 150)	
Output	*Fixed point validation*	
	Performance for σ_B per run	

Table 6: Test no. 3*a*. Uniform mutation validation.

Fixed parameters	Arithmetic	Fixed point, 16 bits
	Mutation	Uniform
	Mutation strength	σ_B
	Population size	$(10/5, 70)$
Runs	10	
Output	*Uniform mutation validation*	
	Performance for uniform mutation per run	

competition. Images were black and white, sized 300×300 pixels at 500 dpi resolution. One random image was used for training and the whole set of 80 images for testing the best evolved individual in each optimization process.

Table 10 shows a compilation of the figures produced during the tests. The performance for each of the standard wavelet transforms, D9/7 and D5/3, obtained with the training image is shown in Table 11.

The data collected on the *boxplot* figures show the statistical behaviour of the algorithm. Besides the typical values shown in this kind of graphs, all of them, like Figure 4, show also numerical annotations for the average (top-most) and median (bottom-most) values at the top of the figure, a circle representing the average value in situ (together with the boxes) and the reference wavelets performance.

For the first step of the proposal, Test no. 1, practically all the runs (10 runs for each of the 20 σ steps, which makes a total of 200 independent runs) of the algorithm evolve towards better solutions than the standard wavelets. Statistical results of the test are included in Figure 4.

Fixed point validation which is accomplished in Test no. 2 is shown in Figure 5 for $Q_b = \{8, 16, 20\}$ bits. 50 runs were made for each Q_b value. It is clear, as expected from the comments in Section 5.1, that 8 bits for the fractional part are not enough to achieve good performance, while the 16 and 20 bits runs behave as expected. Test no. 3*a* tries to validate uniform mutation as a valid variation operator for the EA. Good results are also obtained, as extracted from Figure 6. The only possible drawback for both tests may be the extra dispersion as compared with the original floating point implementation.

When the algorithm is simplified as in Test no. 3*b*, a slightly different behaviour from previous tests is observed. The most remarkable result is the difference in the performance obtained for equivalent σ values which can be seen in Figure 7. For $\sigma \approx \{1.0, \ldots, 2.0\}$, the dispersion of the results is very high, and a reasonable number of individuals are not evolving as expected. Therefore, the test was repeated for $\sigma = \{0.01, \ldots, 0.1\}$, in steps of 0.01. This involves doing another 100 extra runs which are shown in Figure 8, for a total of 300 independent runs. This σ extended test range shows how the algorithm is again able to find good candidate solutions.

Results from Test no. 4 in Figure 9 show the expected behaviour after changing the population size. Making it smaller as in the $(5/2, 35)$ run does not help in keeping the average good performance of the algorithm demonstrated in previous tests for a population size of $(10/5, 70)$. On the other hand, increasing the size to $(15/5, 100)$ shows how the interquartile range is reduced. However, such a reduction would not justify the increase in the computational power required to evolve a 1.5 times bigger population.

The different selection mechanism chosen for Test no. 5 led to a slightly increased performance of the evolutionary search compared with Test no. 3*b*, as shown in Figures 10 and 11.

6.2. Results for Best Evolved Individual. The whole set of results obtained for each test show that the algorithm is able to evolve good solutions (better than the standard wavelets) for an adequate setting of parameters. However, these results

Table 7: Test no. 3*b*. Initial mutation step σ for Uniform mutation.

Fixed parameters	Arithmetic	Fixed point, 16 bits
	Mutation	Uniform
	Population size	(10/5, 70)
Variable parameters	Mutation strength	$\sigma = \{0.1, \ldots, 2.0\}, \Delta\sigma = 0.1$
		$\sigma^{a} = \{0.01, \ldots, 0.1\}, \Delta\sigma = 0.01$
Runs	10 for each parameter variation step (total 200 + 100)	
Output	*Performance versus σ sweep*	
	Initial mutation strength σ_B? for uniform mutation	

[a] See Section 6.1 for a justification of the extended range of σ.

Table 8: Test no. 4. Effect of the population size.

Fixed parameters	Arithmetic	Fixed point, 16 bits
	Mutation	Uniform
	Mutation strength	σ_B
Variable parameters	Population size	(10/5, 70), (5/2, 35), (15/5, 100)
Runs	10 for each parameter variation step (total 30)	
Output	*Performance versus population size*	

Table 9: Test no. 5. *Plus* selection operator.

Fixed parameters	Arithmetic	Fixed point, 16 bits
	Mutation	Uniform
	Population size	(10/5, 70)
Variable parameters	Mutation strength	$\sigma = \{0.01, \ldots, 1.1\}$
Runs	10 for each parameter variation step (total 200)	
Output	*Performance for* plus *selection operator versus σ sweep*	

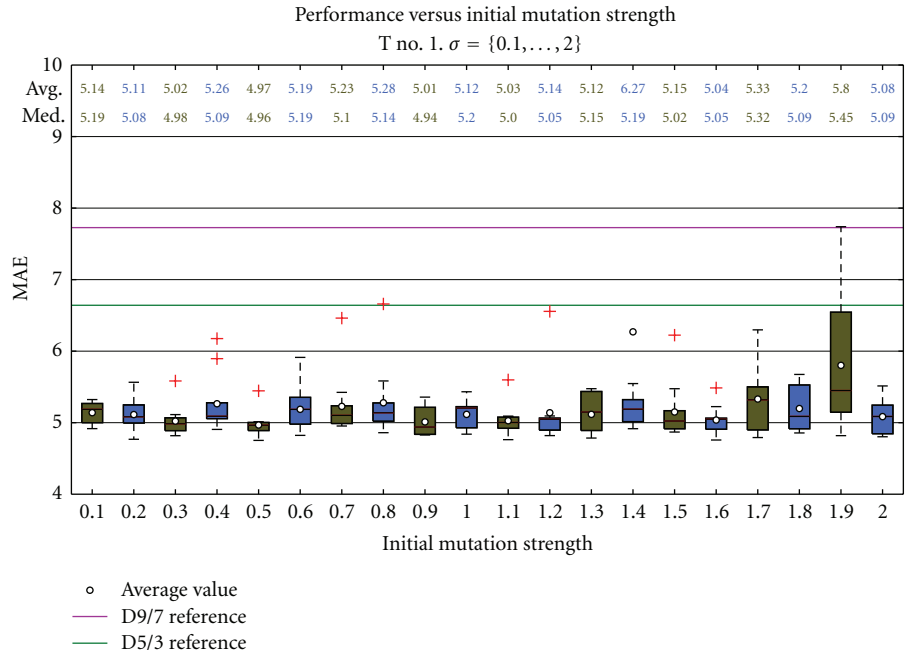

Figure 4: Test no. 1.

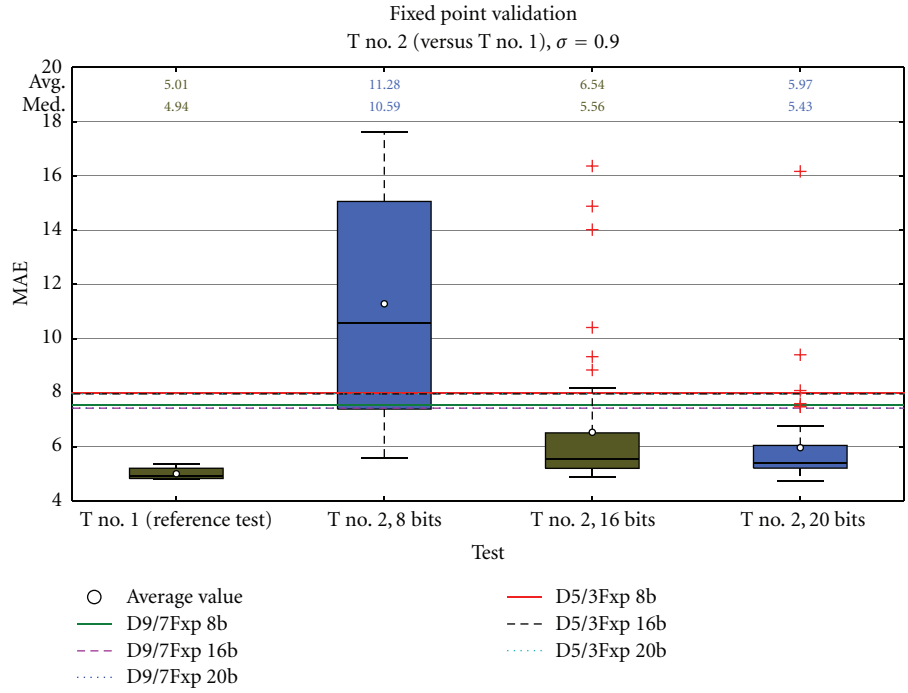

Figure 5: Test no. 2.

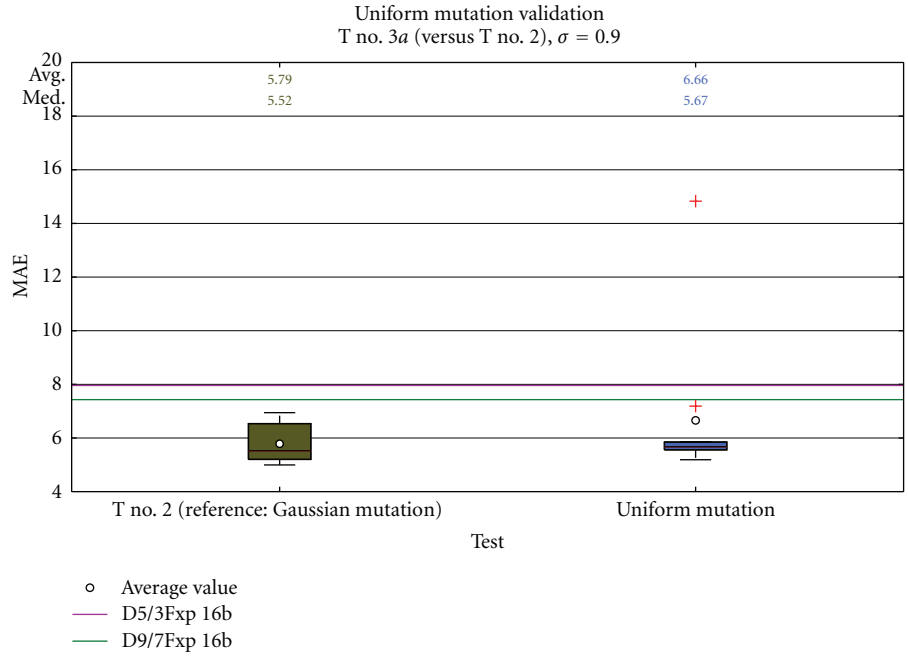

Figure 6: Test no. 3*a*.

Table 10: Tests results figures.

Test no.	1	2	3*a*	3*b*	4	5
Figure	4	5	6	7, 8	9	10, 11

are just for the training image. Therefore, how does the best evolved individual behave for the whole test set?

In this section, the comparisons between the best evolved individual and the reference wavelets against the whole test set are shown. Although evolution used MAE as the fitness function, in order to maximize comparability with other works, the quality measure is given here as PSNR. Results for Gaussian mutation in floating point arithmetic and uniform mutation in fixed point arithmetic, both for *comma* and *plus* selection strategies, respectively, are included. These two

Table 11: Standard wavelets performance for the training image.

Wavelet	D9/7				D5/3			
Arithmetic	Floating point	Fixed point[a], Q_b			Floating point	Fixed point[a], Q_b		
		8 bits	16 bits	20 bits		8 bits	16 bits	20 bits
Performance (MAE)	6.6413	7.5625	7.4190	7.4221	7.7271	7.9882	7.9595	7.9577

[a] Fixed point arithmetic, Q_b bits for the fractional part.

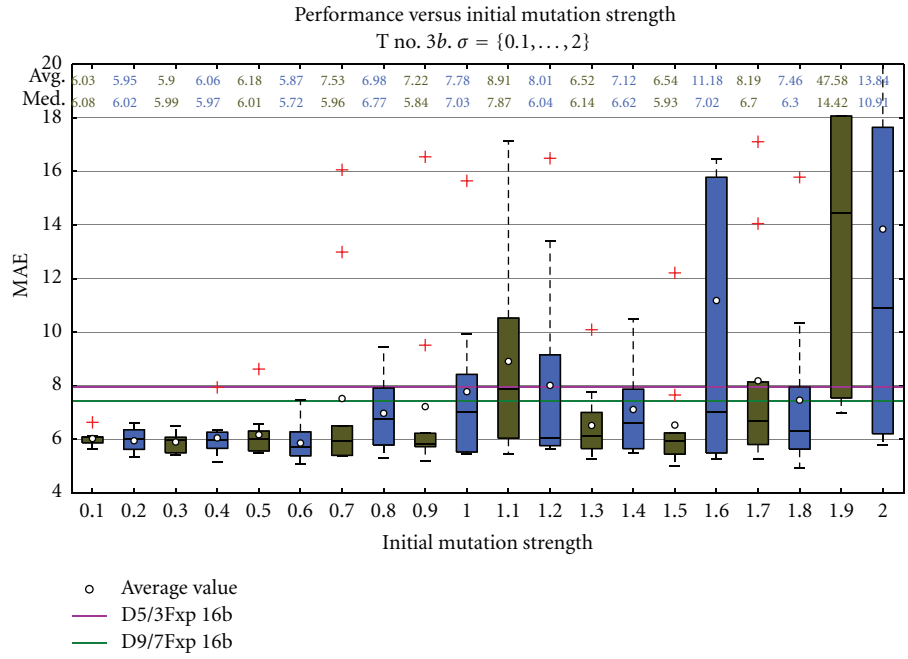

Figure 7: Test no. 3*b*.

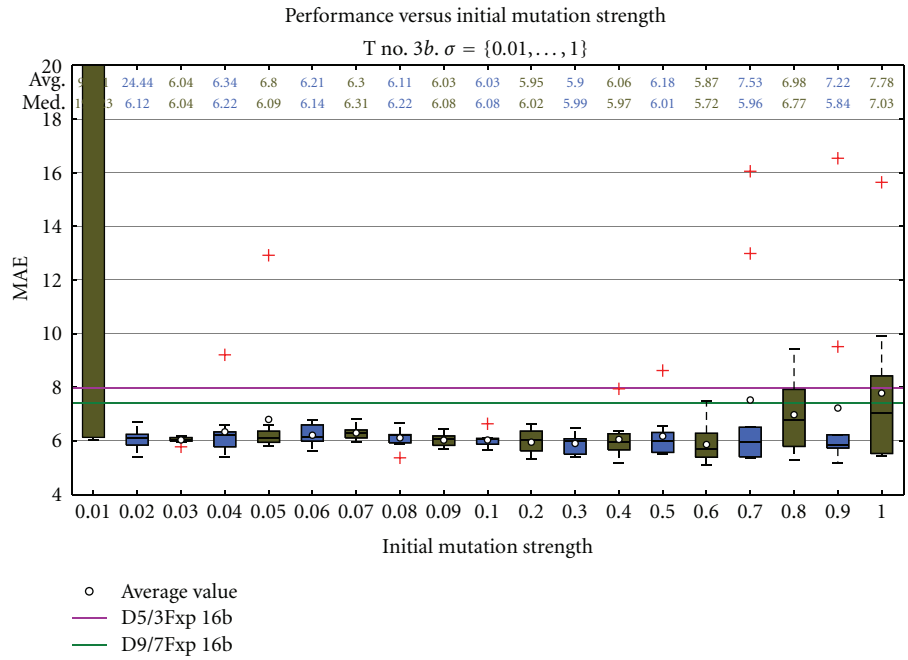

Figure 8: Test no. 3*b*.

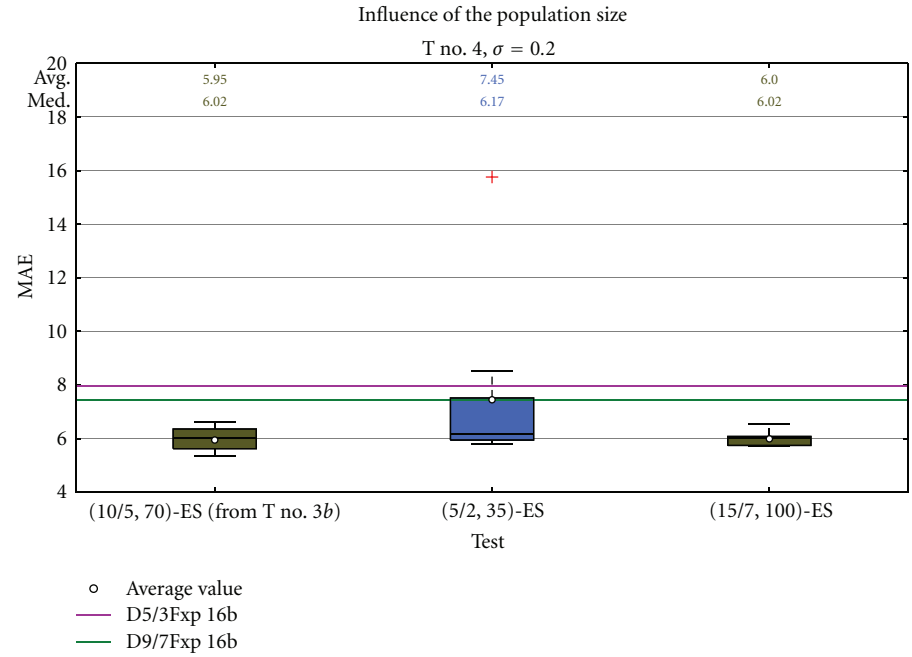

Figure 9: Test no. 4.

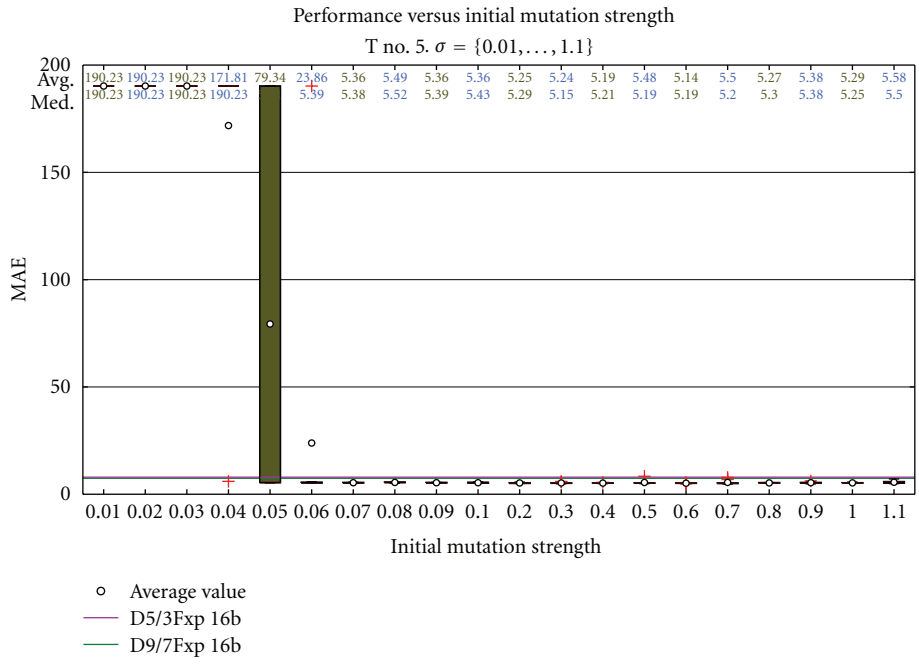

Figure 10: Test no. 5. $\sigma = \{0.01, \ldots, 1.0\}$.

sets of results will assist in the validation of the successive simplifications made to the originally proposed algorithm.

Figure 12 shows a graph of the evolution run of the best individual for the whole set of tests for floating point arithmetic and Gaussian mutation and fixed point arithmetic and uniform mutation for both, *comma* and *plus* selection, respectively. The *best* individual has been chosen as the one averaging the highest performance against the whole test set (not the one performing best for the training image, as is shown in previous figures). Equivalently, Figures 13, 14, and 15 show the comparison against the whole test set for each one of the best individuals of Figure 12. Figure 16 is a direct comparison of a particular image of the test set showing how the best evolved individual for *plus* selection behaves against a fixed point implementation of D9/7. Error images and histograms are included in Figure 17 since direct visual inspection of these images is not easy and will not probably offer enough information for the human eye to make a fair

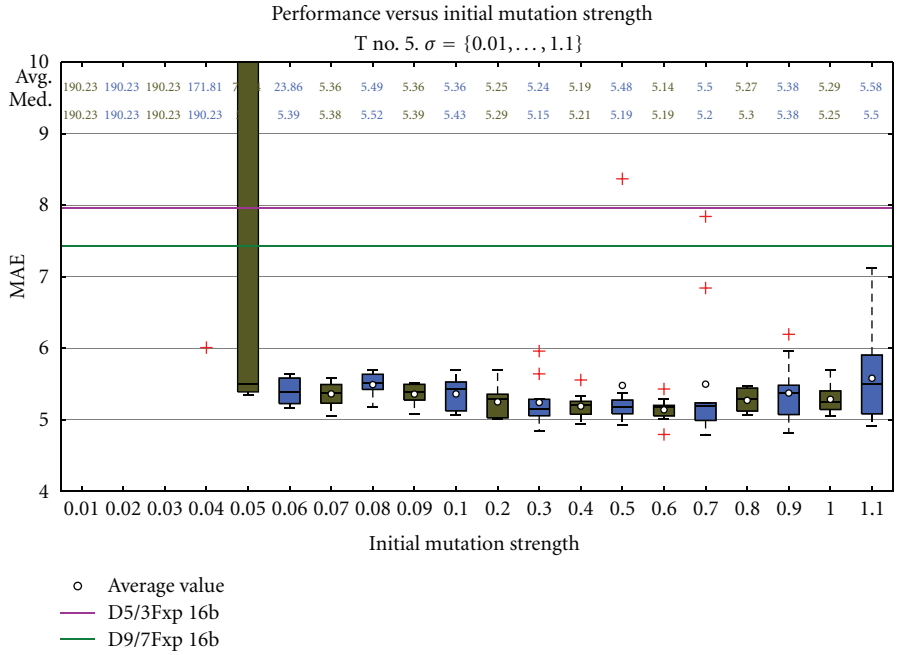

Figure 11: Test no. 5. Zoom-in in Figure 10.

judgement, though some artifacts are clearly visible in the performance of the D9/7 wavelet as shown in Figure 16(c). It can be seen from these error images and histograms how, after applying a *forward transform* + *(ideal) compression* + *inverse transform* to an image, the result of using the evolved wavelet generates an image which keeps a higher degree of similarity with the original one than using the standard D9/7.

6.3. Some Comments on Results. Section 5.3 featured a discussion on the design validation followed to obtain a hardware-friendly ES by successively simplifying a previously validated and much more complex algorithm. Various independent test runs have been performed to look for the best setting of parameters, beginning with a software-friendly, high-precision floating point arithmetic version which used Gaussian mutations. After simplifying the algorithm and validating each of the steps, several conclusions can be extracted.

Because of its usual influence in EAs, search for an adequate mutation rate (mutation strength in ESs) has enjoyed a particular computing effort. It can be said, however, that, for this particular optimization problem, the mutation strength is not critical, as long as it belongs to a reasonable range. Outside of that range, evolution is not able to find good candidate solutions. Table 12 shows the most suitable range of values found for σ, chosen as those runs resulting in average performance values behaving better than the standard wavelets.

The whole set of tests have validated the proposal and found a reasonably good set of parameters for the problem at hand which is

(i) fixed point arithmetic, $Q_b = 16$ bits,

Table 12: Initial mutation strength range.

Conditions	Values
Floating point, Gaussian mutation	$\sigma = \{0.1, \ldots, 2.0\}$
Fixed point, uniform mutation, *comma* selection	$\sigma = \{0.03, \ldots, 0.6\}$
Fixed point, uniform mutation, *plus* selection	$\sigma = \{0.07, \ldots, 1.1\}$

(ii) population sizes: (10/5, 70),

(iii) selection operator with elitism: *plus* selection,

(iv) uniform mutation with an initial mutation step contained within the range shown in Table 12.

As can be observed in Figure 12 the algorithm stagnates soon in every single run, around generation 200 for the floating point and Gaussian mutation runs and generation 160 for fixed point uniform mutation and comma selection. In the floating point case, the stagnation is not complete because it keeps on improving very slowly, but in practical terms it does not imply a substantial improvement in the quality of the transform. Although the best individuals keep on stagnating if elitism is introduced in the evolution, the worst ones still maintain some degree of variation, improving the overall algorithm performance as compared to the nonelitist strategy.

Table 13 shows a comparison of the (best) obtained results (those corresponding to *plus selection*, Test no. 5) against previously reported works. It is clear that an ES is better suited than a GA for this task of optimizing real-valued vectors for wavelet filters, as confirmed by our results and by previous results from other authors as shown in Section 4. This is an expected result since they

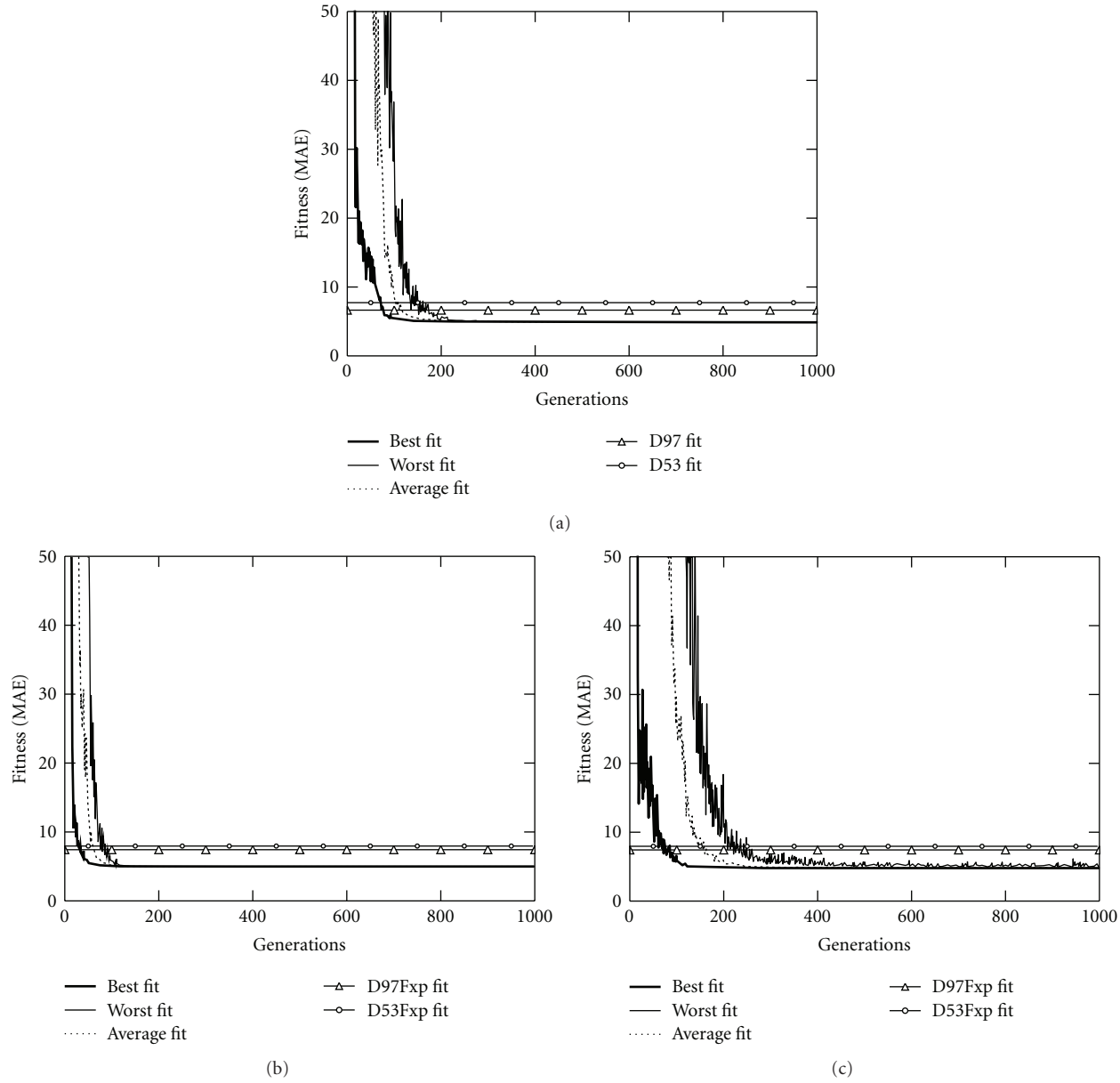

Figure 12: Best evolution run for (a) floating point arithmetic and Gaussian mutation; (b) fixed point arithmetic and uniform mutation, comma selection; (c) fixed point arithmetic and uniform mutation, plus selection.

were originally developed for this task (optimizing real-valued vectors). Compared with [22], where the best evolved wavelet outperforms the reference wavelet by 3.00 dB, the performance versus complexity tradeoff in the algorithm proposed in this paper achieves a good 1.57 dB improvement, which corresponds to the 30.31 dB performance against the whole test set as shown in Figure 15. Besides, it should be noted that for all the runs (140) corresponding to the σ range shown in Table 12 for Test no. 5, the average performance obtained was 29.76 dB, where only 4 out of the whole 140 runs were not able to improve existing wavelets.

Table 13: Comparison of best evolved wavelet against state of the art.

Reference	EA	Seed	Improvement over D9/7 (dB)
[20]	Coevolutionary GA	Random Gaussian	0.75
[21]	GA	D9/7 + mutations	0.76
[22]	CMA-ES	D9/7 + mutations	3.00
This paper	*ES*	*Random Gaussian*	*1.57*[a]

[a] Improvement over fixed point arithmetic version.

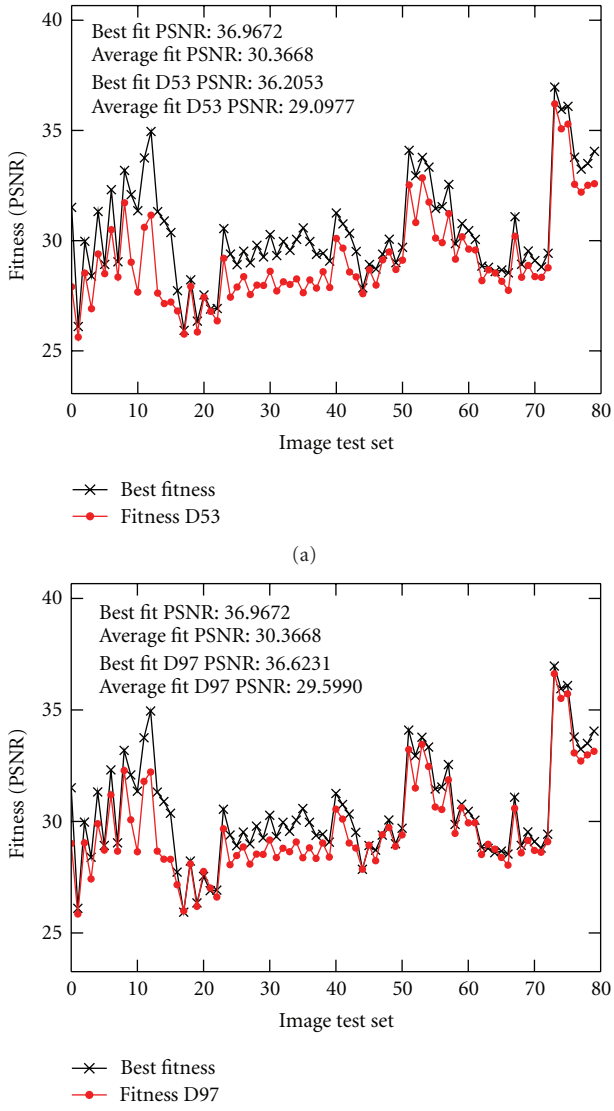

Figure 13: Performance of the best evolved individual for Floating point arithmetic and Gaussian mutation against the whole test set: (a) for D5/3 and (b) for D9/7 wavelets.

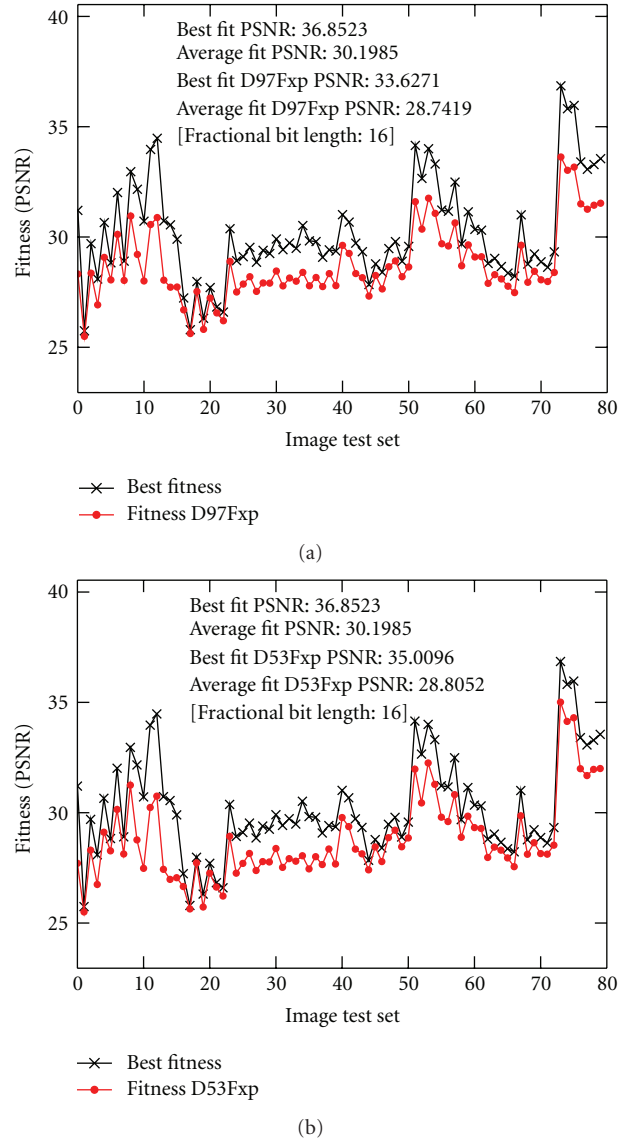

Figure 14: Performance of the best evolved individual for fixed point arithmetic and uniform mutation (*comma* selection) against the whole test set: (a) comparison with D9/7 and (b) comparison with D5/3 wavelets.

7. Hardware Implementation

7.1. Architecture Mapping. HW/SW Partitioning. Typical implementations of evolutionary optimization engines in FPGAs place the EA in an embedded processor. With this approach, some degree of performance is sacrificed to gain flexibility in the system (needed to fine tune the algorithm), so that modifications may be easily done to the (software) implementation of the EA (which is, of course, much easier than changing its hardware counterpart). Table 14 shows the partitioning resulting from applying this design philosophy. According to Algorithm 1, each of the EA operators are shown in the table together with further actions to be accomplished: *recombination* (of the selected parents), *mutation* (of the recombinant individuals to build up a new offspring population), *evaluation* (of each offspring individual), and *selection* (of the new parent population).

Table 14: HW/SW partitioning of the system.

EA operator	Further actions	HW	SW
Recombination	—		√
Mutation	—		√
Evaluation	*Wavelet transform*	√	
	Fitness computation	√	
Selection	*Sorting population*	√	
	Create parent population		√

When a new offspring population is ready, each of its individuals is sequentially sent to the hardware module

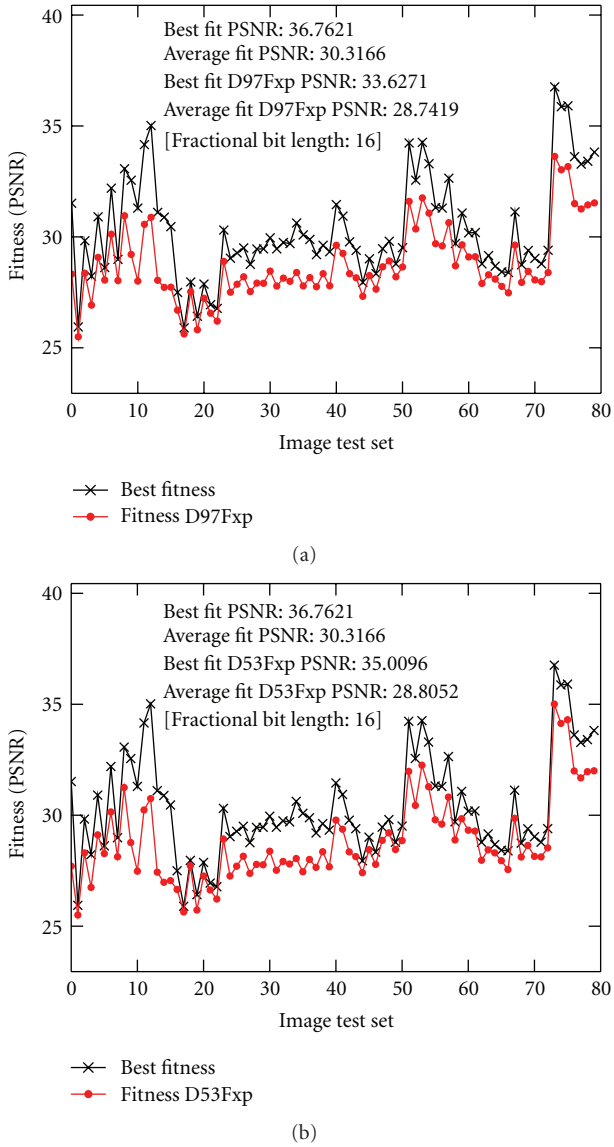

FIGURE 15: Performance of the best evolved individual for fixed point arithmetic and uniform mutation (*plus* selection) against the whole test set: (a) comparison with D9/7 and (b) comparison with D5/3 wavelets.

FIGURE 16: Transform performance. (a) Original fingerprint image; comparison of the performance obtained with (b) D9/7 fixed point implementation and (c) best evolved individual.

responsible for its evaluation. This comprises the computation of the fitness as the Mean Absolute Error (MAE) as shown in (5). To tackle it, the following sequence of operations has to be done: Forward Wavelet Transform (fWT), Compression (C), Inverse Wavelet Transform (iWT) (*wavelet transform*), and MAE figure computation (*fitness computation*). Once each offspring individual has been evaluated, the population is sorted according to the result (*sorting population*). At this stage, the microprocessor may close the evolutionary loop creating the new parent population. Afterwards, recombination and mutation are applied, and a new offspring population will be available for evaluation.

Figure 18 shows the proposed conceptual architecture capable of hosting such a system. The functions which have been implemented in hardware work as attached peripherals to the microprocessor embedded in the FPGA (PowerPC 440).

Since the LS was proposed, several hardware implementations have been reported both for ASICs and FPGAs (JPEG2000 adopted LS). This means that good results centred on exploiting LS features to obtain fast implementations have already been done. But the objectives at this stage of the work are just to prove and validate the concepts and the feasibility of the system as a whole. Therefore, the implementation of the Wavelet Transform is a direct, algorithmic mapping of the LS to its hardware equivalent VHDL description (i.e., no hardware optimizations at the level of data dependencies are accomplished).

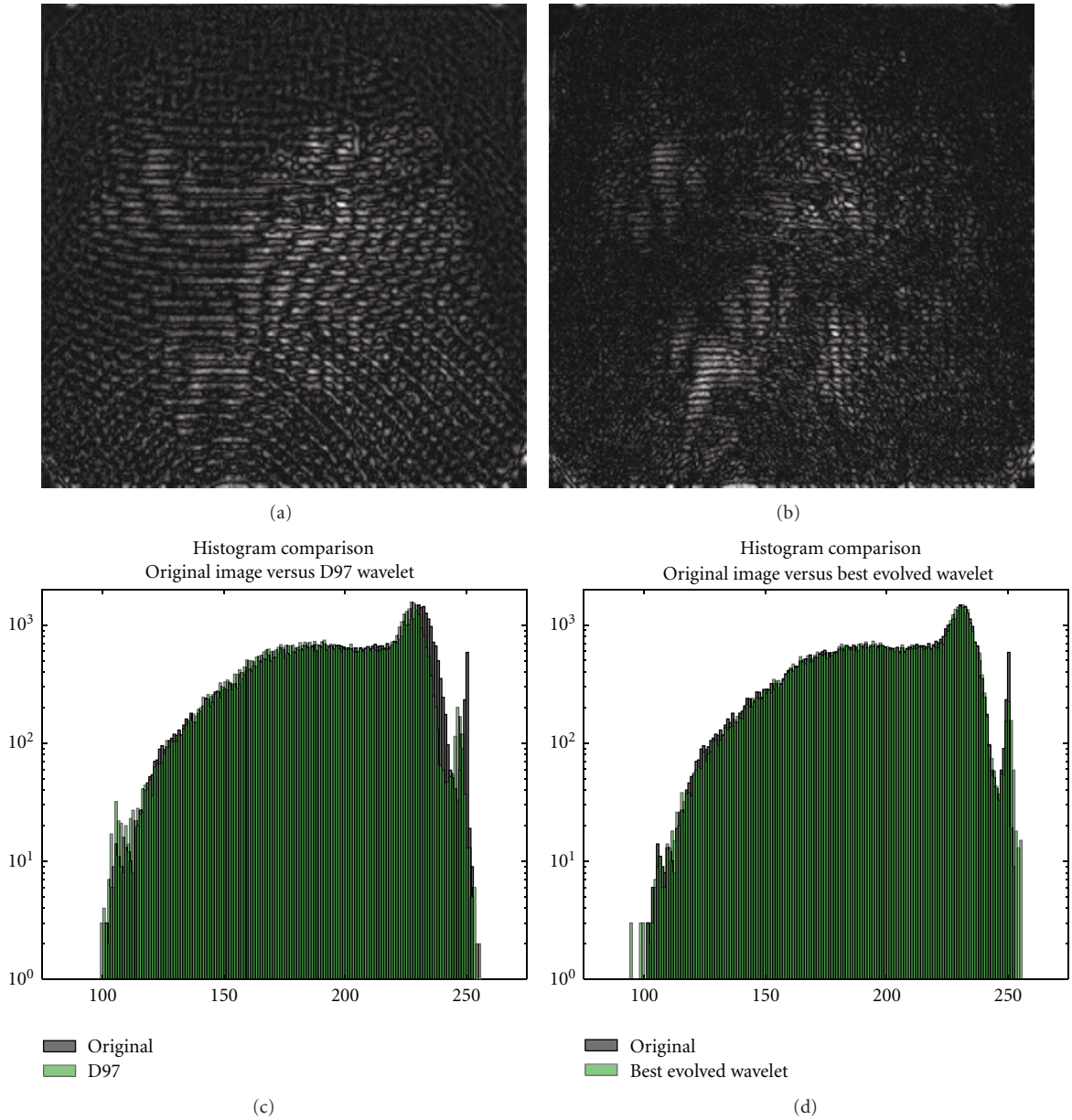

Figure 17: Transform performance. The top row shows the error introduced by each transform: (a) is the error image for the D97 wavelet and (b) for the best evolved individual. The bottom row shows the histograms of each image transform, where: (c) is for D97 and (d) for the best evolved individual.

Taking advantage of the LS features, the fWT and iWT can be computed by just doing a sign flip and a reversal in the order of the operations (P and U stages), so both modules are sharing hardware resources in the FPGA. The Compression block is simple, since it only needs to substitute the fWT result by zeros for each datum of the details bands. Therefore, it works in parallel with the fWT. In a similar manner, the Fitness module computes the difference image as each pixel is produced by the iWT.

The fWT/iWT module is built up by applying the sequence of P, U stages dictated by the LS. To mimic the high-level modelling of the algorithm (see Section 5.2), 6 stages have been implemented ($3P$ and $3U$), each one containing 4 filter coefficients which is enough to implement the most common wavelets utilized at present. Section 7.3 shows the first preliminary results of the implementation. The implementation of each P, U stage can be seen in Figure 19.

The Block RAM modules (BRAMs) embedded in the FPGA are used as data memory for the wavelet transform module. During evolution, it hosts the training image so that the highest memory bandwidth possible is achieved. It has been overdimensioned to host up to four 256×256 pixels (8 bpp) images to speed up the test phase. Therefore, when evolution has finished, this extra memory can be used to load the test images from the system memory. In this phase, one of

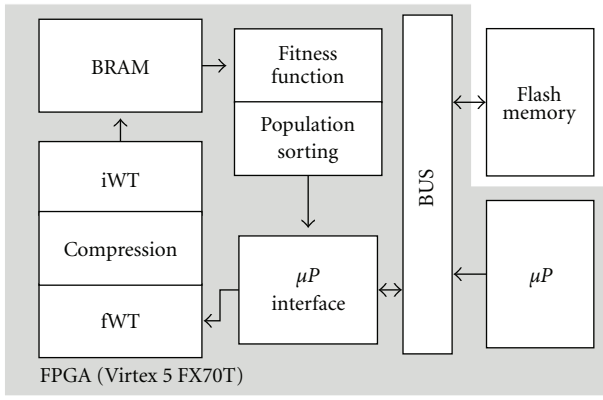

Figure 18: System level architecture.

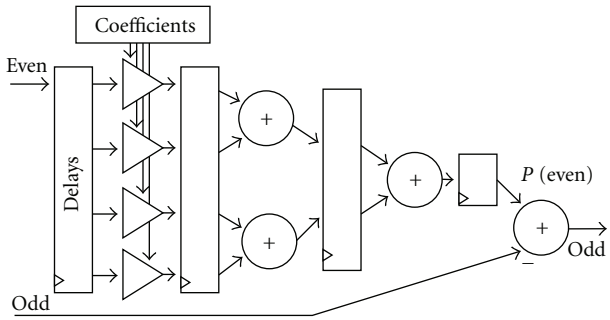

Figure 19: Predict/Update stage implementation.

the four sub-banks is used for the actual image being tested, and the other three are loaded with the following test images in the meantime, acting as a multi-ping-pong memory.

7.2. HW/SW Partitioning Validation. The model developed to validate the algorithm has been profiled. Table 15 shows profiling results for 500 generations for each EA operator. Table 14 is repeated (for clarity) adding extra columns with the result values. Absolute values are not of real interest (although NumPy routines are highly optimized, a C implementation would be faster), since what is being checked is the relative amount of time spent in each phase so that design partitioning is validated as a whole. As expected, most of the time is consumed evaluating the individuals. In each generation, 20.479 ms (= 1433.56/(500 generations ∗ 70 individuals ∗ 2 transforms)) are needed to compute a single wavelet transform, whether it is a forward or an inverse one. The obtained results validate the design partitioning proposed except for the *selection* operator, which is low enough to be implemented in SW. The reason to choose an HW implementation for it is that it can be applied as results are produced by the fitness computation module, saving extra time. In contrast, the simulation of the Python model runs on a single processor thread. Therefore, all operators are applied sequentially. But in the hardware implementation, some operators can be easily applied in parallel. For this reason, and depending on the scope of the system (see Section 8), some other operator will probably benefit from being implemented in hardware, as, for example, the mutation. Besides, the subset of the C language used to program the PowerPC processor in the FPGA imposes restrictions that will probably cause the percentage of the time each operator takes to compute to increase.

Table 15: Algorithm code profiling.

EA operator	Further actions	HW	SW	Time[a]	%
Recombination	—		√	0.14	0.009
Mutation	—		√	0.43	0.029
Evaluation	*Wavelet transform*[b]	√		1433.56	97.470
	Compression	√		4.96	0.337
	Fitness computation	√		31.62	2.150
Selection	*Sorting population*	√		0.040	0.003
	Parent population		√		

[a] All results in seconds.
[b] Results show computation time for both, forward and inverse wavelet transform.

7.3. Preliminary Hardware Implementation Results. The prototype platform selected is an ML507 development board, which contains a Xilinx Virtex 5 XC5VFX70T FPGA device with an embedded PowerPC processor, responsible for running the ES. Table 16 shows the preliminary implementation results for an overdimensioned datapath of 32 bits, using 16 bits for the fractional part representation. This implementation is directed towards a system level functional validation in the FPGA, giving higher area results than expected for the final system.

The current hardware, nonoptimized implementation, delivers one result each clock cycle. For a 256 × 256 pixels image, with a clock frequency of 100 MHz, the computation time of a wavelet transform is approximately 0.65 ms. This is a speedup factor of around (20.5/0.65) 31 times, which would turn into 45 seconds to do all the transforms required by a population of 70 individuals during 500 generations.

8. Conclusion and Future Work

A bio-inspired optimization algorithm designed to improve wavelet transform for image compression in embedded systems has been proposed and validated. A simpler method than the standard ES (and simpler than other previously evolutionary-based reported works) has been developed to find a suitable set of lifting filter coefficients to be used in the aforementioned wavelet transformation. Fixed point arithmetic implementation has been used to validate the upcoming hardware implementation.

The profiling results of the algorithm simulations have validated the proposed HW/SW partitioning, so the resulting hardware architecture can be implemented in the FPGA device. A preliminary test implementation has been prepared to perform a system level functional validation. As these preliminary synthesis results show, the FPGA will be able to host the complete system. Currently, the rest of the system is being implemented in the FPGA before functional simulations are done and an optimized version is implemented if needed.

Table 16: Preliminary implementation results for the main modules in the system.

Module	Resources				Frequency (MHz)
	Slice LUTs	Slice registers	DSP48Es	BRAM (Kb)	
fWT/iWT	3077/44800	3905/44800	62/128	—	147
Compression	35/44800	31/44800	—	—	426
Fitness function	74/44800	60/44800	—	—	348
Population sorting	2913/44800	2140/44800	—	—	233
Image memory	333/44800	20/44800	—	2304/5328	—

When the final implementation of the system in the FPGA is finished and tested, profiling results will be obtained. This is necessary due to the possible effect that the C language subset used to program the PowerPC processor in the FPGA may have, which could impose restrictions that will probably cause the percentage of time each operator takes to compute to increase drastically. For example, if this was the case for the uniform mutation operator (which will initially be implemented in SW), further simplifications to this operator could be tested as suggested for ESs in the literature. Anyway, since the algorithm finds a solution around generation 200, it can be said, being on the conservative side, that, if 500 generations are needed to evolve, just 45 seconds would elapse for the most time-consuming task, making this a sufficiently fast adaptive system.

The current status of this paper shows how adaptive compression for embedded systems based on bio-inspired algorithms can be looked at. Besides, since the process is sped-up by a large factor in the hardware implementation as compared to the software, PC-based model, the system can also be conceived as an accelerator for the optimization process of wavelet transforms (for the construction of custom wavelets).

For a generic vision system as in the one mentioned in the Introduction, this paper allows for the fact that both approximations to *adaptation* mentioned in Section 4 can be combined, firstly defining a new set of wavelets adapted to a specific type of signals (covered by this paper) and, during system operation, using some of the proposed methods in the literature that may help the system to further adapt to local changes in the signal. However, it can be said that, if the EA has been successful and the training data set properly chosen, there should not be a drastic improvement, since the EA should have acquired enough knowledge of that specific type of signal. Moreover, in a hypothetical continuously evolving system, a mechanism that looks for reductions of performance (EA kept running on the background) can be implemented and triggered to keep on evolving the wavelet if relevant changes happen in the input signal (some of them probably caused by a degradation in the sensing devices that diminish the acquired signal quality).

Acknowledgments

This work was supported by the Spanish Ministry of Science and Research under the project DR.SIMON (Dynamic Reconfigurability for Scalability in Multimedia-Oriented Networks) with TEC2008-06486-C02-01. L. Sekanina has been supported by MSMT under research program MSM0021630528 and by the Grant of the Czech Science Foundation GP103/10/1517. Rubén Salvador would like to thank the support received from the Department of Computer Systems, Brno University of Technology, during his research stay as part of his PhD degree.

References

[1] W. Sweldens, "The lifting scheme: a custom-design construction of biorthogonal wavelets," *Applied and Computational Harmonic Analysis*, vol. 3, no. 2, pp. 186–200, 1996.

[2] D. Taubman and M. Marcellin, *JPEG2000: Image Compression Fundamentals, Standards and Practice*, Springer, 1st edition, 2001.

[3] W. MacLean, "An evaluation of the suitability of FPGAs for embedded vision systems," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR '05)*, p. 131, June 2005.

[4] B. Cope, P. Y. K. Cheung, W. Luk, and S. Witt, "Have GPUs made FPGAs redundant in the field of Video Processing?" in *Proceedings of the IEEE International Conference on Field Programmable Technology*, pp. 111–118, December 2005.

[5] S. Che, J. Li, J. W. Sheaffer, K. Skadron, and J. Lach, "Accelerating compute-intensive applications with GPUs and FPGAs," in *Proceedings of the Symposium on Application Specific Processors (SASP '08)*, pp. 101–107, June 2008.

[6] Z. Wei, D.-J. Lee, B. E. Nelson, J. K. Archibald, and B. B. Edwards, "FPGA-based embedded motion estimation sensor," *International Journal of Reconfigurable Computing*, vol. 2008, no. 636145, p. 8, 2008.

[7] C. Farabet, C. Poulet, and Y. LeCun, "An FPGA-based stream processor for embedded real-time vision with convolutional networks," in *Proceedings of the IEEE 12th International Conference on Computer Vision Workshops (ICCV '09)*, pp. 878–885, September 2009.

[8] B. Jawerth and W. Sweldens, "Overview of wavelet based multiresolution analyses," *SIAM Review*, vol. 36, no. 3, pp. 377–412, 1994.

[9] S. Mallat, *A Wavelet Tour of Signal Processing*, Academic Press, 2nd edition, 1999.

[10] W. Sweldens, "The lifting scheme: a construction of second generation wavelets," *SIAM Journal on Mathematical Analysis*, vol. 29, no. 2, pp. 511–546, 1998.

[11] W. Sweldens, "The lifting scheme: a new philosophy in biorthogonal wavelet constructions," in *Wavelet Applications in Signal and Image Processing III*, A. F. Laine and M. Unser, Eds., vol. 2569 of *Proceedings of SPIE*, pp. 68–79, 1995.

[12] A. Eiben and J. Smith, *Introduction to Evolutionary Computing*, Springer, 2008.

[13] H. Beyer and H. Schwefel, "Evolution strategies. A comprehensive introduction," *Natural Computing*, vol. 1, no. 1, pp. 3–52, 2002.

[14] N. Hansen, "The CMA evolution strategy: a comparing review," in *Towards a New Evolutionary Computation*, pp. 75–102, 2006.

[15] R. R. Coifman and M. V. Wickerhauser, "Entropy-based algorithms for best basis selection," *IEEE Transactions on Information Theory*, vol. 38, no. 2, pp. 713–718, 1992.

[16] S. G. Mallat and Z. Zhang, "Matching pursuits with time-frequency dictionaries," *IEEE Transactions on Signal Processing*, vol. 41, no. 12, pp. 3397–3415, 1993.

[17] M. M. Lankhorst and M. D. van der Laan, "Wavelet-based signal approximation with genetic algorithms," in *Evolutionary Programming*, pp. 237–255, 1995.

[18] R. L. Claypoole, R. G. Baraniuk, and R. D. Nowak, "Adaptive wavelet transforms via lifting," in *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 1513–1516, May 1998.

[19] G. Piella and H. J. A. M. Heijmans, "Adaptive lifting schemes with perfect reconstruction," *IEEE Transactions on Signal Processing*, vol. 50, no. 7, pp. 1620–1630, 2002.

[20] U. Grasemann and R. Miikkulainen, "Effective image compression using evolved wavelets," in *Proceedings of the Conference on Genetic and Evolutionary Computation (GECCO '05)*, pp. 1961–1968, ACM, New York, NY, USA, 2005.

[21] B. Babb and F. Moore, "The best fingerprint compression standard yet," in *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics (SMC '07)*, pp. 2911–2916, October 2007.

[22] B. Babb, F. Moore, M. Peterson, T. H. O'Donnell, M. Blowers, and K. L. Priddy, "Optimized satellite image compression and reconstruction via evolution strategies," in *Evolutionary and Bio-Inspired Computation: Theory and Applications III*, vol. 7347, Orlando, Fla, USA, May 2009, 73470O.

[23] B. J. Babb, F. W. Moore, and M. R. Peterson, "Improved multiresolution analysis transforms for satellite image compression and reconstruction using evolution strategies," in *Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference: Late Breaking Papers*, pp. 2547–2552, ACM, Montreal, Canada, 2009.

[24] F. Moore and B. Babb, "A differential evolution algorithm for optimizing signal compression and reconstruction transforms," in *Proceedings of the Conference Companion on Genetic and Evolutionary Computation (GECCO '05)*, pp. 1907–1912, ACM, Atlanta, Ga, USA, 2008.

[25] F. Moore, P. Marshall, and E. Balster, "Evolved transforms for image reconstruction," in *Proceedings of the IEEE Congress on Evolutionary Computation*, vol. 3, pp. 2310–2316, September 2005.

[26] B. Babb, S. Becke, and F. Moore, "Evolving optimized matched forward and inverse transform pairs via genetic algorithms," in *Proceedings of the IEEE International 48th Midwest Symposium on Circuits and Systems (MWSCAS '05)*, vol. 2, pp. 1055–1058, August 2005.

[27] F. Moore, "A genetic algorithm for evolving multi-resolution analysis transforms," *WSEAS Transactions on Signal Processing*, vol. 1, pp. 97–104, 2005.

[28] F. Moore and B. Babb, "Revolutionary image compression and reconstruction via evolutionary computation, part 2: multiresolution analysis transforms," in *Proceedings of the 6th WSEAS International Conference on Signal, Speech and Image Processing*, pp. 144–149, World Scientific and Engineering Academy and Society (WSEAS), Lisbon, Portugal, 2006.

[29] R. Salvador, F. Moreno, T. Riesgo, and L. Sekanina, "Evolutionary design and optimization of wavelet transforms for image compression in embedded systems," in *Proceedings of the NASA/ESA Conference on Adaptive Hardware and Systems (AHS '10)*, pp. 171–178, IEEE Computer Society, June 2010.

[30] J. D. Villasenor, B. Belzer, and J. Liao, "Wavelet filter evaluation for image compression," *IEEE Transactions on Image Processing*, vol. 4, no. 8, pp. 1053–1060, 1995.

[31] Z. Vasicek and L. Sekanina, "An evolvable hardware system in Xilinx Virtex II Pro FPGA," *International Journal of Computer Application*, vol. 1, no. 1, pp. 63–73, 2007.

[32] A. R. Calderbank, I. Daubechies, W. Sweldens, and B. L. Yeo, "Wavelet transforms that map integers to integers," *Applied and Computational Harmonic Analysis*, vol. 5, no. 3, pp. 332–369, 1998.

[33] M. Martina, G. Masera, G. Piccinini, and M. Zamboni, "A VLSI architecture for IWT (Integer Wavelet Transform)," in *Proceedings of the 43rd IEEE Midwest Symposium on Circuits and Systems*, vol. 3, pp. 1174–1177, August 2000.

[34] M. Grangetto, E. Magli, M. Martina, and G. Olmo, "Optimization and implementation of the integer wavelet transform for image coding," *IEEE Transactions on Image Processing*, vol. 11, no. 6, pp. 596–604, 2002.

[35] T. E. Oliphant, "Python for scientific computing," *Computing in Science and Engineering*, vol. 9, no. 3, Article ID 4160250, pp. 10–20, 2007.

[36] J. D. Hunter, "Matplotlib: a 2D graphics environment," *Computing in Science and Engineering*, vol. 9, no. 3, Article ID 4160265, pp. 99–104, 2007.

[37] D. Maio, D. Maltoni, R. Cappelli, J. L. Wayman, and A. K. Jain, "FVC2000: fingerprint verification competition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 3, pp. 402–412, 2002.