Accelerating FPGA-based evolution of wavelet transform filters by optimized task scheduling

Ruben Salvador , Alberto Vidal , Felix Moreno , Teresa Riesgo , Lukas Sekanina

A B S T R A C T

Adaptive embedded systems are required in various applications. This work addresses these needs in the area of adaptive image compression in FPGA devices. A simplified version of an evolution strategy is utilized to optimize wavelet filters of a Discrete Wavelet Transform algorithm. We propose an adaptive image compression system in FPGA where optimized memory architecture, parallel processing and optimized task scheduling allow reducing the time of evolution. The proposed solution has been extensively evaluated in terms of the quality of compression as well as the processing time. The proposed architecture reduces the time of evolution by 44% compared to our previous reports while maintaining the quality of compression unchanged with respect to existing implementations. The system is able to find an optimized set of wavelet filters in less than 2 min whenever the input type of data changes.

1. Introduction

One of the current design challenges in embedded systems engineering is the implementation of adaptation capabilities. Using previous compression standards like JPEG, which relied in the Discrete Cosine Transform (DCT) for its transform stage prevented from implementing this adaptation at the transform level. However, JPEG 2000 [1] switched to the Discrete Wavelet Transform (DWT) [2], which opened up a very interesting possibility for this task to be tackled.

DWT 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 performance improvement of the transform according to each particular type of image so that improved compression (in terms of quality *vs* size) can be achieved, depending on the wavelet used. Besides, the proposal of the *Lifting Scheme* by Sweldens [3] widened the possibilities of the DWT by making the custom construction of wavelets possible with this computation scheme.

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.

Most of the various approaches previously followed by other authors in the search for this transform adaptivity are based on the mathematical foundations of wavelets and multi-resolution analysis. In contrast, a new line of research was opened in previous works by Grasemann and Miikkulainen [4] and Moore [5] which makes use of bio-inspired algorithms, such as Evolutionary Algorithms (EAs), as a design/optimization tool to help to find new wavelet filters adapted to specific types of input data. For this reason, it is the whole system that is being adapted and no extra computing effort is added to the transform computation algorithm, such as classical *adaptive lifting* techniques propose. Hence, the proposal focuses in the implementation of *new* ways for the automatic design of a complete new set of wavelet filters. However, these proposals demanded a huge amount of computing resources available to produce suitable results in a certainly high computing time.

Following this approach of using Evolutionary Computation to design new wavelet filters, our proposal is focused on embedding these ideas in FPGA devices, enabling the implementation of adaptive wavelet transforms in embedded systems. Therefore, our first approach dealt with the simplification of the algorithm to find a trade-off between computing effort and search performance. Once a suitable EA and its corresponding set of parameters was found, a

prototype FPGA implementation was proposed which enabled self-calibration as a way to accomplish the aforementioned adaptation needs. Main system blocks are a DWT core implemented in HW and an EA, obtaining a system able to accomplish self-adaptation by optimizing wavelet filters coefficients.

After this prototype, generic, bio-inspired adaptive System On Chip (SoC) for image compression tasks was validated, a further improved system performance in terms of computing time is tackled in this work. Therefore, the goal of this paper is to propose and evaluate several techniques that will increase the level of parallelism of our previous implementation and thus allow to accelerate the time of evolution. Specifically, parallelization of the underlying computational units and an improved task scheduling and HW/SW communication model are proposed.

Hence, some basic concepts of the involved domains, DWT and EC, are introduced in Section 2, followed by an analysis of related works in Section 3. Next, a summary of the initial prototype implementation and the corresponding results are included in Section 4 to help in a better understanding of the final system and its associated improvements which can be found in Section 5, which features the description of the optimized architecture and system operation scheduling. Section 6 concludes the paper.

2. Overview of related concepts

2.1. Discrete Wavelet Transform

The DWT is a multi-resolution analysis (MRA) tool widely used in signal processing due to its joint time–frequency signal analysis characteristics that concentrates the signal energy into fewer coefficients to increase the degree of compression when the data is encoded. For a general introduction to wavelet based MRA analysis see [6].

The *Lifting Scheme* (LS), introduced by Sweldens [3], reduces the computational cost of the transform as required by the Fast Wavelet Transform (FWT) algorithm and facilitates the construction of custom wavelets for very specific and different types of data. Besides, it 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, what guarantees perfect reconstruction [3,7].

Fig. 1 shows the basic LS, which consists of three stages (also called lifting steps): ***Split*** (also called *Lazy Wavelet*) divides the input data into two smaller subsets, s_{j-1} and d_{j-1} which usually correspond with the even and odd samples. The ***Predict*** and ***Update*** stages, also called *lifting filters*, are computed as in (1). Although not shown in Fig. 1, at the end of each transform level two normalization factors are defined by the Lifting Scheme if, for example, energy conservation is intended [8]. An advantage of the Lifting Scheme is that it defines a perfectly invertible transform by just doing a reversal of the forward transform operations order (Update, Predict, *Merge*) and a simple swap of plus and minus signs.

$$\begin{aligned} d_{j-1}(z) &= d_j(z) + P(z)s_j(z) \\ s_{j-1}(z) &= s_j(z) + U(z)d_j(z) \end{aligned} \tag{1}$$

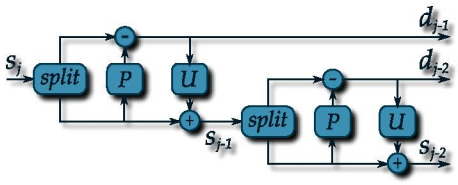

Fig. 1. Lifting Scheme (j stands for the decomposition level).

Hence, according to Fig. 1 the wavelet representation of s_j is 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 $\{s_{-n}, d_{-n}, \ldots, d_{-1}\}$. An algorithmic description of the LS can be expressed as in Algorithm 1. A different notation for the transform coefficients is also often used; for a 2 level image decomposition it becomes $\{LL, LH, HL, HH\}$, where L stands for low pass (data trend) and H for high pass (data details) coefficients respectively.

Algorithm 1. Lifting Scheme

for $j \leftarrow 1,n$ **do** ▷ j stands for the decomposition level
  $\{s_j, d_j\} \leftarrow Split(s_{j+1})$
  $d_j = d_j + P(s_j)$
  $s_j = s_j + U(d_j)$
end for

2.2. Bio-inspired optimization with Evolutionary Computation

Evolutionary Computation (EC) [9] is a sub-field 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) is one of the fundamental algorithms among Evolutionary Algorithms (EA) that utilize a population of candidate solutions and bio-inspired operators to search for a target solution. ES is primarily used for optimization of real-valued vectors. The algorithm operators are iteratively applied within a loop, where each loop run is called a *generation* (g), until a termination criterion is met.

So-called variation operators (mutation and recombination) create the necessary diversity and thereby facilitate novelty while *selection* acts as a force pushing quality since individuals are selected according to the *fitness* figure scored in the *evaluation* phase. *Mutation* delivers a (slightly) modified mutant causing a random, unbiased change, while *recombination* merges (random) information from two (or more) parents. 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 2 shows a pseudo-code description of a typical ES.

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), $\rho \leqslant \mu$ 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*. Selection is based on the ranking of the individuals' fitness ($\mathcal{F}$) (which measures the performance of that solution) taking the μ best individuals. 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 (and are therefore being adapted themselves) with the solution. This is a particular feature of ES called self-adaptation. For a general description of the $(\mu/\rho \dagger, \lambda)$-ES see [10].

Algorithm 2. $(\mu/\rho \overset{+}{,} \lambda)$-ES

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** $l \leftarrow 1, \lambda$ **do**
6: $\mathcal{R} \leftarrow$ Draw ρ parents from $P_{\mu}^{(g)}$
7: $\mathbf{r_l} \leftarrow recombine(\mathcal{R})$
8: $(\tilde{\mathbf{y}}_\mathbf{l}, \tilde{\mathbf{s}}_\mathbf{l}) \leftarrow mutate(\mathbf{r_l})$
9: $\mathcal{F}_l \leftarrow evaluate(\tilde{\mathbf{y}}_\mathbf{l})$
10: **end for**
11: $P_{\lambda}^{(g)} \leftarrow \{(\mathbf{y_l}, \mathbf{s_l}),\ l = 1, \ldots, \lambda\}$
12: $P_{\mu}^{(g+1)} \leftarrow selection\left(P_{\lambda}^{(g)}, P_{\mu}^{(g)}, \mu, \overset{+}{,}\right)$
13: $g \leftarrow g + 1$
14: **end while**

3. Previous work on evolution in FPGAs and Evolved Wavelets

As analysed in the previous sections, this work involves three main disciplines, i.e., Evolutionary Computation, Image Processing and FPGAs as implementation platforms. Therefore, the review of the State of the Art will focus on a somehow combined analysis of this three different subjects.

3.1. Evolvable Systems in FPGAs

Regarding the implementation of Evolvable Systems in FPGAs, this typically serves either in the fitness calculation task or as a single-chip evolvable platform. In the former case, the FPGA is configured externally by a computer (or any other system such as a Digital Signal Processor) where the genetic operations are carried out. Table 1 provides numerous examples of FPGA implementations of digital evolvable systems. One can identify the following components in all systems; the array of reconfigurable elements, evolutionary algorithm (i.e., genetic operations), fitness calculation unit and controller.

The problem domain determines the type of reconfigurable elements. In some cases the evolution is performed directly with reconfigurable cells of the FPGA (e.g., at the level of frames); in other cases a kind of virtual reconfigurable circuit (VRC) [11], which is an application specific reconfiguration layer featuring application specific programmable elements built on top of the FPGA is utilized. The EA and fitness calculation unit can be implemented either as application specific circuits or as software running either in a personal computer or in an on-chip embedded processor. Recent implementations have employed native reconfiguration provided by the Internal Configuration Access Port (ICAP) in Xilinx FPGAs. In some cases, multiple instances of fitness units have been utilized to speed up evolution. Note that the '$k \times$ HW' means in Table 1 that k instances of the fitness unit were implemented.

This quick overview on evolvable systems in FGPAs gives a hint on the various approaches followed by the research community in the quest of two main objectives. First, the use of FPGAs to accelerate the evolution of digital circuits (as a machine-guided design process rather than human engineered) so that the proposed solution is later on implemented in the final system. And secondly, the use of the FPGA as the evolvable platform itself. In Table 1 a distinction is made on the basis of *external* or *internal* reconfiguration. The former approach only allows for the first objective, since the EA is implemented outside the final system. By the contrary, the latter approach, which involves conferring the system the ability to reconfigure itself somehow, also builds the foundations for self-adaptive systems. However, most implementations up to date which make use of internal reconfiguration have just dealt mainly with the acceleration of the evolutionary design process.

3.2. Evolutionary Wavelet Design

In the Introduction Section it was already proposed that the way to accomplish adaptivity within the scope of this work does not use the mathematical foundations of wavelets, which falls into the known as *adaptive lifting* field. This approach mainly involves

Table 1
FPGA implementations of evolvable digital systems.

References	Application	Platform	EA	Fitness
External reconfiguration				
[12]	Tone discriminator	XC6216 logic	PC	PC
[13]	Oscillators	XC6216 logic	PC	PC
[14]	Sorting networks	XC6216 logic	PC	HW
[15]	Arithmetic circuits	Virtex CLB	PC	PC
[16]	Image filters	VRC @ XCV1000	HW	HW
[17]	IIR filters	VRC @ XCV600E	DSP	DSP
[18]	FT[a] arith. circuits	Virtex II Pro logic	PC	PC
Internal reconfiguration				
[19]	FIR filters	Register values	HW	HW
[20]	Logic circuits	VRC @ XC2V3000	HW	HW
[21]	Image filters	VRC @ XC2V3000	HW	HW
[22]	Hash functions	VRC @ XC4VFX20	HW	HW
[23]	Cellular automaton	Virtex II CLB	MicroBlaze	MicroBlaze
[24]	Image filters	VRC @ XC2VP50	PowerPC	HW
[25]	CGP accelerator	VRC @ XC2VP50	PowerPC	HW
[26]	Face recognition	VRC @ XC2VP30	MicroBlaze	HW
[27]	Sonar spectrum class.	VRC @ XC2VP30	PowerPC	HW
[28]	Arith. circuits	VRC @ XCV2000E	HW	$2 \times$ HW
[29]	Image filters, classif.	VRC @ XCV2000E	HW	HW
[30]	Const. Coeff. Mult.	VRC @ XC2VP50	HW	HW
[31]	CGP accelerator	VRC @ XC5VFX100T	PowerPC	$4 \times$ HW
[32]	Small comb. circuits	Virtex 4 logic	PowerPC	$16 \times$ HW
[33]	Image compression	Register values	PowerPC	HW

[a] Fault Tolerant.

the adaptation of the transform to the local properties of the signal *on the fly*, which implies an extra computational effort to detect the singularities of the signal and, afterwards, apply the transform itself.

By the contrary, our objective is to obtain a complete new set of filters using EC techniques to build up a new wavelet transform, which will eventually be adapted to a specific type of signal. Therefore, the general Lifting Scheme still applies, which has the advantage of keeping the computational complexity of the transform at a minimum (just as defined by the LS). This is the reason why this first review of the State of the Art concentrates on *evolutionary design of wavelet filters*.

This work is a continuation of [34], which uses the original idea of combining the lifting technique with EA for designing wavelets proposed by Grasemann and Miikkulainen [35]. Their original contributions are two; the use of a Genetic Algorithm (GA) to encode wavelets as a sequence of lifting steps and the proposal of an idealized version of a transform coder to save time in the complex evaluation method used (which involved computing a number of times the Peak Signal to Noise Ratio (PSNR) for one individual combined with other individuals from each of one of the parallel subpopulations). They propose using only a certain percentage of the largest coefficients (which involves a previous ordering stage) for reconstruction.

The GA used had parallel evolving populations (coevolutionary GA). The evaluation consisted of 80 runs, each of which took approximately 45 min on a 3 GHz Xeon processor. The results obtained in this work outperformed the considered state-of-the-art wavelet for fingerprint image compression, the D9/7 wavelet used by the FBI, in 0.75 dB. The type of images used to adapt the wavelet to is the set of 80 images from the FVC2000 fingerprint verification competition [36].

Works reported by Babb, Moore and co-workers can be considered the current state of the art in the use of EC for image transform design [37–40]. After using a GA algorithm in their previous works, the authors finally propose the use of an ES, outperforming their previous results, but keep on encoding wavelets as filters for the FWT, instead of the LS. The milestones followed in their research are summarized in the next list.

1. Evolve the inverse transform for digital photographs under conditions subject to quantization.
2. Evolve *matched* forward and inverse transform pairs.
3. Evolve coefficients for three and four level MRA transforms.
4. Evolve a different set of coefficients for each level of MRA transforms.

Table 2 shows the most remarkable and up to date published results in the design of wavelet transforms using EC. 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 [35]. The algorithms reported in works [37–39] are highly computationally intensive, so the training runs had to be done using supercomputing resources, available through the use of the Arctic Region Supercomputer Center (ARSC) in Fairbanks, Alaska. Although the work by Grasemann and Miikkulainen [35] was done on an *accessible* Intel Xeon based PC, both training times and computing resources needed in all works analysed in Table 2 show the complexity of the algorithms developed. The use of these powerful computing 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.

4. Prototype adaptive system implementation

This Section introduces a system level overview of the proposal, analysing the design steps which led to [41], which features a thorough validation of the proposed system and its embedded EA. Therefore, a summary of our previously reported work is included here to serve as an introduction to the HW implementation accomplished afterwards.

4.1. System level overview. Design partitioning

Fig. 2 shows a high-level flow chart of the proposed EA together with an abstract view of the system to show the whole idea of this work: *let an EA find an adequate set of coefficients for the lifting filters in order to maximize the wavelet transform performance from the compression point of view for a very specific type of images.*

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 tuning 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). The selected platform to host this system is a Xilinx ML507 board featuring a Virtex-5 FPGA (XC5VFX70T) with an embedded PowerPC® 440 processor.

Section 4.3 shows the results of this partitioning philosophy obtained after doing a software simulation for 500 generations. Each of the operators of the proposed EA (as introduced in following section) are analysed 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 for the next generation).

4.2. Proposed Evolutionary Algorithm

This work proposes using an ES as the search algorithm encoding the individuals (wavelets) using the LS. This is a combination of the original proposals analysed in Section 3.2. However, the use of super-computing resources and the training times needed to

Table 2
State of the Art in evolutionary wavelets design.

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

[a] Thresholding.
[b] Covariance matrix adaptation–evolution strategy.
[c] Quantization.

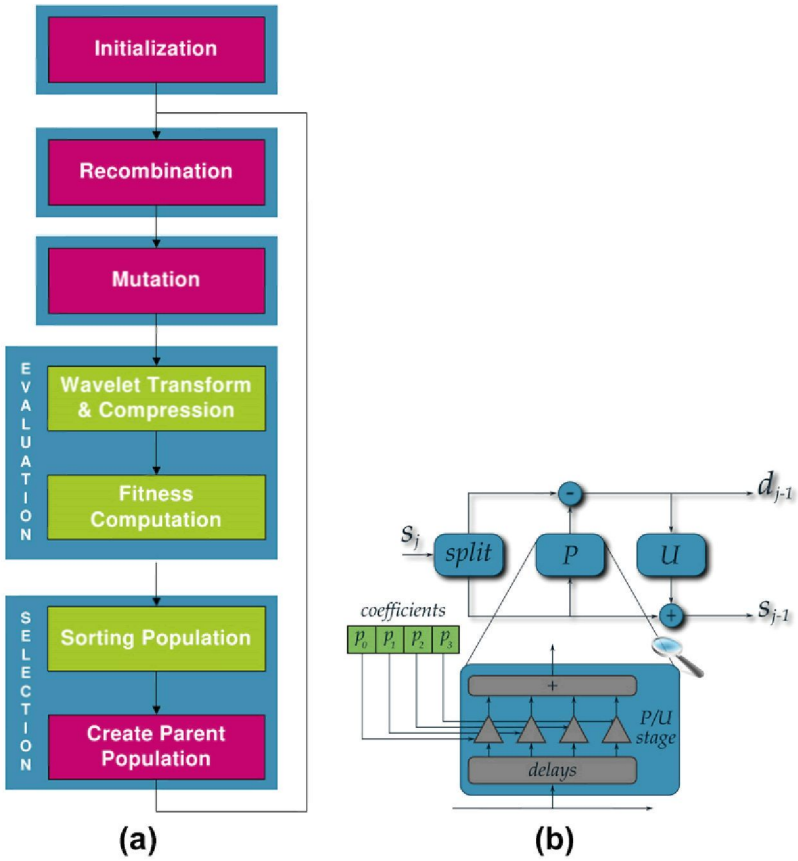

Fig. 2. (a) Flow graph of the algorithm. (b) Idea of the algorithm.

evolve state of the art performing wavelets gives an idea of the complexity of those proposals. This issue makes their implementation as a hardware embedded system highly unfeasible, which is precisely what this work addressed in the previous stages of the research, *to find an adequately tuned EA able to keep up with the quality of the transforms evolved in the State of the Art, but feasible enough to be implemented in an FPGA*. Therefore, several and severe simplifications were proposed as compared to the previous reported work. The summary of our proposals and the resulting ES are reproduced here for reading convenience, pointing to the particular conference papers where they were first presented, which yielded 1.57 dB improvement in Peak Signal to Noise Ratio (PSNR) over the D9/7 wavelet for standard fingerprint images.

The first simplifications to the algorithm [34] as compared to the previously reported State of the Art are summarized below:

1. *Single evolving population* opposed to the parallel populations of the coevolutionary genetic algorithm proposed in [35].
2. Use of *uncorrelated mutations with one step size*[10] instead of the overcomplex CMA-ES method used by Babb et al. [39,38].
3. Evolution of *one single set of coefficients for all MRA levels*.
4. *Ideal compression for the evaluation of the transform*. Since doing all the compression stages required for a *real* compression algorithm would turn out to be an unsustainable amount of computing time, the simplified evaluation method of Grasemann and Miikkulainen [35] was further improved. Therefore, 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 previous works [42] dealing with wavelet filter evaluation for image compression. For 2 levels of decomposition this severe compression is equivalent to an idealized 16:1 compression ratio.

Finally, the last two simplifications were accomplished and validated in a second stage of our work [43].

1. *Uniform random distribution*. Instead of using a Gaussian distribution for the mutation of the object parameters (see below for a description of mutation), a Uniform distribution was tested for being simpler in terms of the HW resources needed for its implementation.
2. MAE as fitness function. PSNR is the quality measure more widely used for image processing tasks. But, as previous works in image filter design via EC show [24], using Mean Absolute Error (MAE) as defined in (2) gives almost identical results because the interest lies in relative comparisons among population members.

$$\mathrm{MAE} = \frac{1}{RC}\sum_{i=0}^{R-1}\sum_{j=0}^{C-1}|I(i,j) - K(i,j)| \tag{2}$$

where R, C are the rows and columns of the image and I, K the original and reconstructed images respectively.

Table 3 gathers all the information related to the proposed ES. Candidate wavelet solutions are encoded so that each P_i, U_i lifting stage is made up of four coefficients with k_i being single scaling coefficients, which yields 26 fixed point coefficients as object parameters as described in Section 2.2. As a comparison, D9/7 wavelet is defined by $\langle P_1, U_1, P_2, U_2, k_1, k_2\rangle$.

4.3. Validation of the partitioning

Modelling and simulation of the algorithm was accomplished and reported in [44] using Python computing language [45], to-

Table 3
Proposed evolution strategy.

Parameter/operator	Value
Representation	$\langle x_1, \ldots, x_n, \sigma \rangle$ $n = 26$, fixed point coefficients
Wavelet encoding mutation[a,b]	$\langle P_1, U_1, P_2, U_2, P_3, U_3, k_1, k_2 \rangle$ $\sigma' = \sigma \cdot \exp^{\tau N(0,1)}$ $x_i' = x_i + \sigma' \cdot U_i(-1,1)$
Learning rate τ	$\tau \propto 1/\sqrt{\alpha n}, \alpha = \{1,2\}$
Evaluation	MAE
Selection	Comma
Recombination	Intermediate, $\rho = 5$
Parent population size	$\mu = 10$
Offspring population size	$\lambda = 70$
Seed for initial population	Random

[a] $N(0,1)$: draw from the standard normal distribution.
[b] $U_i(-1,1)$: separate draw from the discrete uniform distribution for each variable i.

gether with its numerical and scientific extensions, NumPy and Scipy [46], as well as the plotting library, MatPlotlib [47]. The simulation platform was a laptop computer containing an Intel Core™ 2 Duo processor at 2 GHz running Debian "Wheezy" GNU/Linux 64 bits operating system. The 16-bit fractional part for fixed-point binary arithmetic, as shown in [48,49] for 8 bits per pixel (bpp) input images, was modelled with integer types, defining the required quantization/dequantization and bit-alignment routines to mimic hardware behaviour.

Table 4 shows profiling results for 500 generations for each EA operator. 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 computing time in simulation (columns 3 and 4) 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 (forward or inverse). Obtained results validate the design partitioning proposed for the FPGA implementation (columns 5 and 6) except for the *selection* operator, which is low enough to be implemented in SW. The reason to choose a HW implementation, featuring an Insert Order Machine (IOM) as will be shown in next Section, is that it can be applied in parallel to the evaluation 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. However, in the hardware implementation some operators can be easily applied in parallel. For this reason, and depending on the scope of the system, some other operator will probably benefit from being implemented in hardware, as, for example, mutation. Besides, the subset of the C-language used to program the PowerPC processor in the FPGA imposes restrictions that will probably make that the percentage of the time each operator takes to compute increases.

4.4. FPGA implementation

The initial implementation accomplished in [33] was meant for the self-calibration of the transform stage of an image processing system. EA runs on the PowerPC processor embedded in the FPGA while the adaptive wavelet core is attached to it as a peripheral of the CoreConnect IBM PLB Bus licensed by Xilinx. Peripheral is configurable through just two registers; one for the PowerPC to send data and commands to the peripheral and one for the peripheral to communicate its status and send results back to the processor. However, this implementation was not optimized.

This prototype system implementation was just meant to be a proof of concept, aiming to validate the suitability of an FPGA implementation, so no considerable effort was made in optimizing system performance. Besides, since this work does not deal with the hardware implementation of an efficient wavelet transform architecture, it is just the validation of the system adaptivity as a whole what has been analysed. For instance, the Lifting Scheme is a direct mapping from its algorithmic description, so no optimizations at the level of data dependencies were considered (any existing lifting based wavelet implementations, as shown in [50] for example, could be used in this work, as long as the associated set of coefficients is made configurable by using registers); neither the concurrent access to the internal RAM memories so fitness computation could be made in parallel to the inverse DWT as these results were produced; and so on. The only concurrent operation taking place in the system was the *population sorting* phase, which instead of waiting until all the individuals had been evaluated, sorted them as fitnesses results were produced. This way, just some clock cycles after finishing the evaluation of the last individual, was the population sorting also finished. The description of the modules which constitute the adaptive wavelet core is as follows:

- ***DWT*** performs a DWT, which is defined as a set of lifting filter coefficients. A total of 3 Update and 3 Predict stages of 4 coefficients each can be configured (this amounts to 26 coefficients, taking into account the 2 normalization factors). Since both *lifting* filters are structurally equal, a general filtering computing stage has been designed as a direct mapping of the lifting algorithm (1). This stage can be configured to perform a Predict or an Update filter. Fig. 3 shows the implementation of the filters, which is very similar to the one reported in [51]. The full transform is obtained by cascading several Predict and Update stages (3 of each at a maximum as stated previously). In order for this generic filtering stage to fulfil the lifting algorithmic description, a configuration bit sets the operating mode of the transform, forward or inverse, which simply configures the last module in each filtering stage to be an adder or a subtractor, as shown in Section

Table 4
System execution profiling.

Operator	Further actions	Profiling		Partitioning	
% (s)	%	(s)	%	HW	SW
Recombination	–	0.14	0.009		✓
Mutation	–	0.43	0.029		✓
Evaluation	*Wavelet transform*[a]	1433.56	97.470	✓	
	Compression	4.96	0.337	✓	
	Fitness computation (Fitness)	31.62	2.150	✓	
Selection	*Sorting population (IOM)*	0.040	0.003	✓	
	Parent population				✓

[a] Computation time results for both, 2-level forward and inverse wavelet transform.

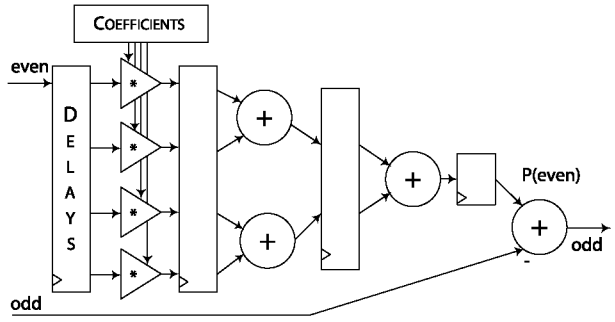

Fig. 3. Predict/update stage implementation.

2.1. Besides, another configuration bit sets whether the ideal compression proposed is done on the fly or not.

As shown in previous works [48,49] by other authors, 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 2–5-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). In this prototype implementation the datapath has been over-dimensioned to 16 bits fractional part and 10 bits integer part, for a total of 26 bits for the result of the transform.

- ***Fitness*** computes the fitness function, MAE, as in (2). Since dividing by the number of pixels *RC* just means scaling the summation value, it has not been implemented in HW to save resources. Maximum possible fitness value is $255 \times (256 \times 256) \approx 2^{8+8+8} = 2^{24}$.
- ***IOM*** (Insert Order Machine) sorts the population individuals according to their fitnesses as they are evaluated. It keeps track of the particular individual each fitness value belongs to so that when the processor retrieves the result of the evaluation, the values sent are composed of the tuple $\langle fitness\_value, individual\_id \rangle$. Since maximum fitness value needs 24 bits for its representation, this tuple can be packed in a single 32 bit transfer for populations of up to 256 individuals.
- ***Communications IF*** is responsible of interfacing with the bus-based system and decoding its commands.
- ***Control*** is the global control of the peripheral. Communicates with the Communications IF driving the rest of the peripheral according to the commands decoded.
- ***Memory and memory controller***. The internal peripheral memory is implemented with the BRAM blocks available in the FPGA. The two instantiated memories have been designed to host:
  - The original image (pixels are 8 bit wide integers).
  - Subsequent transformation results
    - The result of the DWT plus the compression if configured (26-bit wide fixed point format as explained previously).
    - The reconstructed image (whose effective bit width is actually 8).

- ***Ser/Des*** (Serializer/Desearializer). Serializer splits the 32 bits data bus into 4 8-bit wide pixels. The opposite operation is performed by the Deserializer.

The system operation phases were clearly defined in the peripheral by the different operating modes it supports, coded as a State Machine in the control module. A communications IF module was in charge of decoding the configuration commands sent by the microprocessor to the peripheral, which set it into a different operating mode according to the FSM commands driven by the Control module. As a summary, these modes dealt with image transfers from/to the Compact Flash and the internal memory of the peripheral (*IMG_RCV* and *IMG_BACK*); configuration of the peripheral to a given wavelet by sending it the corresponding coefficients in the correct order to set up a forward or inverse transform (*CFG_WV*); computation of a 1-dimension, 1-level forward or inverse DWT or ideal compression (*IMG_OP*); fitness computation (*INDIV_FIT*); and retrieval of the evaluation results (*POP_FIT*);

4.4.1. Notes on the implementation of the ES

Section 4.2 described the ES implemented in the PowerPC processor. Regarding ***mutation*** for real-valued search spaces in ESs, it 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). In this work it has been implemented using the standard C-based *rand ()* and *srand ()* functions, which yield a Uniform distribution in the range $\langle 0 \ldots RAND\_MAX \rangle (RAND\_MAX = 2{,}147{,}483{,}647)$ and seed the generator, respectively. To obtain a Normal distribution $N(0,1)$ two Uniform distributions U,V are needed according to the Marsaglia Polar Method [52], described below:

$$z_1 = \pm\sqrt{-2 \times \log(R)} \times \frac{U}{R}$$
$$z_2 = \pm\sqrt{-2 \times \log(R)} \times \frac{V}{R} \quad (3)$$

being U, V two independent uniform distributions $U(0,1)$ and $R = U^2 + V^2$. With this method z_1 and z_2 are obtained which follow two standard Normal distributions.

4.5. Performance results

Adaptation results were demonstrated in [33], and, since the scope of this paper is accelerating the time of evolution, previous timing results are reproduced here to clarify reading. Measured latency of the DWT module is 56 clock cycles. Therefore, for the size of images used for evolution, the number of clock cycles needed to compute the first forward transform level, s_{j-1}, is:

$$((256 \times 256) + 56) \times 2 = 131{,}184$$

Equivalently, for the second transform level s_{j-2}, where the smaller (compared to the original input s_j) subset s_{j-1}, sized 128×128, is used as input, the number of clock cycles needed for its computation is 32,880. This means that a whole 2-level forward DWT takes 131,184 + 32,880 = 164,064 clock cycles. The same analysis can be applied for the 2-level inverse DWT, since the computation stages are right the same. As for the compression module, it needs to retrieve the whole image from memory to perform its operation, which yields $256 \times 256 = 65{,}536$ clock cycles. The same applies for the fitness module.

Based on this previous analysis, it can be concluded that the evaluation of one individual takes 164,064 + 65,536 + 164,064 + 65,536 = 459,200 clock cycles. If system frequency is set to 100 MHz, 4.592 ms are needed for the evaluation of a single candidate wavelet. System is let to evolve during 1000 generations, doing 70 evaluations per generation, which means 70,000 evaluations are done. Hence, a time t_{eval} of around 5.34 min is needed to perform all evaluations.

Previous timing analysis for t_{eval} applies to the adaptive wavelet core timing, i.e., *hardware* time. However, ES running in the PowerPC processor will add more time to the evolution, mainly due to the computation of random values and to the exponential function involved in the mutation operator as well as to the communication overhead caused by HW/SW communication. This time, t_{EA}, or *software* time, has been measured to be around 3.5 min. Therefore, if

evolution time is defined as $t_{evo} = t_{eval} + t_{EA}$, approximately 9 min are needed by the system to complete the 70,000 evaluations.

5. Improving system performance

In this Section, the various optimizations addressed to improve system performance are analysed. Fig. 4 shows the optimized system architecture, which, as opposed to the prototype (as shown in [33]), splits the internal modules of the peripheral to allow for an enhanced control strategy and HW/SW communication as will be analysed below. The functional description of the modules in Section 4.4 applies to this enhanced architecture.

It can be observed in Fig. 4 that there is a direct connection of each module to the PLB Bus, which simplifies the control strategy, defining some more registers for commanding the peripheral.

- ***Control***.
  - *Command Reg (W).* Sets the peripheral into one of the operating modes defined.
  - *Status Reg (R).* Status of the peripheral may be *active* if a previous command is still being executed or *idle* if it is already finished.
- ***DWT***.
  - *Config Reg (W).* A given wavelet transform (forward and inverse) is configured into the peripheral.
- ***Fitness/IOM***.
  - *Configuration Reg (W).* Number of sorted individuals to be read back to the processor.
  - *Data Reg (R).* Tuple ⟨*fitness_value*, *individual_id*⟩ representing the evaluation of an individual.
- ***Memory***.
  - *Data Reg (W).* Image write register (Compact Flash → Peripheral image transfer).
  - *Data Reg (R).* Image read register (Peripheral → CompactFlash image transfer).

The optimizations accomplished try to reduce the time needed to perform evolution, $t_{evo} = t_{eval} + t_{EA}$, which can be decomposed as an equivalent timing of $t_{evo} \equiv t_{HW} + t_{SW} + t_{HW/SW}$, where $t_{HW/SW}$ stands for the HW/SW communication overhead. Following subsections address the different optimizations accomplished, which deal with the concurrent execution of the evaluation tasks (*Opt1*, which affects HW related timing), an improvement on the HW/SW communication which reduces the number of times the processor needs to access the peripheral (*Opt2*, which affects HW/SW related timing) and the pre-calculation of the random numbers and exponential functions which define the mutation steps (*Opt3*, which affects SW related timing). Once each improvement is analysed, the measured saved time is reported independently of the others so at the end all of them are considered together and the overall performance improvement is reported.

5.1. Concurrent operation of the evaluation tasks (Opt1)

The architectural description and the timing analysis of previous Section clearly show that a great amount of time could be saved if a better memory architecture and some degree of pipelining at the level of HW tasks was accomplished. Previous system architecture forced to schedule the tasks needed for evaluation in a sequential manner, i.e., forward DWT (fDWT); compression (Comp); inverse DWT (iDWT); and fitness calculation (Fitness).

This situation is shown in the top schedule (grayed out) of Fig. 5. However, as soon as the first results of the details bands of the first transform level are produced (indicated as a grey-dotted box inside f/iDWT), compression may be triggered. Since compression just replaces the resulting transform coefficients by zeros, one clock cycle after fDWT is finished compression will be finished too. The same analysis can be applied to the iDWT. In this case, after the first inverse transform level is finished, fitness unit may begin its operation just some clock cycles afterwards, right when the first final results of the inverse transform are produced.

The result of this enhanced scheduling strategy, due to an improvement in the pipeline of the system, reduces the time needed for the evaluation of the candidate wavelets, as shown in Fig. 5, where the configuration of a new wavelet has been advanced some time as compared to the non-optimized version. Specifically, hardware related evaluation time is reduced by an approximate 29 %, since now $t_{HW} \equiv t_{eval} \approx 164{,}064 + 164{,}064 = 328{,}128$ clock cycles.

5.2. Improved HW/SW communication (Opt2)

The internal structure of the prototype adaptive wavelet core demanded an overloaded HW/SW communication, since every single operation had to be commanded from the processor. As instance, to perform an n-level forward or inverse DWT, $2 \times n$ *IMG_OP* commands had to be issued (factor 2 is due to the fact that each issued command performed a 1-direction transform, just over rows or columns), being the control module responsible of taking

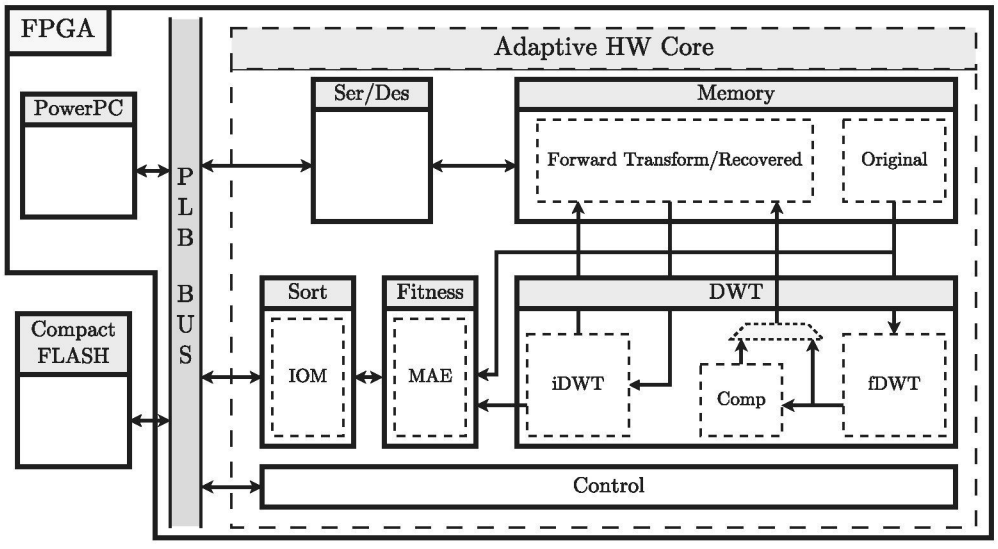

Fig. 4. System level architecture. Although fDWT and iDWT share the same HW, they are separated in the diagram for an improved functional understanding.

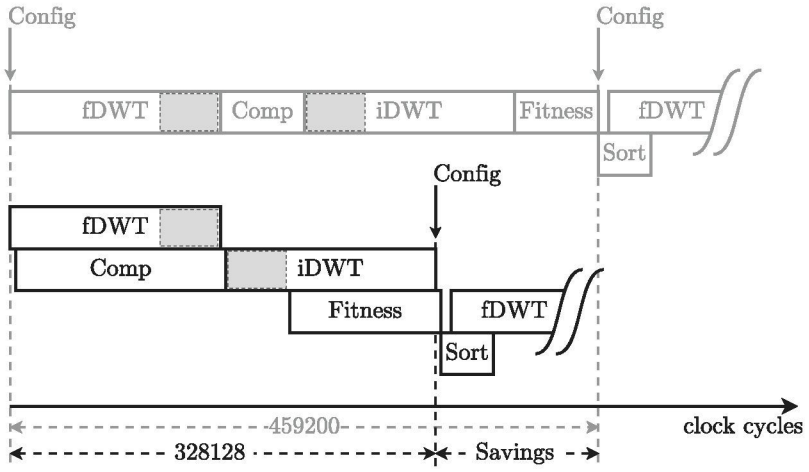

Fig. 5. Comparison of the HW scheduling of the system for the evaluation of one individual. Upper (grey) scheduling corresponds to the prototype while the lower (black) corresponds to the optimized system implementation.

care of the current transform level. Besides, to configure a given wavelet, forward or inverse transform coefficients had to be sent to the peripheral in two separate phases while with this new communication model forward and inverse wavelet configuration is a one step process.

Therefore, to perform the evaluation of one individual using 2-level DWTs, 2 wavelet configurations, 4×2 *IMG_OP* commands, an extra *IMG_OP* command for compression and one *INDIV_FIT* command had to be issued. However, only one wavelet configuration and one *IMG_OP* command are now needed to perform a single evaluation. This reduction in the number of commands issued by the microprocessor, though probably negligible for a single evaluation, is for sure important when 70,000 evaluations are performed. Besides, a cleaner code is achieved in the processor since a simplified system control strategy is obtained.

Expected time reduction is difficult to be estimated since it involves taking into account the overhead introduced by the arbitration policies of the PLB system bus as well as the transfers themselves. Therefore, actual performance gain is measured using a peripheral timer attached to the PowerPC to profile the elapsed clock cycles. Next Section reports these results.

5.3. Pre-computation of mutation steps (Opt3)

One of the most time consuming tasks in the system is the mutation operator, which can be found in Table 3. As it can be derived, to generate a new individual after recombination of its parents several random numbers and further computations with them need to be done. To be precise, the following calculations have to be performed:

- One draw from the standard normal distribution $N(0, 1)$.
- Computation of the term $\Delta\sigma = \exp^{\tau \cdot N(0,1)}$.
- One separate draw from the discrete uniform distribution $U_i(-1, 1)$ for each i (object parameter), i.e., 26 draws.

All of these calculations (mutation steps) can be scheduled concurrently with the evaluation of the λ (70) individuals, so that when a new population has been completely evaluated, the mutation of the strategy and object parameters can be finished. Therefore, mutation can be rewritten as:

$$\Delta\sigma = \exp^{\tau \cdot N(0,1)} \tag{4a}$$

$$\Delta\overline{U} = U_i(-1, 1) \tag{4b}$$

$$\sigma' = \sigma \cdot \Delta\sigma \tag{4c}$$

$$x_i' = x_i + \sigma' \cdot \Delta\overline{U} \tag{4d}$$

In this case, mutation time can be decomposed as $t_{mut} = t_{mut\_steps} + t_{mut\_op}$, where *mut_steps* (4a) and (4b) are computed concurrently with the evaluation phase and *mut_ops* (4c) and (4d) straight afterwards recombination (which follows evaluation and selection) to complete the process of the mutation operator. This situation is shown in Fig. 6. Again, the exact number of cycles will be measured in-system after implementation, but it is expected that the pre-calculation of the mutation steps takes a reasonably shorter time than the evaluation of the 70 individuals.

5.4. Performance results of the optimized system architecture

After synthesis, implementation and verification of the system in the FPGA according to the prototype working version, a valid frequency of 100 MHz is still achieved as shown in Table 6. Since this paper deals with acceleration of the evolution time, it is just mentioned here how the FPGA is able to host such a system in terms of available resources, while the following analysis concentrates on the related timing issues.

As it was analysed in Section 5.1 *Opt1* saves 28.5% of the time used for evaluation, which, for the 70,000 evaluations considered and using a 100 MHz clock, means saving 1.52 min (from 5.34 down to 3.82 min). This situation is shown in Table 7, which gives the profiling results of the optimized system implementation. Results for the prototype and optimized system are shown for the different timings considered when decomposing evolution time as $t_{evo} \equiv t_{HW} + t_{SW} + t_{HW|SW}$. Besides, the improvement introduced by each optimization is also shown in Table 7. This improvement, whether termed as *relative* or *absolute*, is referred to each of the time-related terms considered in t_{evo} decomposition, not to the whole evolution time which is shown in the last row of the table.

Regarding *Opt2*, from 11,372 clock cycles per candidate evaluation in the non-optimized version, which required 2 wavelet configurations and 10 commands issued from the PowerPC, 5130 clock cycles are now needed for 1 wavelet configuration and 1 command issued from the PowerPC. In this case, the improvement introduced by this optimization is not so important as that of *Opt1*, since it only means 4.36 s for the whole evolution. However, additional and subjective measures in terms of system organization, maintainability and code layout are obtained.

And last but not least, *Opt3* also introduces significant savings in computing time. Selection, recombination and mutation operators have been profiled and the results are shown in Table 5. *Selection* is performed once while *Recombination* and *Mutation* operate 70 times per generation respectively. These results validate the scheduling proposal for the pre-calculation of the mutation steps in par-

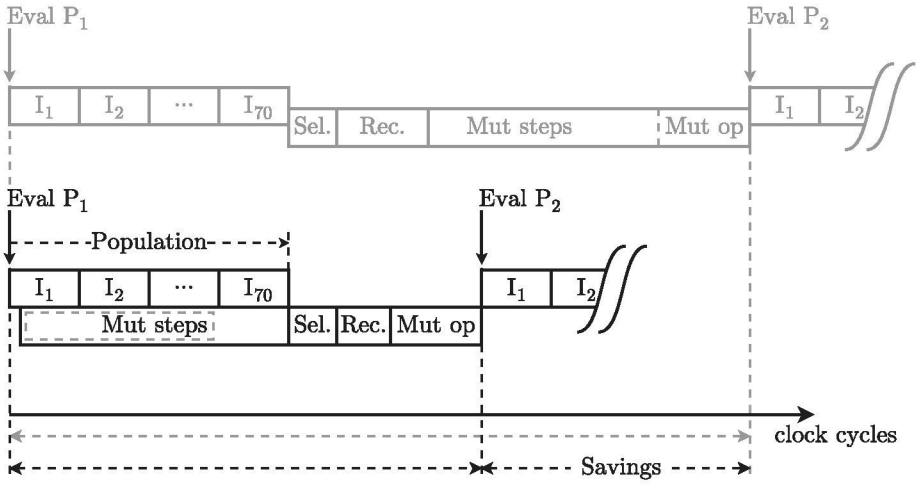

Fig. 6. Pre-calculation of the mutation steps $\Delta\sigma$ and $\Delta\overline{U}$.

Table 5
Profiling of the genetic operators for one individual evaluation.

Operator	Further actions	Clock cycles
Selection	–	7332
Recombination	–	10,191
Mutation	$\Delta\sigma$	81,000
	$\Delta\overline{U}$	63,400
	mut_ops	92

allel to the evaluation of the population, since t_{mut_steps} is around half of the evaluation time. Hence, the total time needed to create a new individual after the evaluation of the previous population is 154,788 clock cycles for the prototype system (the cycles of the selection operator have been averaged by 70 to consider only the proportional time needed for one individual). In contrast, for the optimized system, since *mut_steps* $\Delta\sigma$ and $\Delta\overline{U}$ are computed concurrently to the evaluation of the previous population, the total time required to create a new individual is reduced down to 10,388 clock cycles. This results are compiled also in Table 7, where the 93% relative improvement for this time t_{SW} is also shown.

To summarize, the different techniques implemented to speed up evolution yield a significant improvement in the overal computing time. As shown in Table 7, an improvement of around 44% is obtained, which corresponds to a total of 3.2 min. It has to be noted that the last row in Table 7 reports results computed using an embedded timer to profile the different sub-times considered, and it thus shows the expected evolution time. However, as we reported previously, around 9 min were measured for the whole evolution using a stopwatch. This difference (from 7.2 to 9 min) is due to some extra computations taking place in the system, e.g., *initialization*, where the original image stored in the Compact Flash memory as a text file is binary converted and copied in the peripheral memory (around 20 s); random creation of the initial population; and *supervision* of the evolution where some debugging outputs are sent through the attached JTAG console to check system status, among others. In the case of the optimized system this extra computations increase the time to around 5.8 min. However, this extra time would be very close to zero in a final system implementation.

Table 7
Performance results of the optimized system architecture and percentage gain compared to the prototype implementation reported in [33].

	Prototype	Optimization		Improvement	
				Relative [a]	Absolute [b]
t_{HW} [c]	459,200	*Opt1*	328,128	28.5%	1.52 min
$t_{HW/SW}$ c	11,372	*Opt2*	5130	54.8%	4.36 s
t_{SW} c	154,788	*Opt3*	10,388	93.3%	1.68 min
t_{evo} (min)	7.2	4		44%	3.2

[a] For one candidate wavelet evaluation.
[b] For the 70,000 considered evaluations @ f = 100 MHz.
[c] In clock cycles.

6. Conclusion

Our previous works proposed a self-adaptive FPGA-based architecture for image compression in embedded systems by optimizing DWT performance. The combination of EC and a reconfigurable hardware platform produces a system which is able to self-adapt to changes in the type of input data (images) being dealt with in a reasonable time. Evolved wavelet filters outperformed standard wavelets such as D9/7 by 1.57 dB (in PSNR) for standard fingerprint images.

Table 6
Implementation results for the main modules in the system.

Module	Resources				Frequency (MHz)
	Slice LUTs	Slice Registers	DSP48Es	BRAM (Kb)	
DWT	2818/44,800	4417/44,800	76/128	–	112.67
Compression	43/44,800	39/44,800	–	–	414.25
Fitness function	95/44,800	66/44,800	–	–	341.88
IOM	3920/44,800	2209/44,800	–	–	231.93
Original image memory	68/44,800	55/44,800	–	576/5328	429
Transform memory	314/44,800	120/44,800	–	1728/5328	209

In this work, an improved memory architecture, combined with several techniques that increase the level of parallelism and an optimized task scheduling was proposed to reduce the time of evolution. It is important to recall that these changes in the system architecture have not influenced the quality of the (evolved) results, which remains exactly the same as before. It is just performance that has been improved by a combined 44%, reducing total evolution time from 7.2 min to 4 min.

Although it may still be considered as a high evolution time for an adaptive system, the validation of the algorithm reported in [41] has shown how around 100 and 300 generations are enough for the system to evolve an optimized solution. Hence, the effective evolution time could be further reduced a minimum of a 50% if a conservative 500 generation ES is used. This would yield a total, conservative, evolution time of 2 min. It has to be noted that there is still room for improvement, which constitutes the possible future research directions, e.g., introducing multiple fitness units or a simplified mutation operator as well as increasing the operating frequency of the PowerPC (which is supported in this device family) and that of the HW modules by reducing their critical path.

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 number TEC2008-06486-C02-01. Lukas Sekanina was supported by the Czech Science Foundation project P103/10/1517, the research programme MSM 0021630528 and the European Regional Development Fund in the IT4Innovations Centre of Excellence project CZ.1.05/1.1.00/02.0070.

References

[1] D. Taubman, M. Marcellin, JPEG2000: Image Compression Fundamentals, 1 ed., Standards and Practice, Springer, 2001.
[2] S. Mallat, A Wavelet Tour of Signal Processing, second ed., Academic Press, 1999.
[3] W. Sweldens, The lifting scheme: a custom-design construction of biorthogonal wavelets, Applied and Computational Harmonic Analysis 3 (1996) 186–200.
[4] U. Grasemann, R. Miikkulainen, Evolving wavelets using a coevolutionary genetic algorithm and lifting, in: Genetic and Evolutionary Computation GECCO 2004, Lecture Notes in Computer Science, vol. 3, Springer, Berlin/Heidelberg, 2004, pp. 969–980.
[5] F. Moore, A genetic algorithm for evolving multi-resolution analysis transforms, WSEAS Transactions on Signal Processing, vol. 1, World Scientific and Engineering Academy and Society, 2005, pp. 97–104.
[6] B. Jawerth, W. Sweldens, An overview of wavelet based multiresolution analyses, SIAM Review 36 (1994) 377–412.
[7] W. Sweldens, The lifting scheme: a construction of second generation wavelets, SIAM Journal on Mathematical Analysis 29 (1998) 511–546.
[8] G. Uytterhoeven, Wavelets: Software and Applications, Ph.D. thesis, Department of Computer Science, K.U.Leuven, Leuven, Belgium, 1999. Roose Dirk, Bultheel Adhemar (supervisors).
[9] A. Eiben, J. Smith, Introduction to Evolutionary Computing, Springer, 2008.
[10] H. Beyer, H. Schwefel, Evolution strategies. A comprehensive introduction, Natural Computing 1 (2002) 3–52.
[11] L. Sekanina, Virtual reconfigurable circuits for real-world applications of evolvable hardware, Lecture Notes in Computer Science 2003 (2003) 186–197.
[12] A. Thompson, Silicon evolution, in: Proc. of Genetic Programming GP'96, MIT Press, 1996, pp. 444–452.
[13] L. Huelsbergen, E. Rietman, R. Slous, Evolving oscillators in silico, IEEE Transactions on Evolutionary Computation 3 (1999) 197–204.
[14] J.R. Koza, F.H. Bennett, D. Andre, M.A. Keane, Genetic Programming III: Darwinian Invention and Problem Solving, Morgan Kaufman Publishers, San Francisco, CA, 1999.
[15] T. Gordon, Exploiting Development to Enhance the Scalability of Hardware Evolution, Ph.D. thesis, Department of Computer Science, University College, London, 2005.
[16] Y. Zhang, S. Smith, A. Tyrrell, Intrinsic Evolvable Hardware in Digital Filter Design, in: Applications of Evolutionary Computing, LNCS, vol. 3005, Springer Verlag, Quimbra, Portugal, 2004, pp. 389–398.
[17] D. Gwaltney, K. Dutton, A VHDL core for intrinsic evolution of discrete time filters with signal feedback, in: Proc. of the 2005 NASA/DoD Conference on Evolvable Hardware, IEEE Computer Society, Washington, DC, USA, 2005, pp. 43–50.
[18] R.S. Oreifej, R.N. Al-Haddad, H. Tan, R.F. DeMara, Layered approach to intrinsic evolvable hardware using direct bitstream manipulation of Virtex II Pro device, in: Proc. of 2007 Conf. on Field Programmable Logic and Applications, IEEE Computer Society, 2007, pp. 299–304.
[19] G. Tufte, P. Haddow, Evolving an adaptive digital filter, in: The Second NASA/DoD workshop on Evolvable Hardware, IEEE Computer Society, Palo Alto, California, 2000, pp. 143–150.
[20] L. Sekanina, S. Friedl, An evolvable combinational unit for FPGAs, Computing and Informatics 23 (2004) 461–486.
[21] T. Martinek, L. Sekanina, An evolvable image filter: experimental evaluation of a complete hardware implementation in FPGA, in: Evolvable Systems: From Biology to Hardware, LNCS, vol. 3637, Springer Verlag, 2005, pp. 76–85.
[22] R. Salomon, H. Widiger, A. Tockhorn, Rapid evolution of time-efficient packet classifiers, in: IEEE Congress on Evolutionary Computation, IEEE CIS, Vancouver, Canada, 2006, pp. 2793–2799.
[23] A. Upegui, E. Sanchez, Evolving hardware with self-reconfigurable connectivity in Xilinx FPGAs, The 1st NASA/ESA Conference on Adaptive Hardware and Systems (AHS-2006), IEEE Computer Society, Los Alamitos, CA, USA, 2006, pp. 153–160.
[24] Z. Vasicek, L. Sekanina, An evolvable hardware system in Xilinx Virtex II Pro FPGA, International Journal of Innovative Computing and Applications 1 (2007) 63–73.
[25] Z. Vasicek, L. Sekanina, Hardware accelerators for cartesian genetic programming, in: Proceedings of the 12th European Conference on Genetic Programming, LNCS, vol. 4971, Springer Verlag, 2008, pp. 230–241.
[26] K. Glette, Design and Implementation of Scalable Online Evolvable Hardware Pattern Recognition Systems, Ph.D. thesis, University of Oslo, 2008.
[27] K. Glette, J. Torresen, M. Yasunaga, An online ehw pattern recognition system applied to sonar spectrum classification, in: Evolvable Systems: From Biology to Hardware, LNCS, vol. 4684, Springer Verlag, 2007, pp. 1–12.
[28] J. Wang, C. Piao, C. Lee, Implementing multi-VRC cores to evolve combinational logic circuits in parallel, in: Evolvable Systems: From Biology to Hardware, LNCS, vol. 4684, Springer Verlag, 2007, pp. 23–34.
[29] L.C. Wang J., Chen Q.S., Design and implementation of a virtual reconfigurable architecture for different applications of intrinsic evolvable hardware, IET Computers and Digital Techniques 2 (2008) 386–400.
[30] Z. Vasicek, M. Zadnik, L. Sekanina, J. Tobola, On evolutionary synthesis of linear transforms in FPGA, in: Proc. of the 8th Int. Conference on Evolvable Systems: From Biology to Hardware, LNCS, vol. 5216, Springer Verlag, Berlin, 2008, pp. 141–152.
[31] Z. Vasicek, L. Sekanina, Hardware accelerator of Cartesian genetic programming with multiple fitness units, Computing and Informatics 29 (2010) 1359–1371.
[32] F. Cancare, M. Santambrogio, D. Sciuto, A direct bitstream manipulation approach for virtex4-based evolvable systems, in: Proceedings of 2010 IEEE International Symposium on of Circuits and Systems (ISCAS), IEEE Computer Society, 2010, pp. 853–856.
[33] R. Salvador, A. Vidal, F. Moreno, T. Riesgo, L. Sekanina, Bio-inspired FPGA architecture for self-calibration of an image compression core based on wavelet transforms in embedded systems, Proceedings of SPIE 8067, 806704 (2011), doi./10.1117/12.887123.
[34] R. Salvador, F. Moreno, T. Riesgo, L. Sekanina, Evolutionary design and optimization of wavelet transforms for image compression in embedded systems, in: Proc. of the 2010 NASA/ESA Conference on Adaptive Hardware and Systems, IEEE Computer Society, 2010, pp. 177–184.
[35] U. Grasemann, R. Miikkulainen, Effective image compression using evolved wavelets, in: Proceedings of the 2005 Conference on Genetic and evolutionary computation, GECCO '05, ACM, New York, NY, USA, 2005, pp. 1961–1968.
[36] D. Maio, D. Maltoni, R. Cappelli, J. Wayman, A. Jain, FVC2000: fingerprint verification competition, IEEE Transactions on Pattern Analysis and Machine Intelligence 24 (2002) 402–412.
[37] B. Babb, F. Moore, The best fingerprint compression standard yet, IEEE International Conference on Systems, Man and Cybernetics ISIC (2007) 2911–2916.
[38] B. Babb, F. Moore, M. Peterson, T.H. O'Donnell, M. Blowers, K.L. Priddy, Optimized satellite image compression and reconstruction via evolution strategies, Evolutionary and Bio-Inspired Computation: Theory and Applications III, vol. 7347, SPIE, Orlando, FL, USA, 2009. pp. 73470O–10.
[39] B.J. Babb, F.W. Moore, 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, ACM, Montreal, Quebec, Canada, 2009, pp. 2547–2552.
[40] F.W. Moore, B. Babb, A differential evolution algorithm for optimizing signal compression and reconstruction transforms, Proceedings of the 2008 GECCO Conference Companion on Genetic and Evolutionary Computation, ACM, Atlanta, GA, USA, 2008, pp. 1907–1912.
[41] R. Salvador, F. Moreno, T. Riesgo, L. Sekanina, Evolutionary approach to improve wavelet transforms for image compression in embedded systems, EURASIP Journal on Advances in Signal Processing 2011 (2011) 1–20. Article ID 973806.

[42] J. Villasenor, B. Belzer, J. Liao, Wavelet filter evaluation for image compression, IEEE Transactions on Image Processing 4 (1995) 1053–1060.
[43] R. Salvador, F. Moreno, T. Riesgo, L. Sekanina, High level validation of an optimization algorithm for the implementation of adaptive wavelet transforms in FPGAs, in: 13th Euromicro Conference on Digital System Design: Architectures, Methods and Tools (DSD), 2010, pp. 96–103.
[44] R. Salvador, F. Moreno, T. Riesgo, L. Sekanina, Implementation of bio-inspired adaptive wavelet transforms in FPGAs. Modelling, validation and profiling of the algorithm, in: Proceedings of the XXV Conference on Design of Circuits and Integrated Systems, DCIS, 2010, pp. 210–215.
[45] G.V. Rossum, The Python Language Reference Manual, Network Theory Ltd., 2003.
[46] T.E. Oliphant, Python for scientific computing, Computing in Science and Engineering 9 (2007) 10–20.
[47] J.D. Hunter, Matplotlib: a 2D graphics environment, Computing in Science and Engineering 9 (2007) 90–95.
[48] M. Martina, G. Masera, G. Piccinini, M. Zamboni, A VLSI architecture for IWT (integer wavelet transform), in: Proceedings of the 43rd IEEE Midwest Symposium on Circuits and Systems, vol. 3, 2000, pp. 1174–1177.
[49] M. Grangetto, E. Magli, M. Martina, G. Olmo, Optimization and implementation of the integer wavelet transform for image coding, IEEE Transactions on Image Processing 11 (2002) 596–604.
[50] T. Acharya, C. Chakrabarti, A survey on lifting-based discrete wavelet transform architectures, The Journal of VLSI Signal Processing 42 (2006) 321–339, doi:10.1007/s11266-006-4191-3.
[51] M. Martina, G. Masera, G. Piccinini, M. Zamboni, Novel JPEG 2000 compliant DWT and IWT VLSI implementations, The Journal of VLSI Signal Processing 35 (2003) 137–153, doi:10.1023/A:1023696430633.
[52] G. Marsaglia, Normal (Gaussian) random variables for supercomputers, The Journal of Supercomputing 5 (1991) 49–55, doi:10.1007/BF00155857.