Bio-inspired FPGA Architecture for Self-Calibration of an Image Compression Core based on Wavelet Transforms in Embedded Systems

Rubén Salvador[a], Alberto Vidal[a], Félix Moreno[a], Teresa Riesgo[a], Lukáš Sekanina[b]

[a]Centre of Industrial Electronics, Universidad Politécnica de Madrid
José Gutierrez Abascal, 2, 28006, Madrid, Spain

[b]Faculty of Information Technology, Brno University of Technology
Božetěchova 2, 612 66 Brno, Czech Republic

ABSTRACT

A generic bio-inspired adaptive architecture for image compression suitable to be implemented in embedded systems is presented. The architecture allows the system to be tuned during its calibration phase. An evolutionary algorithm is responsible of making the system evolve towards the required performance. A prototype has been implemented in a Xilinx Virtex-5 FPGA featuring an adaptive wavelet transform core directed at improving image compression for specific types of images.

An Evolution Strategy has been chosen as the search algorithm and its typical genetic operators adapted to allow for a hardware friendly implementation. HW/SW partitioning issues are also considered after a high level description of the algorithm is profiled which validates the proposed resource allocation in the device fabric.

To check the robustness of the system and its adaptation capabilities, different types of images have been selected as validation patterns. A direct application of such a system is its deployment in an unknown environment during design time, letting the calibration phase adjust the system parameters so that it performs efcient image compression. Also, this prototype implementation may serve as an accelerator for the automatic design of evolved transform coefficients which are later on synthesized and implemented in a non-adaptive system in the final implementation device, whether it is a HW or SW based computing device.

The architecture has been built in a modular way so that it can be easily extended to adapt other types of image processing cores. Details on this pluggable component point of view are also given in the paper.

1. INTRODUCTION

Bringing *adaptation* to embedded systems is a highly demanded feature which will contribute to solve the ever increasing demand on complexity and flexibility. Since deployment may occur in very diverse environments which may be even unknown at design time, it is very difficult to foresee all the possible situations that may arise during system's lifetime.

Therefore, if systems are let to operate on their own a self-adaptation mechanism must be implemented which responds in the presence of changes in the system's environment. Hence, not only *Autonomous Systems* would benefit from this approach, but, in general, any system (even human-operated devices) that may see its operating conditions change from those specified at design time, whether these changes refer to the input data characteristics (which depend on the environment somehow) or the system's specification itself.

This work proposes self-calibration as a way to accomplish these adaptation needs. A generic bio-inspired adaptive system for image compression tasks which is suitable to be implemented as a System On Chip (SoC) is presented so that its parameters are automatically tuned during a self-calibration phase. Main system blocks are

Further author information: (Send correspondence to R.S.)
R.S.: E-mail: ruben.salvador@upm.es, Telephone: +34 91 3363191
L.S.: E-mail: sekanina@fit.vutbr.cz, Telephone: +420 54114-1215

a Discrete Wavelet Transform (DWT) HW core and an Evolutionary Algorithm (EA) responsible of adjusting the wavelet filters coefficients.

To summarize, this work describes an FPGA-based architecture which allows a generic artificial vision system to be adapted during a self-calibration phase. This allows the system to efficiently deal with images coming from very diverse sources. Hence, it involves the automatic on-line design of a complete new set of wavelet (lifting, as introduced below) filters.

The rest of the paper is organized as follows. Following there is a short introduction to DWT and EAs as well as a short review of the current State of the Art in the topic. Section 2 introduces the general architecture of the system before Sections 3 and 4 go into the details of the proposed HW architecture and the EA running in the embedded processor respectively. Afterwards, Section 5 validates the proposal showing the results achieved before concluding the paper in Section 6.

1.1 Discrete Wavelet Transform for Image Compression

Compression standard JPEG2000[1] relies on the DWT for its transform stage, which 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 vs size) can be achieved, depending on the wavelet used.

Sweldens proposed the *Lifting Scheme* (LS) algorithm[2, 3] to compute the DWT which has also simplified the construction of custom wavelets adapted to specific and different types of data. This is the main reason why it is really well suited for the task of using an EA to encode wavelets, since any random combination of lifting filters will encode a valid wavelet which guarantees perfect reconstruction.

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.[4]

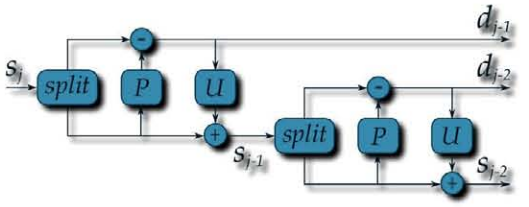


Figure 1. Lifting scheme

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:

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

The *Split* stage (also called the *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 eq. (1). Although not shown in Figure 1, at the end of each transform level two normalization factors are defined by the lifting scheme if, for example, energy conservation is intended.[5] 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.

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

As eq. (1) shows, the search for an optimum set of coefficients for the lifting filters, P and U, is the problem that needs to be solved by the EA. Besides, since the target implementation are Embedded Systems based on FPGA devices, some restrictions to the computing requirements of the algorithm need to be imposed in order to overcome the supercomputing resources needed by the works currently found in the state of the art.

1.2 Evolutionary Computation for System Optimization

Evolutionary Computation (EC)[6] 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 using a population of candidate solutions and bio-inspired operators to search for a target solution.

The algorithm operators are iteratively applied within a loop, where each 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. A general pseudo-code description of an EA is shown in Algorithm 1.

Algorithm 1 General scheme of an Evolutionary Algorithm

```
1: INITIALIZE population
2: EVALUATE each candidate
3: while not_termination_condition do
4:     SELECT parents
5:     RECOMBINE (pairs of) parents
6:     MUTATE resulting offspring
7:     EVALUATE new candidates
8:     SELECT individuals for the next generation
9: end while
```

1.3 Previous work on evolutionary wavelet design

With the introduction of the LS, custom construction of wavelets adapted to specific signal types was made possible. Therefore, instead of dealing with the mathematical construction of new wavelets, an EA can be used to design/optimize the constituent wavelet filters. Our previous work[7] features a deep review of the current State of the Art. However, some brief comments are included here for the readers' convenience.

First approach in evolutionary wavelet design was tackled by Grasemann and Miikkulainen[8,9] who made use of a Coevolutionary (subpopulations evolving in parallel) Genetic Algorithm (GA) that encodes wavelets as a sequence of lifting steps. 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. Works[10–13] reported by Babb, Moore et. al., can be considered the current state of the art, outperforming the D9/7 wavelet in 3 dB. They made use of very complex and computationally intensive algorithms, so the training runs were done using supercomputing resources. A standard set of 80 images was used for comparison purposes in these works, the same one as in this work, as will be shown in Section 5)

The use of super-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, which is precisely what this work addressed in a previous stage of the research,[7] *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*. The cut-downs done to the algorithm (a PC-based SW simulator version of the work reported in this paper) did undoubtedly affect the search performance, but a trade-off was found which validated the proposal, reporting 1.57 dB improvement over the D9/7.

2. SYSTEM ARCHITECTURE

The intended system should be able to perform self-calibration prior to its operating phase. In addition, should a change in the input data specification happen to occur, a new self-calibration phase has to be triggered. A mixed HW/SW architecture has been selected because it offers a reasonable trade-off between performance and flexibility. Figure 2 shows the general architecture of the system, outlining the HW/SW partitioning accomplished, where the EA is running on the embedded PowerPC processor and the wavelet core is attached to it as a peripheral along with some other HW modules which will be described in detail in Section 3.

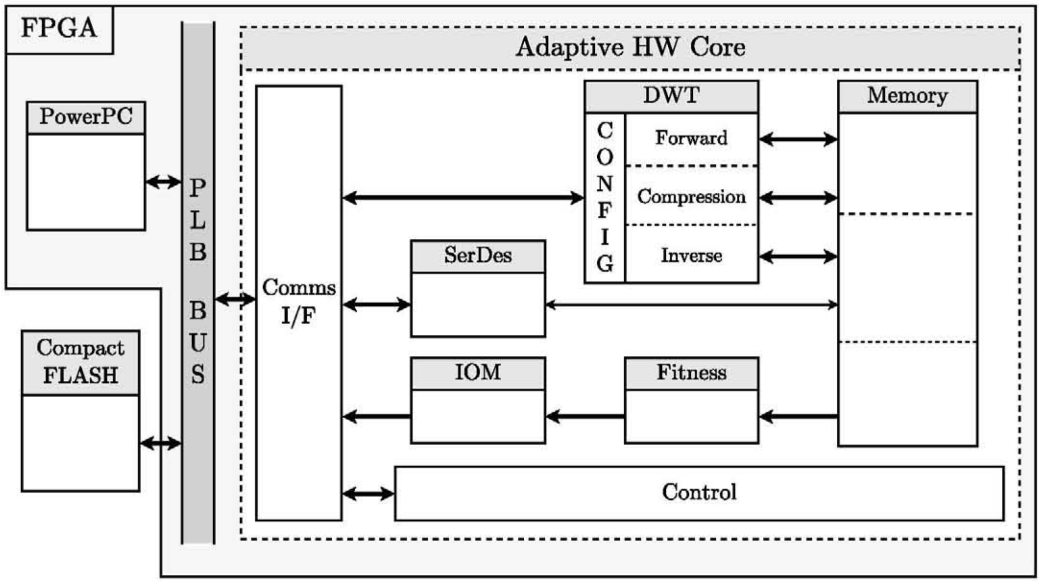


Figure 2. System level architecture.

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). Table 1 shows the partitioning resulting from applying this design philosophy and the results obtained after doing a software simulation for 500 generations, which was addressed in a previous stage of this work.[7] Each of the EA operators (as will be introduced in Section 4) are shown 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).

As it can be extracted from Table 1, the most time consuming and core part of the proposed system, the DWT, is performed in HW to speed-up the adaptation as well as the normal system operation (the result of the previous work yielded a time of 20.479 ms needed to compute a single wavelet transform, whether it is a forward or an inverse one). Besides, the sorting of the population according to the measured fitness is also implemented in HW not only for timing reasons, but also for simplicity, using an Insert Order Machine (IOM) which acts in parallel to the evaluation as results are produced.

Regarding the evaluation, as it consists in *measuring the wavelet performance in compression*, after doing a forward DWT, a compression stage is needed. Since the remaining compression stages in a typical *real* compression algorithm are highly time consuming tasks, 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[14] 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. This approach was also used in the previous stages of our own work,[15] together with some other proposals at the algorithmic level which will be analysed in Section 4.

Table 1. 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 (Fitness)	✓		31.62	2.150
selection	*sorting population (IOM)*	✓		0.040	0.003
	parent population		✓		

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

Figure 3 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.*

The system operates by loading a training image from the Compact Flash memory into the peripheral private RAM memory (implemented with on-chip FPGA RAM resources). Afterwards, the EA is initialized with a set of random candidate solutions (the initial parent population). This population goes through the variation operators (recombination and mutation) to create the offspring population to be evaluated. This phase, *evaluation*, comprises a forward DWT, the ideal compression phase mentioned above, reconstruction of the image (inverse DWT) and computation of the fitness function. Once all the offspring population has been evaluated and sorted according to the ranking fitness, the new parent population is created after applying the (survivor) selection operator.

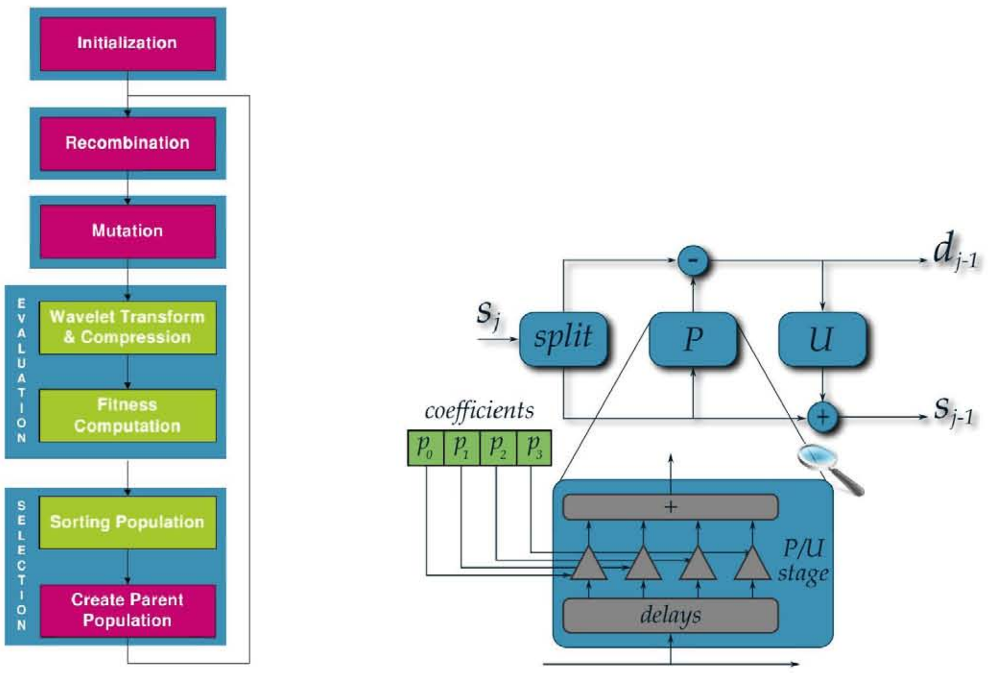
This *prototype* system implementation is just a proof of concept, so no considerable effort has been made in optimizing the system performance. For instance, as will be analysed in the following Section describing the Hardware implementation, the lifting scheme is a direct mapping from its algorithmic description, so no optimizations at the level of data dependencies has been considered; neither the concurrent access to the internal RAM memories so fitness computation could be made in parallel to the inverse DWT as these results are produced (as would have been the optimal solution). The only segmentation accomplished has been the *population sorting* phase, which instead of waiting until all the individuals had been evaluated, sorts them as fitness results are produced. This way, just some clock cycles after finishing the evaluation of the last individual, will the population sorting also be finished.

2.1 HW/SW communication

The prototype system, based on the standard Xilinx embedded development flow, is made up of a PLB Bus based system with a PowerPC processor. Attached to this bus is the adaptive peripheral, configurable through a set of registers. Just two registers have been defined; one for the PowerPC to send data to the peripheral and one for the peripheral to communicate its status to the processor. From the PowerPC side the meaning of these registers is:

- *Data/Command register* (DC Reg, Write)
  - *Data.* Configuration data (image pixels and filter coefficients) sent to the peripheral.
  - *Command.* Sets the peripheral into one of the 6 operating modes defined (described in Section 3).
- *Status register* (SR, Read). The processor polls the peripheral status waiting for a command to be finished.

The architecture of the system is flexible enough so not only easy upgrades of the hard cores are possible, but also adding new image processing cores with a set of configuration parameters. These can be adapted and



(a) (b)

Figure 3. (a): Flow graph of the algorithm. (b) Idea of the algorithm.

reconfigured from the PowerPC processor according to the proposed EA, since the evaluation phase is carried out by the very same hardware core. The only module which could probably need to be changed is that responsible of the fitness computation, but for most image (filtering) processing tasks, it is general enough to fulfill adaptation requirements.

3. ADAPTIVE WAVELET HARDWARE CORE

According to Figure 2 the description of each of the modules of 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, which may be configured to perform a Predict or Update stage. Besides, a configuration bit sets the operating mode of the transform, forward or inverse, and another one sets whether the ideal compression proposed is done or not.

  As shown in previous works[16,17] 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 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). 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, defined as the *Mean Absolute Error* (MAE) (2).

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

where R, C are the rows and columns of the image and I, K the original and transformed images respectively. Since dividing by 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***. Module responsible of interfacing with the bus-based system and decoding its commands.
- ***Control***. 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***. Internal peripheral memory implemented with the BRAM blocks available in the FPGA. It has been designed to host:
  - The original image (pixels are 8 bit wide integers).
  - The result of the DWT plus the compression if configured (28 bits wide fractional format as explained previously).
  - The reconstructed image (8 bit wide integers).
- ***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 have been clearly defined in the peripheral by the different operating modes it supports, coded as a State Machine in the control module. The communications IF module is in charge of decoding the configuration commands sent by the microprocessor to the peripheral, which will set it into a different operating mode according to the FSM commands driven by the Control module. These modes are described in Table 2.

Table 2. Control State Machine modes

Mode	**Description**
IMG_RCV	Image reception from the processor (to be stored in the internal memory of the peripheral)
CFG_WV	Candidate wavelet configuration (filters coefficients sent from the processor)
IMG_OP	Computes DWT over one of the images stored in the memory
INDIV_FIT	Compute fitness figure for the last configured individual
POP_FIT	Sent ordered result of the evaluation of the whole population
IMG_BACK	Sent image back from internal memory to processor (to be stored in Compact Flash)

In the *IMG_RCV* mode the processor reads an image from the Compact Flash memory and sends it to the peripheral to be stored in the memory module. *CFG_WV* configures the peripheral to a given wavelet by sending it the lifting coefficients (in the correct order to set up a forward or inverse transform) and 6 extra bits to configure each of the lifting filters to behave as a Predict or Update stage. *IMG_OP* computes a 1-level DWT of the previously configured DWT, so to perform an n-level DWT, n IMG_OP commands have to be issued, being the control module responsible of taking care of the current transform level. When the whole wavelet transform has been computed, *INDIV_FIT* computes the fitness function of the current candidate solution, being automatically sent afterwards to the IOM module which will sort this individual according to the obtained fitness. Once all the offspring population has been evaluated, and, therefore, ordered, the microprocessor can issue a *POP_FIT* command to retrieve the results of the evaluation (the number of computed fitness values sent back to the processor is configurable so that no unneeded transfers are performed). Finally *IMG_BACK* is in charge of reading an image from the internal memory of the peripheral and send it to the microprocessor which will store it in the Compact Flash.

4. PROPOSED EVOLUTIONARY ALGORITHM DRIVING ADAPTATION

As explained in Section 2 the embedded PowerPC processor is responsible of running the EA, in particular, an Evolution Strategy (ES). Previous works done by other authors made use of supercomputing resources to evolve state of the art performing wavelets. Since this implementation is targeting embedded systems, several and severe simplifications were proposed as compared to these works. Besides, another simplification to the standard, simplest ES, has been accomplished. This proposal and the thorough set of tests performed to tune the ES and find a suitable set of parameters for evolution to succeed, in the case of adapting the DWT to fingerprint images, can be found in our previous work.[7] The summary of these proposals and the resulting ES are reproduced here for the shake of clarity, pointing to the particular conference papers where they were first presented.

The first simplifications[18] to the algorithm 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.[9]
2. Use of *uncorrelated mutations with one step size*[19] (introduced in Section 4.1) instead of the overcomplex CMA-ES method used by Babb et. al.[11,12]
3. Evolution of *one single set of coefficients for all MRA levels.*
4. *Ideal compression for the evaluation of the transform.* Since doing a complete compression would turn out to be an unsustainable amount of computing time, the simplified evaluation method of Grasemann et al.[9] was further improved with the proposed ideal compression.

Finally, the last two simplifications were accomplished and validated in a second stage of our work.[15]

1. *Uniform random distribution.* Instead of using a Gaussian distribution for the mutation of the object parameters (see Section 4.1 for a description of the parameters), 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,[20] using MAE gives almost identical results because the interest lies in relative comparisons among population members.

4.1 Implementation of the ES

This Section describes the ES implemented in the PowerPC processor. In the analysis of previous works done in Section 1.3, ESs are the chosen type of EA in the State of the Art results, so this decision seems reasonable. However, since works using a Genetic Algorithm also performed well, we decided to make use of the wavelet encoding proposed in those works, the lifting scheme, which is also more suitable for an embedded system implementation.

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* ($\mu < \lambda$ stands), or both the parents and offspring, referred to as *plus-selection.* The former, *comma-selection*, is generally preferred in ES over the *plus-selection*, where the selection pool consists of the $\mu{+}\lambda$ individuals of the parent and offspring populations, for being, in principle, able to leave (small) local optima and not letting survive misfit parameters of the individuals. Therefore, no *elitism* is allowed.

Standard ***representation*** of the individuals in ESs, $\langle x_1, \ldots, x_n, \sigma\rangle$, is composed of a set of *object parameters* x_i to be optimized, and a single *strategy parameter* σ, which determines the degree of perturbation of the mutation operator. Therefore, the encoding of each wavelet individual is of the form:

$$\langle P_1, U_1, P_2, U_2, P_3, U_3, k1, k2\rangle \tag{3}$$

where each P_i, U_i is made up of 4 coefficients and k_i are single coefficients, yielding the 26 fixed point coefficients introduced in previous Section. As a comparison, standard D9/7 wavelet uses $\langle P_1, U_1, P_2, U_2, k1, k2\rangle$.

One of the particular features of ESs is that the individual step sizes of the variation operator for each coordinate is governed by self-adaptation. This self-adaptation of the step size σ, also known as *mutation strength* (i.e. standard deviation of a normal distribution), implies that σ is also included in the chromosomes, undergoing variation and selection itself (co-evolving along with the solutions).

For real-valued search spaces, ***mutation*** in ESs 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 $\langle 0 \ldots RAND\_MAX\rangle$ ($RAND\_MAX = 2147483647$) 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,[21] described below:

$$\begin{aligned} z_1 &= \pm\sqrt{-2 \times \log(R)} \times \frac{U}{R} \\ z_2 &= \pm\sqrt{-2 \times \log(R)} \times \frac{V}{R} \end{aligned} \tag{4}$$

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.

For the ***survivor selection*** after creating λ offspring and calculating their fitnesses, the best μ individuals are selected deterministically for the new parent population based on the ranking of the individuals' fitness. The selection scheme used is the *comma-selection*.

Regarding the ***recombination*** process, *intermediate recombination* has been selected, where the object and strategy parameters of the selected parents are averaged. The selection of these ρ recombinants is done randomly from the parent population.

Table 3 summarizes the proposed ES in charge of looking for the optimum coefficients throughout the search space.

5. RESULTS

5.1 Experimental Set-Up

The prototype system has been implemented in a Xilinx ML507 board featuring a Virtex-5 FPGA (XC5VFX70T) with an embedded PowerPC® 440 processor. Since the algorithm has already been validated in our previous works making use of a software version of the system, the aim of this Section is to validate the FPGA implementation.

5.2 Adaptation Results

5.2.1 Evolution results for fingerprint images

The specific type of images chosen to demonstrate the adaptation capabilities of the system are fingerprints, in particular, the first set of 80 images of the FVC2000[22] fingerprint verification competition. Images were greyscale, 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. Every result shown in this work is compared with the performance obtained with the standard D9/7 wavelet. Since *evolution* time depends on the time spent evaluating candidate solutions, which involves the whole chain of *forward DWT + compression + inverse DWT*, the smaller the images, the lower the time to do a transform and hence achieve adaptation. For this reason, images were resized to 256×256.

Table 4 gathers the performance (MAE) results of the adaptation of the system for the training images and for a couple of test images of the same type not seeing during evolution to demonstrate the generality of the result. The best evolved individual (*Evo_FP*) is compared with the standard D9/7, clearly showing a performance improvement.

Table 3. Proposed Evolution Strategy

Parameter / Operator	**Value**
Representation	$\langle x_1, \ldots, x_n, \sigma \rangle$ $n = 26$, fixed point coefficients
Wavelet Encoding	$\langle P_1, U_1, P_2, U_2, P_3, U_3, k_1, k_2 \rangle$
Mutation[ab]	$\sigma' = \sigma \cdot \exp^{\tau \cdot N(0,1)}$ $x_i' = x_i + \sigma' \cdot U_i(-\sigma', \sigma')$
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] $U_i(-\sigma', \sigma')$: separate draw from the discrete uniform distribution for each variable i

Table 4. Results for fingerprint images

Wavelet	**Training image *T1***	**Test image *T1* #1**	**Test image *T1* #2**
D9/7	6.64	6.80	7.28
Evo_FP	4.97	5.05	5.96

5.2.2 Test evolved wavelet on a different type of images (T2)

Another test has been performed simulating the case of a change in the input type of images. In this case, Magnetic Resonance Images (MRI) have been used. Table 5 gathers the results of applying the standard D9/7 wavelet (first row) over this new type of images and its comparison with the performance obtained with the current wavelet transform configured in the system, *Evo_FP*, which is adapted to fingerprints (second row). As expected, since this wavelet was adapted to deal with a different type of images the performance could improve if a new self-calibration phase is triggered. The results after the system adapts itself for this new type of input data are shown in the last row of Table 5.

Table 5. Results for MRI images

Wavelet	**Training image *T2***	**Test image *T2* #3**	**Test image *T2* #4**
D9/7	9.14	8.08	7.58
Evo_FP	8.15	7.46	6.76
Evolved	7.96	6.95	6.57

5.3 Implementation Results

The implementation results for this prototype featuring an over-dimensioned datapath (Q10.16) introduced in Section 3 can be found in Table 6. Since this is just a *proof-of-concept* implementation, no effort has been put into improving overall system performance and architectural resource consumption (any of the multiple lifting scheme FPGA or VLSI implementations which can be found on the literature can benefit from this approach to coefficients adaptation).

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

Module	**Resources**				**Frequency (MHz)**
	Slice LUTs	Slice Registers	DSP48Es	BRAM (Kb)	
DWT	2818 / 44800	4417 / 44800	76/128	–	112.67
Compression	43 / 44800	39 / 44800	–	–	391.34
Fitness function	95 / 44800	66 / 44800	–	–	341.88
IOM	3920 / 44800	2209 / 44800	–	–	231.93
Image memory	34 / 44800	27 / 44800	–	2304 / 5328	–

5.3.1 Adaptation time

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, $s_jj - 1$, is:

$$((256 \times 256) + 56) \times 2 = 131184$$

Equivalently, for the second transform level, s_{j-2}, where the smaller subset (compared to the original input s_j) s_{j-1} is used as input, the number of clock cycles needed for its computation is 32880. This means that a whole 2-level forward DWT takes 164064 clock cycles. The same analysis can be applied for the 2-level inverse 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 $256 \times 256 = 65536$ 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 $164064 + 65536 + 164064 + 65536 = 459200$ clock cycles. If system frequency is set to 100 MHz, 4.592 ms are needed for the evaluation of a single candidate wavelet. Since system is let to evolve during 1000 generations, doing 70 evaluations per generation, around 5.34 minutes are needed for evolution to finish.

Previous timing analysis applies to the adaptive wavelet core timing. However, ES running in the PowerPC processor will add more time to the evolution. The final time measured increases to 12 minutes, which means that the system is able to adapt to a new type of input data in this time. This may look like a huge amount of time, but as shown in our previous work,[7] between generations 100 and 300 the system has already evolved a better solution. This reduces the effective evolution time to between just 1.2 and 3.6 minutes.

6. CONCLUSION

A self-adaptive FPGA-based architecture for image compression in embedded systems has been proposed and validated. The system improves existing standard wavelets like D9/7 used in JPEG2000. Besides, it is able to self-adapt to changes in the type of input data (images) being dealt with. As Section 5.2 shows, the system is able to optimize standard compression performance. While the quality of results depends on a particular application domain, better than the standard wavelets are still obtained in both cases tested.

Future work will center in improving system performance. The timing analysis and the architecture description clearly show that a great amount of time can be saved if a better memory architecture and system pipelining is used, since compression could be done without previously saving the forward DWT in memory. The same applies for the inverse DWT and the fitness computation. This results in $164064 + 164064 = 328128$ clock cycles for the evaluation of an individual, which means saving around 29% of the HW computation time.

Besides, a Uniform mutation may also be used for the mutation of the strategy parameter, σ. This simplification could lead to further reduce computation time. However, optimization performance of the evolutionary search should be checked as was done previously with the object parameters.

Using images sized 128×128 pixels will also help in reducing even more the evaluation time, but, again, results have to be carefully checked to make sure the solution is general enough.

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 has been supported by MSMT under research program MSM0021630528 and by the grant of the Czech Science Foundation GP103/10/1517.

REFERENCES

1. D. Taubman and M. Marcellin, *JPEG2000: Image Compression Fundamentals, Standards and Practice*, Springer, 1 ed., Nov. 2001.
2. W. Sweldens, "The lifting scheme: A custom-design construction of biorthogonal wavelets," *Appl. Comput. Harmon. Anal.* **3**(2), pp. 186 – 200, 1996.
3. W. Sweldens, "The lifting scheme: a construction of second generation wavelets," *SIAM J. Math. Anal.* **29**(2), pp. 511–546, 1998.
4. 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., **2569**(1), pp. 68–79, SPIE, 1995.
5. G. Uytterhoeven, *Wavelets: software and applications.* PhD thesis, Department of Computer Science, K.U.Leuven, Leuven, Belgium, Apr. 1999. Roose, Dirk and Bultheel, Adhemar (supervisors).
6. A. Eiben and J. Smith, *Introduction to Evolutionary Computing*, Springer, Oct. 2008.
7. R. Salvador, F. Moreno, T. Riesgo, and L. Sekanina, "Evolutionary approach to improve wavelet transforms for image compression in embedded systems," *EURASIP Journal on Advances in Signal Processing* **2011**, pp. 1–20, 2011.
8. U. Grasemann and R. Miikkulainen, "Evolving wavelets using a coevolutionary genetic algorithm and lifting," in *Genetic and Evolutionary Computation GECCO 2004*, *Lecture Notes in Computer Science* **3103**, pp. 969–980, Springer Berlin / Heidelberg, 2004.
9. U. Grasemann and R. Miikkulainen, "Effective image compression using evolved wavelets," in *Proceedings of the 2005 Conference on Genetic and evolutionary computation, GECCO '05*, pp. 1961–1968, ACM, (New York, NY, USA), 2005.
10. B. Babb and F. Moore, "The best fingerprint compression standard yet," in *IEEE International Conference on Systems, Man and Cybernetics, 2007. ISIC.*, pp. 2911–2916, 2007.
11. 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*, **7347**, pp. 73470O–10, SPIE, (Orlando, FL, USA), May 2009.
12. 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, Qubec, Canada), 2009.
13. F. W. Moore and B. Babb, "A differential evolution algorithm for optimizing signal compression and reconstruction transforms," in *Proceedings of the 2008 GECCO conference companion on Genetic and evolutionary computation*, pp. 1907–1912, ACM, (Atlanta, GA, USA), 2008.
14. J. Villasenor, B. Belzer, and J. Liao, "Wavelet filter evaluation for image compression," *IEEE Transactions on Image Processing* **4**, pp. 1053 –1060, aug 1995.
15. R. Salvador, F. Moreno, T. Riesgo, and L. Sekanina, "High level validation of an optimization algorithm for the implementation of adaptive wavelet transforms in FPGAs," in *Digital System Design: Architectures, Methods and Tools (DSD), 2010 13th Euromicro Conference on*, pp. 96–103, 2010.
16. 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, 2000.*, **3**, pp. 1174–1177 vol.3, 2000.
17. 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* **11**(6), pp. 596–604, 2002.

18. 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 2010 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, pp. 171–178, IEEE Computer Society, jun. 2010.
19. H. Beyer and H. Schwefel, "Evolution Strategies. A comprehensive introduction," *Natural Computing* **1**, pp. 3–52, Mar. 2002.
20. Z. Vasicek and L. Sekanina, "An evolvable hardware system in Xilinx Virtex II Pro FPGA," *Int. J. Innov. Comput. Appl.* **1**(1), pp. 63–73, 2007.
21. G. Marsaglia, "Normal (gaussian) random variables for supercomputers," *The Journal of Supercomputing* **5**, pp. 49–55, 1991. 10.1007/BF00155857.
22. D. Maio, D. Maltoni, R. Cappelli, J. Wayman, and A. Jain, "FVC2000: fingerprint verification competition," *IEEE Transactions on Pattern Analysis and Machine Intelligence* **24**, pp. 402–412, Mar 2002.