Self-Reconfigurable Evolvable Hardware System for Adaptive Image Processing

Rubén Salvador, Andrés Otero, Javier Mora, Eduardo de la Torre, Teresa Riesgo, and Lukas Sekanina, *Senior Member*, *IEEE*

Abstract—This paper presents an evolvable hardware system, fully contained in an FPGA, which is capable of autonomously generating digital processing circuits, implemented on an array of processing elements (PEs). Candidate circuits are generated by an embedded evolutionary algorithm and implemented by means of dynamic partial reconfiguration, enabling evaluation in the final hardware. The PE array follows a systolic approach, and PEs do not contain extra logic such as path multiplexers or unused logic, so array performance is high. Hardware evaluation in the target device and the fast reconfiguration engine used yield smaller reconfiguration than evaluation times. This means that the complete evaluation cycle is faster than software-based approaches and previous evolvable digital systems. The selected application is digital image filtering and edge detection. The evolved filters yield better quality than classic linear and nonlinear filters using mean absolute error as standard comparison metric. Results do not only show better circuit adaptation to different noise types and intensities, but also a nondegrading filtering behavior. This means they may be run iteratively to enhance filtering quality. These properties are even kept for high noise levels (40 percent). The system as a whole is a step toward fully autonomous, adaptive systems.

Index Terms—Evolvable hardware, FPGAs, self-adaptive systems, reconfigurable hardware, adaptable architectures, autonomous systems, evolutionary computing and genetic algorithms

1 Introduction

Embedded systems engineering is nowadays facing an enormous challenge derived from the ever increasing demand for highly versatile electronic devices, high performance, increasing complexity, flexibility, low power, and reprogrammability. Low time to market and high operational lifes are also required. These stringent requirements, which oppose each other, have caused an exponential increase in the underlying complexity of embedded systems, since, among other concerns, the need for systems able to adapt to very diverse operating conditions throughout its lifetime arise. Besides, this adaptation should be possible without human intervention.

A system design trend that seems to be consolidating as a de facto standard in the last years is the use and integration of different IP cores created and validated independently, as a way to manage complexity. This helps in reducing design time, and consequently, time to market, while offering lower power and higher performance computing that are distinctive of hardware implementations. Besides, to obtain the previously mentioned adaptation capabilities, the challenge is to combine hardware performance with the flexibility required by adaptive systems, while keeping design times as low as possible. Therefore, an adequate combination of techniques and technologies is needed to obtain this expected self-adaptive behavior.

Making use of the dynamic and partial reconfiguration (DPR) technology of modern SRAM-based FPGAs opens new possibilities toward building efficient adaptive systems, because of the lower costs associated with changing just a portion of the system without stopping it entirely, as compared with full reconfiguration. In addition, it enables self-reconfiguration, allowing this way to control the process from inside the device, which is a clear advantage to maximize integration.

On the other side, the use of evolutionary computation as an automatic problem solver in an increasing number of applications is a known fact. The variety of publications focusing on this topic demonstrates how the solution to a myriad of different problems can be automated by the use of a proper evolutionary algorithm (EA).

The interactions between EAs and electronics gave rise to the field known as evolvable hardware (EHW), which proposes the use of an EA for the automatic design and adaptation of circuits. If evolution is performed offline, as part of the system design flow, an optimum circuit can eventually be found that will afterward be implemented in the final system as in traditional design approaches. In this case, the design space is explored in a different, more automatic, way not constrained by designers' knowledge. However, if evolution runs in the final system, a possibility for *autonomous* adaptation arises. Besides, if the system implementation substrate is itself adaptive (reconfigurable),

- *R. Salvador is with the Electronic and Microelectronic Design Research Group (GDEM), Research Center on Software Technologies and Multimedia Systems for Sustainability (CITSEM), Departamento de Sistemas Electrónicos y de Control, EUIT Telecomunicacion, Universidad Politécnica de Madrid, Carretera de Valencia Km. 7, 28031 Madrid, Spain. E-mail: ruben.salvador@upm.es.*
- *A. Otero, J. Mora, E. de la Torre, and T. Riesgo are with the Centro de Electronica Industrial, ETSI Industriales, Universidad Politécnica de Madrid, José Gutierrez Abascal, 2, 28006 Madrid, Spain. E-mail: {joseandres.otero, javier.morad, eduardo.delatorre, teresa.riesgo}@upm.es.*
- *L. Sekanina is with the IT4Innovations Centre of Excellence, Faculty of Information Technology, Brno University of Technology, Bozetechova 2, 612 66 Brno, Czech Republic. E-mail: sekanina@fit.vutbr.cz.*

Manuscript received 15 Aug. 2012; revised 15 Jan. 2013; accepted 24 Feb. 2013; published online 4 Apr. 2013.
Recommended for acceptance by K. Benkrid, D. Keymeulen, U.D. Patel, and D. Merodio-Codinachs.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TCSI-2012-08-0559.
Digital Object Identifier no. 10.1109/TC.2013.78.

circuit structure might eventually be able to change autonomously. This is the so-called online EHW, which is further classified in EHW taxonomy as A*daptive hardware.*

Evolvable hardware was proposed by Higuchi [1] in 1993 as the automatic design of hardware using an EA. In [2], EHW was mentioned as "this new *species* (of hardware)" whose objective is "*the development of a new generation of hardware, self-configurable and evolvable, environment-aware, which can adaptively reconfigure to achieve optimal signal processing, survive and recover from faults and degradation, improving its performance over lifetime of operation.*" The goal is to create systems able to adapt to their environment without human intervention. Ideally, these systems are able to deal with problem specification changes and respond to unexpected input signals variations, changes in conditions like energy availability, bandwidth adaptation, and many others. This line of research brings together reconfigurable hardware, autonomous systems, artificial intelligence, and automatic design. First steps were given in the mid 1990s with Xilinx multiplexer-based XC6200 FPGA family. Once this family was discontinued, manufacturers' reconfiguration technologies proved not yet mature enough to provide a seamless tool to be embedded in the final SoC. This was mainly due to the fact that unconstrained evolution in commercial FPGAs by direct bitstream manipulation was considered not possible because random modifications of the bitstream were dangerous for the device integrity. Besides, its huge size turned the search space unmanageable, and also, reconfiguration times achievable with the configuration ports are still high enough to prevent this technology to be embraced as a standard. An alternative trying to overcome these limitations was proposed in the early 2000s. It is usually referred to as virtual reconfigurable circuits (VRC) [3].

A VRC within an FPGA is a (virtual) reconfiguration layer built on top of the device fabric that reduces the complexity of the reconfiguration process while increasing speed, creating a kind of "application specific programmable elements." It consists of an array of processing nodes where each of them contains all the intended required functions (selectable using multiplexers) along with certain internode connectivity configuration. This information is expressed in the chromosome of each candidate solution, which is used as the set of the configuration bits. Although reconfiguration is very fast, because it only involves writing a big register, this approach suffers from a huge area overhead (all functions are implemented in every node of the graph) while multiplexers increase circuit delay.

The work introduced in this paper explores dynamic and partial reconfiguration capabilities of modern FPGAs as a key technology to build adaptive systems driven by an EA. The first milestone in our research proposed an alternative reconfigurable core architecture based on DPR and an enhanced HW-ICAP (ICAP stands for Xilinx Internal Configuration Access Port) module [4]. The application domain was adaptive image filtering, which in real-world scenarios usually involve the need to deal with large data sets and a high degree of uncertainty. This situation makes the optimization of the performance of an image processing system a complex trial-and-error process which is backed on the designer's skills and experience. As suggested in [5], an EA is expected to efficiently deal with this uncertainty coming from the diverse and unknown types of noise that might appear at the input signal. In [6], a simulation of the image processing matrix and the EA, which is designed to minimize the number of reconfigurations between evaluations, was accomplished. Afterward, once the whole system was implemented, fault-tolerance tests were accomplished [7] to check self-healing capabilities against both transient and permanent faults. Even a quite good behavior against cumulative faults was noticed, showing its robustness against permanent continuous circuit degradation. The evolvable platform features a widely known 2D systolic processing architecture highly used in VLSI signal processing, and an optimized DPR control engine that allows the implementation of adaptive (evolvable) processing hardware with native reconfiguration support. The architecture consists of a highly regular and parallel 2D mesh-type array of processing elements (PE) arranged like a systolic structure that provides intensive data processing capabilities in a broad range of fields.

This paper not only proposes the mentioned evolvable platform, but also an enhanced evaluation method that let the system evolve not only its structure, but also the associated timing of the processing circuit, which was fixed by design in the initial implementations. An online, automatic, latency analysis is performed, which will eventually allow the system to decide whether it needs to evolve toward a bigger or smaller circuit to achieve the required performance. Besides, more restrictive input signal conditions are analyzed, such as increasing noise intensities for different types of noise and edge detection. It is demonstrated how the system is able to adapt to different environmental conditions which affect its inputs. This kind of self-calibration feature is very important for systems that are deployed on hazardous, unreachable, and even unknown environments at design time, providing systems with adaptive signal processing and self-healing capabilities.

VRC has been for years the preferred method for providing reconfiguration capabilities to evolvable systems. Since it is such a different approach to DPR, various pros and cons can be observed in each method. Therefore, an initial comparison between both implementation methods, i.e., DPR and VRC, for the proposed reconfigurable processing architecture was accomplished in [8]. A VRC-based implementation of the processing array using VHDL was accomplished so that a fair comparison in terms of performance and implementation area between both could be obtained for modern FPGAs. Since the implementation results using DPR have been improved since that time, an update of the comparison is also included in this work.

The paper is structured as follows: Section 2 features an analysis of the related work in this field of research. Section 3 introduces the proposed architecture together with the results of the experiments done to check its adaptation capabilities. An important system enhancement that allows for a dynamic timing (evolved circuit latency) self-adjustment is introduced in Section 4. In Section 5, a discussion on the system generalization and its behavior for different types and intensities of noise as well as for other filtering tasks is conducted. This section also presents

an interesting filter feature discovered by evolution, nondestructive-cascaded filtering. The implementation results are included in Section 6, while Section 7 analyzes the impact of both, virtual and native reconfiguration, in terms of resources and timing performance. This paper is concluded in Section 8.

2 Related Work

Adaptive image processing (as a particular application of adaptive systems) is a highly active field of research. However, we do not consider in here more advanced filtering techniques (such as switching filters or adaptive median filters [9], [10]), because they are much more complex than the filters that can be embedded into the proposed processing array. Besides, our approach deals with the adaptation of the system as a whole, designing automatically, and online, new candidate filters, as opposed to most of the techniques proposed within classical linear and nonlinear filter theory that require an *on-the-fly* adaptation to the local properties of the signal as this is being filtered (e.g., checking for singularities to adapt to).

First attempts with FPGAs (other than parametric optimization based on EAs for FPGAs forerunner devices such as GALs [11]) were accomplished with discontinued Xilinx XC6216, which allowed for safe random modifications of the configuration bitstream, providing a possibility for unconstrained evolution. Thompson [12] designed an EA running in a PC that was connected to this device to reconfigure and evaluate each of the candidate circuits to evolve a tone discriminator. However, since following FPGA families did not allow for these safe modifications and reconfiguration times were very slow, this unconstrained evolution performed directly over the silicon substrate (although the EA run on a PC) was abandoned.

As fabrication technology improved, bigger and more complex FPGAs began to appear. This increasing resource capacity allowed also embedding the EA in the device (as specific HW or running in an embedded processor), so that autonomous systems using internal reconfiguration might be implemented. However, reconfiguration techniques were not yet mature enough to be embraced as a standard. Slow reconfiguration times, along with an obfuscated, unknown bitstream, made DPR not feasible for EHW. This issue gave birth to the custom reconfiguration scheme introduced above and known as VRC [3], which achieves high reconfiguration speed and a suitable computation granularity by defining the minimum reconfiguration unit and an ad hoc circuit network described using standard HDLs as any other module in the FPGA. This approach involves implementing all the possible configurations of the architecture simultaneously on the device, selecting the desired one using control (configuration) registers and multiplexers, performing (virtual) reconfiguration (writing those configuration registers) typically in a few clock cycles. The main problem of VRCs is the area overhead caused by the inclusion of the simultaneous implementations of every function plus the required routing multiplexers. Maximum achievable frequency is also reduced. Therefore, the use of runtime reconfiguration has remained in the background until technology was improved. In the last years, DPR has received an increased interest from the community, and some interesting approaches are arising. Therefore, the analysis of the state of the art will focus on evolvable digital systems using internal reconfiguration, where the full system is implemented in the FPGA, including EA and fitness calculation (so no external tools are needed such as a PC running the EA). Table 1 gathers the main published works in the subject from the last years, classifying them according to several categories. This table clearly shows how most of the design efforts have been tackled using VRCs. Therefore, those approaches where no VRCs are used (so a contained use of resources is obtained) are emphasized in the text, as well as those where circuit structure is modified (designed by evolution), not where just a set of register values is optimized.

TABLE 1
FPGA Implementations of Evolvable Digital Systems

Ref.	Application	Platform	EA	Fitness
[13]	Hash Functions	VRC @ XC4VFX20	HW	HW
[14]	Image Filters	VRC @ XC2VP50	PowerPC	HW
[15]	CGP Accelerator	VRC @ XC2VP50	PowerPC	HW
[16]	Face Recognition	VRC @ XC2VP30	MicroBlaze	HW
[17]	Sonar Spectrum Class.	VRC @ XC2VP30	PowerPC	HW
[18]	Arithmetic Circuits	VRC @ XCV2000E	HW	2×HW
[19]	Image Filters, Classif.	VRC @ XCV2000E	HW	HW
[20]	Const. Coeff. Mult.	VRC @ XC2VP50	HW	HW
[21]	CGP Accelerator	VRC @ XC5VFX100T	PowerPC	4×HW
[22]	Small Comb. Circuits	Virtex 4 logic	PowerPC	16×HW
[23]	Cellular Automaton	Virtex II CLB	MicroBlaze	HW/Micro
[24]	Pattern recognition	VirtexII XC2VP30	PowerPC	HW
[25]	Pattern recognition	Various Virtex/Spartan	PowerPC	HW

For instance, Upegui proposes in [23] changing some configurable modules of a neural network, to adapt the processing structure. This approach, which makes use of ICAP to perform reconfiguration, is mainly constrained by its very specific purpose and the coarse granularity of the modules. In [24], Torresen et al. propose a data classification system based on a set of functional units (FUs). Flexibility is achieved by choosing between different presynthesized configurations that differ in the number of FUs, using DPR. However, only a reduced set of experiments was performed since system frequency was very slow to build up feasible EHW systems. A different approach is proposed by the same group in [25] that exploits the SRL behavior of some Virtex devices LUTs to change its logic. In this case, the reconfiguration process is carried out by directly shifting the configuration bits into the LUTs (no ICAP involved). This alternative achieves higher reconfiguration speed, reducing the overhead compared with VRCs. However, the solution is very device dependent, and reconfigurability is limited to adapt LUT functions. Furthermore, only 25 percent of the Virtex-5 LUTs has this behavior. More recently, a direct bitstream manipulation approach [22] is applied in a Cartesian genetic programming (CGP) [26] scheme, with static routing. In this case, reconfiguration is achieved by changing the values stored in the LUTs to

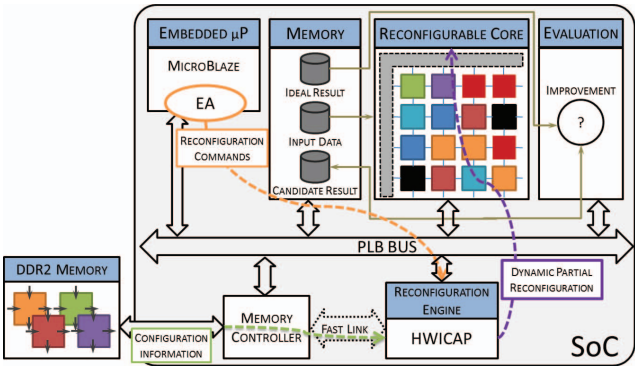

Fig. 1. Overview of the system architecture, which is implemented in an FPGA-based SoC by integrating different IP cores, mainly, a reconfigurable core that provides adaptive processing features, a reconfiguration engine in charge of commanding the required reconfiguration commands, and an EA running on an embedded microprocessor soft core.

change its functionality, as well as the communication channels.

The work proposed in this paper is based on an enhanced DPR reconfiguration engine as well as in a novel architecture that, without introducing complexity like VRCs do, increases the granularity of the PEs allowing functional evolution. Besides, a smart evaluation method is proposed to allow for circuit and associated timing dynamic adaptation. The proposal as a whole is a step forward toward autonomous, adaptive HW.

3 Autonomous, Self-Adaptive Platform Description

The platform, which is based on an FPGA SoC architecture, contains a set of IP cores, some of which are adaptive, connected through a common bus interface. An adaptive IP core is a standard IP core with the capability of adjusting its internal processing features as commanded by an adaptation engine. This adaptation engine acts upon the measured component performance trying to fulfill the requirements within a changing operating environment. In this proposal, the adaptation engine is an evolutionary algorithm.

The main components are a reconfigurable core (RC) and a reconfiguration engine (RE). RC is the processing array, which is (re)configured by the RE during the adaptation process exploiting DPR. The combination of self-reconfiguration by using the internal ICAP configuration port and an EA provides the system with the required adaptation capabilities. An embedded MicroBlaze processor issues the required reconfiguration commands to configure the RC to the candidate solutions proposed by the EA as shown in Fig. 1. A peripheral for HW fitness evaluation can also be observed, as well as a tightly coupled RAM memory, which also serves in the acceleration of the individuals' evaluation.

Regarding the RE as well as the tools and methodology followed in this work, it can be said that a lesser device-dependent solution is proposed. Whether it is true that this solution does depend on a particular device, it is also true that this proposal diminish this dependence as shown in [4]. Basically, the tool [27] used to build each PE bitstream automatically carries out, starting from the conventional netslits generated by vendor tools, all the necessary

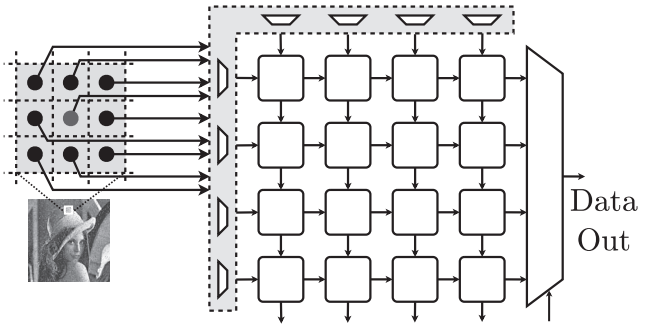

Fig. 2. Processing matrix featuring a highly regular and parallel 2D array of processing elements arranged as a systolic structure,which operates over a square window of input pixels.

transformations to be consumed by the evolvable hardware platform. In case the target device changes, new partial bitstreams can be automatically generated by just relaunching the tool.

The RC architecture is a highly regular and parallel two dimensional mesh-type array of A×B PEs organized as a systolic structure where internode connectivity is fixed and restricted to the four closest neighbors (North, South, East, West), as shown in Fig. 2. A and B are selected according to the implementation and simulation results shown in [4] and [6], yielding an array of 4×4 PEs. The input to the array is the same as in typical image convolution filters, a moving square window, sized 3×3. However, there is not a predefined routing from the window to the input PEs. By the contrary, each input PE has an associated 9-to-1 multiplexer so that circuit inputs may also be evolved. The output of the array is obtained from any of the east (right-side) PEs, by using a four inputs multiplexer controlled by evolution.

Internally, each PE can be dynamically configured to have different functionality and input mapping. Therefore, although internode connections are fixed, certain flexibility in the adaptation of the data transmission flow is allowed. Fig. 3a shows a conceptual view of a PE, which consists of a functional block (FB) and the associated routing in the input, and a register (R) in the output. Each combination of functions and connections is presynthesized and stored as an independent module in the PE library [4], [6], shown in Table 2. Unlike VRCs, fixed connections and a single function are implemented in each PE at a time, eliminating the area and timing penalties. The proposed architecture is a generic evolvable processing framework, and its suit-

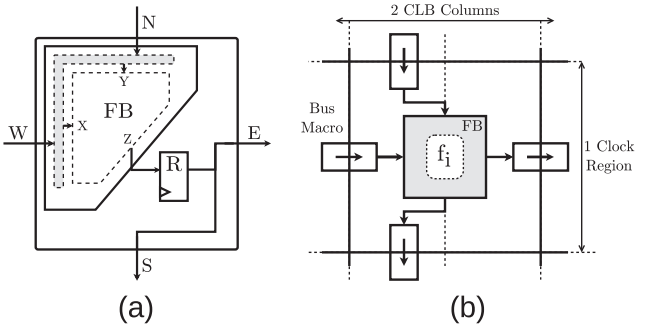

Fig. 3. View of a PE: (a) from a conceptual point of view, (b) from the reconfiguration point of view (register not shown for simplicity).

TABLE 2
Set of Components in the Library

Code	Function	Description
f_0	N + W	N + W (adder)
f_1	N << 1	N + N [a]
f_2	W << 1	W + W [a]
f_3	N $+^S$ W	N + W with saturation
f_4	N $+^S$ N	N + N with saturation [a]
f_5	W $+^S$ W	W + W with saturation [a]
f_6	(N + W) >> 1	Average
f_7	255	Constant
f_8	N >> 1	Right shift N by 1
f_9	W >> 1	Right shift W by 1
f_{10}	N	Identity
f_{11}	W	Identity
f_{12}	max(N,W)	Maximum
f_{13}	min(N,W)	Minimum
f_{14}	N $-_S$ W	Substraction with saturation to 0
f_{15}	W $-_S$ N	Substraction with saturation to 0

[a] *Improved implementation as a shifter.*

ability for different processing tasks depends on the chosen library. Adaptation is achieved by directly configuring the required PE in each position of the array, taking it from the library of presynthesized PEs. This process can be seen as placing pieces in a puzzle. For each piece (PE to reconfigure), the RE places the required element as commanded by the processor in the correct position of the matrix. Fig. 3b shows a PE from the reconfiguration point of view.

Reconfiguration time is kept low because low mutation rates in the EA produce few PEs to reconfigure, as suggested in CGP [26], which is the reference EA selected for this work. Also, unlike the standard Xilinx module, this improved RE implemented in HW allows for read-back and reallocation to be performed, reducing this way external memory accesses when moving/copying one PE from one position to another within the array. Besides, the ICAP was overclocked at up to 200 MHz and attached to an external DDR2 memory through a fast Xilinx NPI to accelerate the process. Each FB according to Table 2 is presynthesized and stored as an independent module in a library of reconfigurable processing elements defined by their partial bitstreams, which is loaded during system start-up procedure in the DDR2 memory. There is one bus macro [28] for each port of the PE (N,S,E,W), which works as the anchoring point in the pieces of the puzzle. This way, one PE can be safely replaced by another one since all of them share common connection interfaces.

3.1 Selected Evolutionary Algorithm

With respect to the evolutionary framework, the EA running on the embedded processor is inspired from VRC-based CGP architectures, such as those mentioned in Section 2. Therefore, since the chosen representation is very similar to CGP, adaptation is driven through a simple $(1+\lambda)$ evolution strategy (ES) with 1 parent and λ offspring, as suggested in [26] ($\lambda = 8$ in this work). For the selected 4×4 array, a chromosome size of 25 integer genes is needed:

$$\langle InMux_1, \ldots, InMux_{A+B}, PE_{00}, \ldots, PE_{AB}, OutMux \rangle,$$

where each $InMux_i$ gene is one of the eight input multiplexers, each PE_{ij} is one of the 16 PEs, and *OutMux* is the output multiplexer that selects as output one of the four possible output PEs. Each $InMux_i$ gene can be encoded using 4 bits to select one of the nine input pixels from the window; each PE_{ij} can also be encoded using 4 bits to select one out of the 16 possible components in the library; and *OutMux* gene can be encoded using two bits. This yields 98 bits as the chromosome size, although genes are operated at the integer level.

From a random initial population, selection chooses the fittest individual as parent for the next population, which consists of the selected parent and its mutants. Elitism is enabled and if two individuals score the same best fitness, diversity of the population is maintained by selecting the one that did not act as a parent in the previous generation. For each mutant offspring, the mutation operator modifies k randomly selected genes from the parent. Uniform integer distributions are used for this operator.

A conveniently defined fitness function needs to be implemented in the system so that evolution finds its way to the required goal, which is to minimize the difference between the filtered image and the original pattern image. In this work, mean absolute error (MAE) was selected. To obtain the fitness value of a candidate filter, a three step process is followed. First, the processing matrix is reconfigured with a candidate circuit; afterward, the image is filtered; and finally, the fitness value is calculated as

$$MAE = \frac{1}{RC} \sum_{r=0}^{R-1} \sum_{c=0}^{C-1} |I(r,c) - K(r,c)|, \quad (1)$$

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

3.2 Evaluation of the Platform

An extensive set of tests was performed to validate the system evolvability and thus its capacity to adapt to varying signal conditions to maintain close to optimal signal processing, even in the presence of unknown types of noise of different intensities. Salt and Pepper (S&P) noise intensity was varied from 5 up to 40 percent in 5 percent steps. For each intensity level, one training image corrupted with that amount of noise was used to evolve a circuit able to filter out that noise. Afterward, this evolved circuit was tested for all the other noise intensities, expecting worse results as input noise intensities grow compared to the one used to evolve that circuit. Fifty evolutionary runs for each noise intensity were repeated to perform a statistical (average, best) analysis.

Fig. 4 shows a graph compiling all these results for the best and the average of 50 runs. Performance of a standard median filter is added for comparison purposes. As it can be seen in Fig. 4a, the filtering process for images containing up to 15 percent noise intensity is better than with the median for any evolved filter, in average. If the graph is looked from the filters axis, it is clear that for increasing noise intensities used in the training image (filters F30, F35...) a "more general" filter is obtained, because it also behaves well (better than the median) for different noise

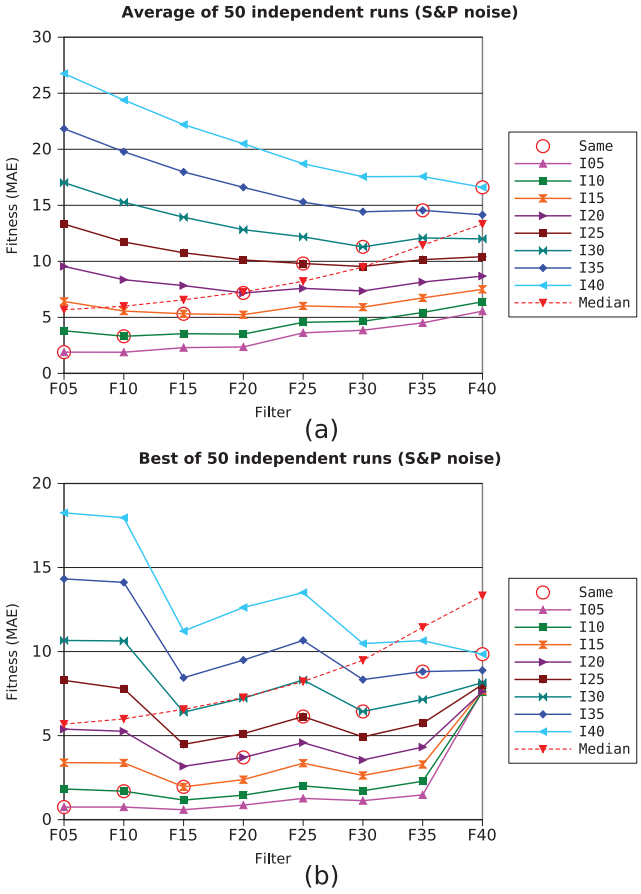

Fig. 4. Adaptation experiments results: (a) average evolved filters and (b) best of 50 runs. Each line represents the results of filtering an image corrupted by a certain noise intensity (*I05* stands for "image with 5 percent noise," etc.) with the filters shown in the x-axis. Each of these filters was evolved for a different amount of noise (*F05* stands for "filter evolved using a 5 percent image noise). The circles ("same" in the legend) represent the results of using a given filter with the image containing the same noise intensity (the image used for the evolution of each filter, *F05* over *I05*, etc.). The result for the median filters is indicated with a dotted line for comparison purposes.

intensities). Besides, results show how the evolved filters quality does not deteriorate as fast as the median does for increasing noise levels. Moreover, most of the best evolved filters as shown in Fig. 4b are way better than the median (except for those in the upper left part of the graph, which were evolved for low noise training images and, therefore, do not perform well with high noise images).

4 Improving Adaptability and Self-Tuning Timing

The proposed architecture has a fixed set of PEs and internode connectivity, which could lead to define a static circuit latency. However, when a PE is reconfigured, the internal data-path is modified depending on which inputs that particular PE uses (as Table 2 shows). This means a change in the internal routing occurs. Besides, the output multiplexer controlled by the EA makes it impossible to forecast the selected output PE (note this behavior is intended to provide a better adaptability). Therefore, there is no analytical way to determine which the correct circuit latency is.

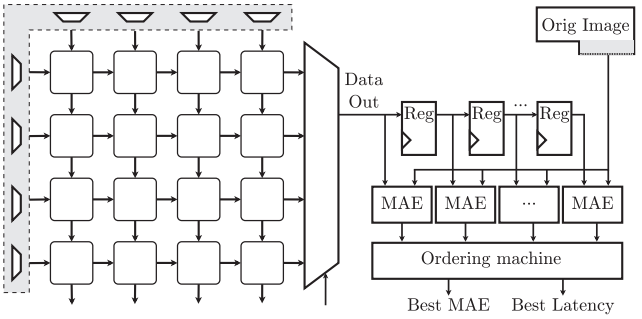

Fig. 5. Modified architecture for self-tuning timing.

For this reason, a new architecture is proposed to take into account that a dynamically changing (unknown) circuit needs a dynamically changing latency, which is adjusted online as the circuit evolves. A first approximation could be to include the latency in the chromosome so that it might evolve along with the circuit structure. However, this would increase the search space. Since changing the circuit latency means sampling the output at a different moment, it is easy and not expensive (from the resources point of view) to sample at different moments, replicating the fitness computation units. This situation is shown in Fig. 5.

The new architecture contains several MAE computation modules and a best candidate selector that keeps track of the best associated latency. The input to each fitness unit (stream of filtered pixels) is sampled in a different clock cycle by the insertion of a simple shift register. Selection of the best candidate is still done according to its fitness, but now for each offspring individual several different circuits are tested at once. If L different latency values are tested, $\lambda \times L$ versions of the same circuit are now evaluated. This involves replicating the fitness unit L times, but probably valid evolved circuits that are not fitted for the previous fixed latency may now not pass away since the behavior for different latencies is checked for a single chromosome.

4.1 Evaluation of the New Platform Architecture

The same set of tests accomplished in Section 3.2 was repeated for the new architecture. Results are collected in Fig. 6. As it can be derived from the results obtained, the latency selector strategy for circuit evaluation shows how system adaptability is improved by including a kind of self-tuning timing adaptation. Since it is difficult to appreciate the improvement in the graphs as compared to previous section, the average MAE reduction for the 50 independent runs for each of the eight noise intensities was calculated. Result yields a difference of 1.061 MAE in average (standard deviation difference of 0.58). The analysis performed in the previous section applies also in this case, since graphs are structurally equal but with a slightly lower MAE as a result of an improved evolutionary process. Fig. 7 shows a statistical analysis comparing the results obtained for the 50 independent evolutionary runs in both cases, with and without the latency selector strategy. As it can be observed in the boxplots, there is lesser dispersion in the results in the case the latency selector is used, as well as a lower MAE. Therefore, it can be concluded that the dynamic latency evaluation helps in improving the search process.

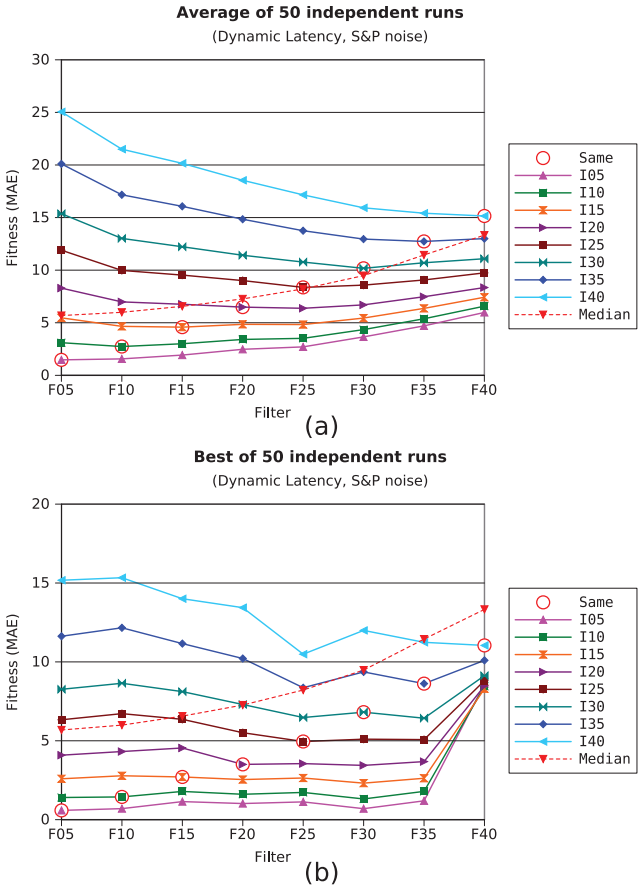

Fig. 6. Results of the adaptation experiments using latency evaluation: (a) average evolved filters and (b) best of 50 runs. For an explanation of axis and legends labels, see Fig. 4.

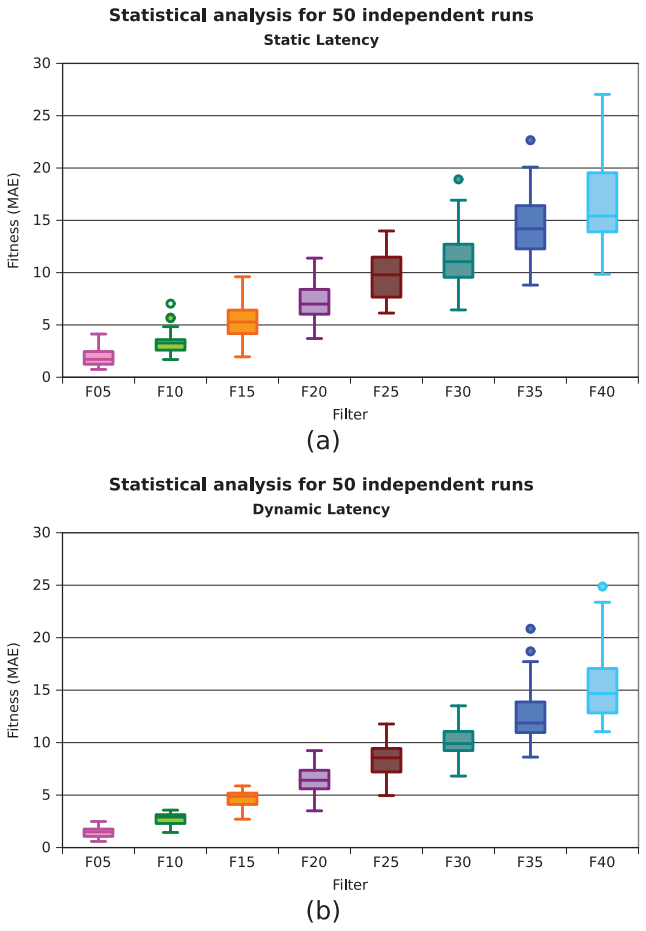

Fig. 7. Statistical analysis for the 50 independent evolutionary runs: (a) without latency selector and (b) with latency selector. For an explanation of axis labels, see Fig. 4. Each boxplot corresponds to the "*circles*" in each of the "Average versus Best" figures, i.e., the statistical analysis of the results for each of the 50 evolutionary runs (*F05* filter over *I05* image, etc.).

4.2 System Behavior for Impulse Noise

More tests were performed to check how the system behaves when different types of noise that are more difficult to remove shall appear in the input. The experiment was run in the same conditions as the previous one. Fig. 8 shows the results obtained in case impulse noise (random shots with unlimited intensity) is used in the training image. Average obtained MAE is slightly worse than in the case of S&P noise, as it is also the case for the median filter. However, better filters compared to the median one are still evolved even for this type of noise, proven to be more difficult to deal with (due to the noise pixels distributed in the whole luminance range, as opposed to the 0/255 pair of values in the S&P noise). This demonstrates how the system is able to adapt to and filter out conveniently more difficult types of noise, outperforming standard filters.

5 System Generalization

The experiments performed with different types of noise and noise levels are a proof that the architecture performs correctly but, to show that the solution can be generalized to other filtering tasks, and to all type of images, a generalization analysis was performed, checking that the architecture is valid for other image processing algorithms, and the results are valid for images different to the training one. To check the generality of the evolved filters, a set of 36 test images (sets 1, 4, and 5 from http://decsai.ugr.es/cvg/dbimagenes/g256.php) was filtered using the best evolved solution *F05* (evolved using 5 percent noise) for three different noise intensities for each type of noise and compared with the median filter, as shown in Table 3. Results show how the evolved filters consistently outperform the median filter for a wide set of images, except for the highest noise levels. Note, however, that the filter under test was evolved for low noise intensities and better performance has already been shown for the highest noise levels.

Fig. 9 shows a summary of the results for different types of noise. Horizontal lines represent different experiments, namely salt and pepper noise, burst noise, edge detection, and a combination of salt and pepper noise and edge detection. For all of them, the EA and the array are identical, and only the input training image and the reference image are changed for each experiment. Columns show the training image, the resulting output of the filter for the training image after evolution, and the reference image (evolution goal). The last two columns show the input and the resulting outputs for images different to the training one.

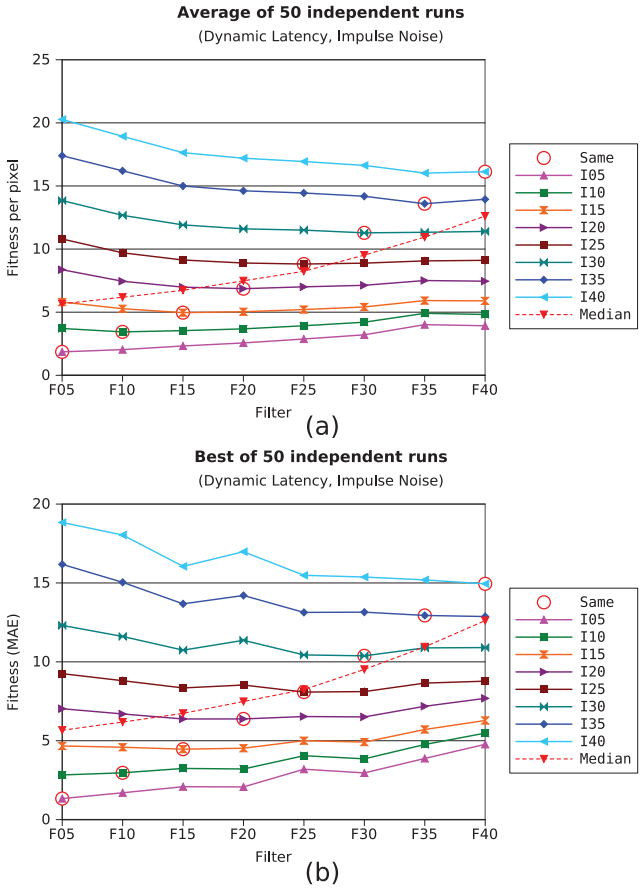

Fig. 8. Results of the adaptation experiments using latency evaluation for impulse noise: (a) average evolved filters and (b) best of the 50 runs. For an explanation of axis and legends labels, see Fig. 4.

All these results show how the system is able to evolve very different types of filters, behaving quite well when increasing noise levels appear at the input.

5.1 Cascaded Filtering

An interesting feature was observed when analyzing the evolved filters, showing that images do not degrade when filtered. An experiment on cascading several times the same filter was performed, to check how the fitness and visual quality behave. Results show how the subsequent filtering stages keep on improving the filtering result, removing more and more noise each time without producing artifacts in the image. This situation contrasts with the increasing blurring effect produced by the median filter. Fig. 10a shows the results of applying the best F05 filter to various image noise intensities for the training image for up to six iterations, comparing the results with the median filter. Fig. 10b is included to prove the generality of the filter behavior by doing the same test using a random image from the test set. Although there is not such a big difference as in the case of the training image, the same kind of behavior is still observed.

This feature can be exploited in scenarios where a high filtering quality is needed due to very noisy conditions. In these situations, more complex filters (e.g., if a 5×5 or a 6×6 array is available or any other *traditional* filter) would exhibit a better filtering behavior. However, if the available device does not have enough resources to cope with those complex filters, various filtering stages might be applied consecutively to improve overall filtering quality. On the other hand, two consecutive smaller arrays could probably fit in the device. In this case, EA would deal with a much smaller search space, greatly speeding the adaptation process, while keeping high filtering quality.

TABLE 3
Average MAE Obtained for a Test Set of 36 Images

Image set	Best F05	Median
5% S&P noise	0.689	5.245
15% S&P noise	2.802	6.033
30% S&P noise	9.170	8.645
5% impulse noise	1.804	5.267
15% impulse noise	5.354	6.152
30% impulse noise	13.052	8.580

Fig. 11 shows an example of how the filtering quality of the evolved filters (in the case of the aforementioned *F05*) looks like and how it compares with a median filter for various intensities of noise. Although some noise pixels still remain in the pictures in both cases, no blurring effect is produced by the evolved filters, as is the case with the median, and thus, more details of the original image are maintained as can be seen.

6 Implementation Results

The evolvable platform was implemented in a Virtex-5 LX110T FPGA included in Digilent's XUPV5 Evaluation Platform. Results are given for a 4×4 PEs array. Compared to the previous implementation [8], resource usage has been improved. Each PE needs now 5×2 CLBs of reserved logic, as opposed to the previous 40 CLBs, which means that reserved logic has been reduced one-fourth for the whole array. According to individual synthesis results per PE (needed to extract the partial bitstream), these occupy from 7 to 10 slices, depending on the specific PE functionality. Overhead due to the communication is also very high, since six of these slices are dedicated to bus-macro terminals. Table 4 shows post mapping implementation results offered by the ISE tool.

6.1 Timing Analysis

Two distinct modes of operation have to be considered to evaluate timing performance. One of them is the adaptation phase, when the component is evolving. Once a working circuit has been found, evolution is stopped and the system enters into its standard mode of operation. Within these phases, several different tasks need to be accomplished. In each generation during evolution, the EA creates new offspring candidate solutions (task T_{offs}) which are evaluated (T_{eval}) and assigned a fitness value (T_{fit}) before selection (T_{sel}) of the best fitted individual(s) for the next generation is done. This cycle is repeated throughout a given number of generations. To evaluate an individual, the array needs to be reconfigured to this candidate filter. Therefore, T_{eval} can be split into reconfiguration (T_{rec}) and filtering (T_{filt}). Finally, during normal system operation, images are filtered with the selected candidate, which corresponds with T_{filt}. Taking into account the associated

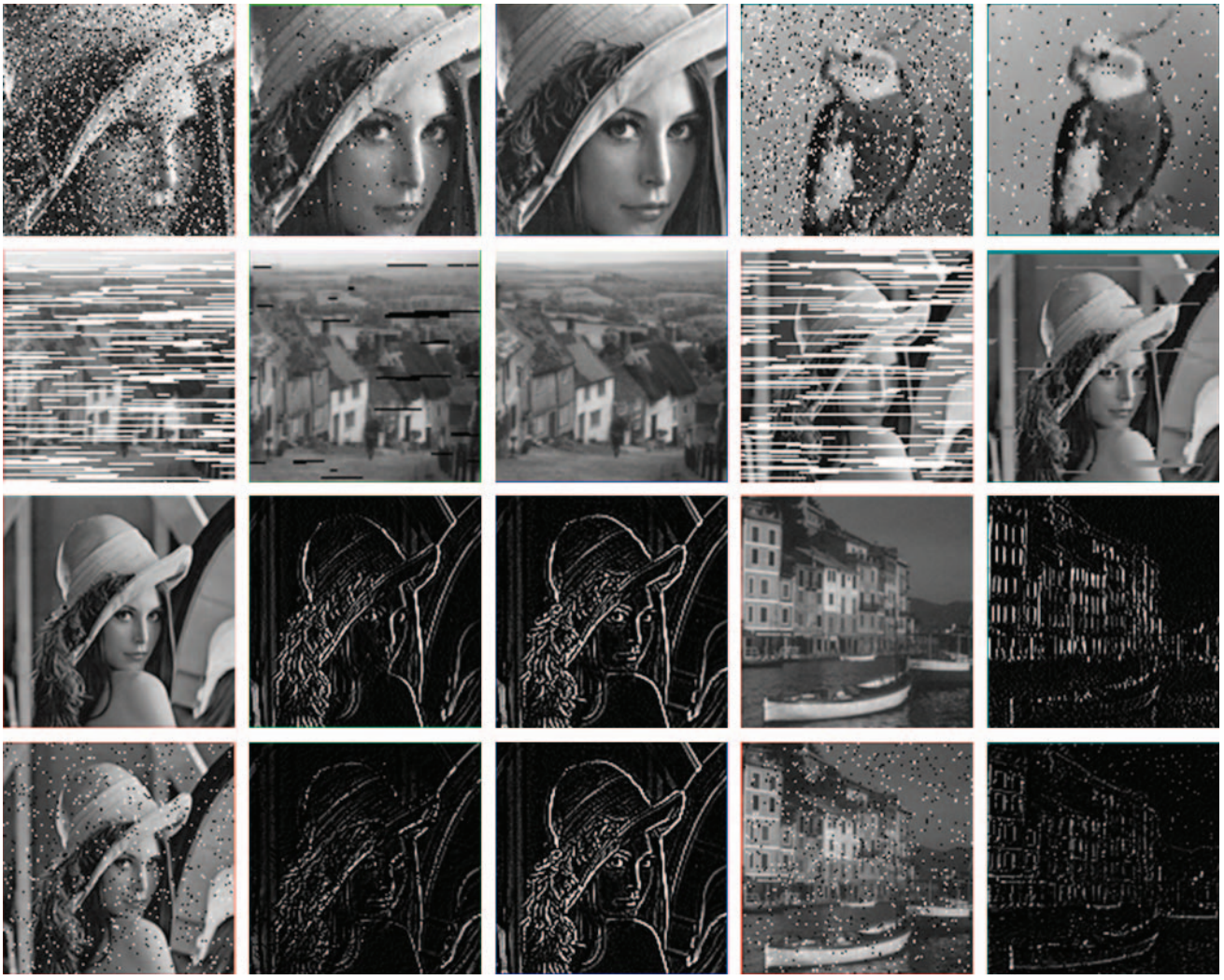

Fig. 9. Results of the system generalization experiments. Rows, from first to fourth, contain the experiments for different types of filters, namely salt and pepper noise, burst noise, edge detection, and a combination of salt and pepper noise and edge detection, respectively. First three columns show the training image, the filtered image after evolution is complete, and the reference image (evolution goal), respectively. The last two columns check the generality of the evolved filters by showing the result (fifth column) of filtering an input image not seen during evolution (fourth column).

times for each of these tasks (t_{offs} for T_{offs}, etc.) time elapsed in each generation during evolution can be written as

$$t_g \equiv \lambda \times \left(t_{offs} + t_{rec} + t_{filt} + t_{fit}\right) + t_{sel}, \tag{2}$$

where λ is the population size. Therefore, if N_g is the number of generations, the total time needed for evolution can be expressed as $t_{evo} \equiv t_g \times N_g$. Equation (2) can be simplified since the execution of some of the tasks is overlapped; the generation of new candidate configurations, which is done by the EA running in the microprocessor, and the evaluation of previous candidate solutions, which takes longer for the reference application. Besides, within evaluation, fitness computation and selection are also done in HW in parallel with the filtering process as output data are being generated. For that reason, (2) can be rewritten as

$$t_g \approx \lambda \times \left(t_{rec} + t_{filt}\right) \equiv \lambda \times t_{eval}, \tag{3}$$

where t_{eval} is the time needed to evaluate a single candidate circuit. Now, t_{evo} becomes

$$t_{evo} \approx \lambda \times N_g \times \left(t_{rec} + t_{filt}\right) \equiv N_{EVALS} \times t_{eval} \tag{4}$$

where N_{EVALS} is the total number of evaluations. Due to the inherent pipeline of the matrix, the evaluation of one candidate circuit requires $R \times C$ clock cycles after reconfiguration takes place, where R and C are the rows and columns of the training image, respectively. Therefore, filtering time is

$$t_{filt} \approx (lat + (R \times C)) \times 1/f, \tag{5}$$

where *lat* stands for the latency of the matrix, which compared to the $R \times C$ product can be neglected.

The timing analysis of the system shows two important factors. First of all, for the selected application, image processing, the evaluation of new candidates is expected to be the most time-consuming task. However, reconfiguration will take longer and will depend on the number of changed PEs (mutated genes). Timing results yield a maximum operating frequency of 200 MHz, measuring a reconfiguration time for a single PE of 31.84 μs. From a given candidate configuration to the next, changing k (three in this work) PEs involves changing also some other PEs

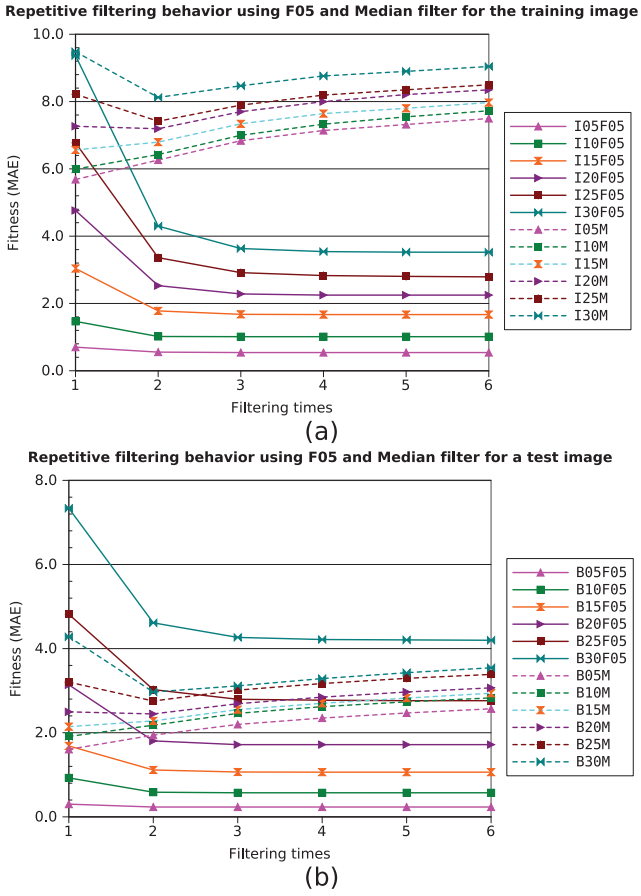

Fig. 10. Comparison of repetitive filtering behavior for F05 and median filter for (a) the image used for evolution and (b) a random test image.

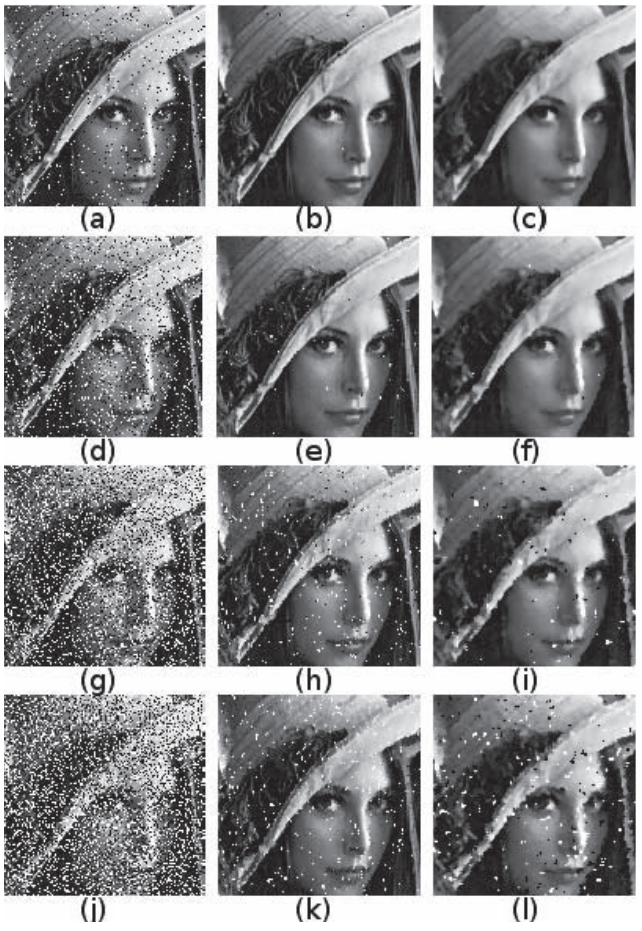

Fig. 11. Visual comparison of filtering performance: First column contains the original images for 5, 15, 30, and 40 percent noise, second column contains the result of the evolved filter performance (best F05), and third the median filter.

that are needed to return to the original, common parent. In this case, for three mutated genes, 139.22 μs is needed. Therefore, an evolutionary cycle of 100,000 generations (900,000 circuit evaluations for the selected $(1+8)-ES$) using a 128×128 size image has been measured to take 178 seconds. This timing performance is analyzed in the following section.

7 Virtual versus Native Reconfiguration

It has been mentioned previously that, to provide the evolvable IP core with adaptation features, some kind of reconfiguration capability is required. This is needed since the component needs to have a way to randomly adjust its internal processing architecture. There are two methods known for reconfiguration comprising structural changes in FPGAs; native and virtual reconfiguration. While native reconfiguration requires FPGAs with DPR capabilities, virtual reconfiguration can be implemented in any modern FPGA. There is also a third reconfiguration method proposed in [25] and analyzed in Section 2; an intermediate-level reconfiguration technique that consists of using the FPGA lookup tables as shift registers for reconfiguration purposes. However, this solution is very device dependent, and reconfigurability is limited to adapt LUT functions (besides only 25 percent of the Virtex-5 LUTs has this behavior).

The advantages and disadvantages of VRC versus DPR seem to be clear; extremely high virtual reconfiguration speed (just some clock cycles to write a big register) at the cost of extra delay and area overhead in the case of VRCs versus a higher operating frequency at the cost of a lower reconfiguration speed. The overhead in the VRC comes from the fact that every PE contains all possible functions synthesized and a multiplexer to select one among all of them, which makes maximum frequency lower compared to the native implementation. To quantify these differences for this specific architecture, a comparison has been carried out for both implementations.

For maintaining advantages of IP-based design such as modularity and reusability, reconfiguration details are hidden to the designer offering a common configuration interface. Therefore, two versions of the component's processing matrix featuring the two reconfiguration options mentioned previously have been created and evaluated.

TABLE 4
Resource Usage Results for the Evolvable Processing Architecture

Module	Slices	Slice Regs	Slice LUTs	LUTRAM	DSP	BRAM
Component	4803	9091	7436	558	0	38
Array	320	1280	1280	320	0	0
Misc	1101	1932	1932	64	0	12
HWICAP	1615	2765	2344	145	0	9
Mem. Ctrl	1767	3114	1880	29	0	17

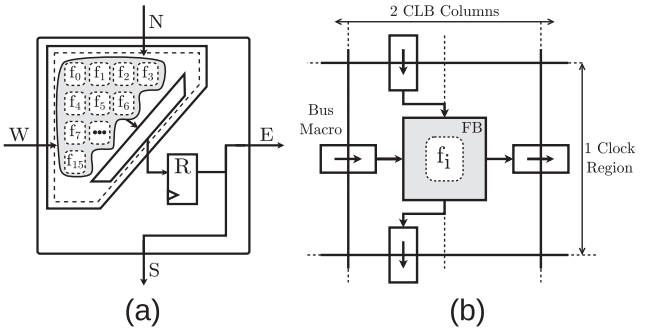

Fig. 12. (a) shows the abstraction of a PE in the VRC case and (b) using DPR as shown in Fig. 3 (register not shown for simplicity).

TABLE 6
Reconfiguration and Filtering Times for the VRC and the DPR Implementations

	Time (μs)		
Task	*RA@200MHz*	*VRC@100MHz*	*SW model*
Reconfig. (1 PE)	31.84	0.4	negligible
Reconfig. (3 PE)	139.22		
Filtering	82.12	163.84	~ 2 ms
Total evolution (s)	178	132	26 minutes

Average time measured during evolution. For a given circuit, changing k PEs involves changing also some PEs needed to return to the original, common parent.

In the following analysis, RA stands for reconfigurable array in the native reconfiguration version of the system. Fig. 12 shows a view on a PE from the DPR and the VRC point of view.

Regarding the VRC implementation, a highly functional description was used, which yielded moderate synthesis results. Table 5 shows a resource usage comparison. Results for the RA are repeated for clarity purposes. It can be seen that the RA uses around 1.5 times the resources of the VRC. Most of it is due to the RE (HWICAP and multiport memory controller, MPMC) needed to access the DDR2 memory containing the PE library. Therefore, the extra resources *wasted* in the VRC implementation are compensated by those needed to use DPR, so no big difference is observed here. However, the array size is clearly smaller using DPR, which will eventually make a difference, for example, in power consumption. Besides, the HWICAP and the MPMC can be shared among other parts of the final system, making their impact in resource usage much more balanced. Also, a higher array size is in favor of RA, since variable size-dependent resource usage is smaller.

Regarding timing performance, significant dissimilarities are expected due to the big differences between reconfiguration times and filtering times of each version. Filtering time will probably differ as a result of the extra delay expected in the VRC implementation due to the multiplexers. Table 6 shows the comparison of the results. Reconfiguration time does not depend on the number of mutated genes in the VRC, because it involves writing the whole configuration register for each configuration. However, it does in the case of the RA, since each mutated gene involves changing (reconfiguring) one PE, yielding a higher timing overhead in reconfiguration. Since filtering time is double in the VRC, the whole timing performance tends to compensate under these circumstances. However, it is still significantly higher, around 40 seconds (1.34 times), using DPR. The total evolution time in Table 6 also includes system initialization procedure (mainly image loading from external FLASH memories) and experiments result's logging. Although the target platform is an FPGA-based embedded system, a computer simulation was also done using a SW model built in C, for comparison purposes. Last column in Table 6 shows the total evolution time obtained with this SW model (based on the one reported in [6]). This simulation was run in a computer featuring an Intel Core i5 processor at 3.3 GHz. The result clearly shows how any of the FPGA implementations is vastly faster as expected from a hardware implementation.

A high number of circuit evaluations per second is obtained in both cases at the end of the evolutionary experiment; 5,056 for the RA and 6,818 for the VRC implementations respectively. The operating frequency of the final circuit is significantly higher in the RA case, yielding a higher throughput for normal system operation, i.e., when it is not under adaptation. In any case, a VRC would be able to achieve a frame rate of up to 48.23 images per second for full-HD resolution.

Compared to the results in [8], although resource usage (reserved logic) has been reduced one-fourth for the whole array, since PEs occupy now less than a frame (the minimum reconfiguration unit), reconfiguration time per PE is doubled. This is due to the fact that before overwriting the configuration memory with the new PE, a read must be done to preserve those PEs (within the frame) that do not need to be reconfigured.

TABLE 5
Comparison of Resource Usage Results for the Evolvable Processing Architecture Using DPR versus VRC

Version	Module	Slices	Slice Regs	Slice LUTs	LUTRAM	DSP	BRAM
	Component	4803	9091	7436	558	0	38
	Array	320	1280	1280	320	0	0
RA	Misc	1101	1932	1932	64	0	12
	HWICAP	1615	2765	2344	145	0	9
	Mem. Ctrl	1767	3114	1880	29	0	17
	Component	2791	4224	5472	128	0	12
VRC	Array	1096	215	2539	0	0	0
	Misc	1567	3676	2300	128	0	12

8 Conclusions

In this paper, we have shown that mapping evolvable hardware architectures that can evolve autonomously straight on the reconfigurable fabric of FPGAs, using DPR, is possible. Furthermore, evolution may adapt filter quality to varying noise intensities and types of noise, as well as the quality can be adapted to different quality levels, by cascading the same filter several times, since filters do not degrade the image. This, compared with traditional convolution filters, produces much better results for a wide range of noise densities. Besides, the proposed system is able to evolve filters for different tasks such as edge detection, including images corrupted with noise.

Future work will address the issue of autonomously detecting when an adaptation is needed. This involves the system should be able to detect that the environment changed (due to noisy transmissions, image sensor damage, etc.) so that different sources of data corruption appear in the input signal, such as different types and intensities of noise.

Regarding the extra fitness modules needed for the dynamic latency computation, they can be implemented also as reconfigurable elements, to take them out of the fabric when not needed. The same can be said for the original module, which is only needed during evolution/adaptation and during periodic system checks.

Also, a comparison between VRC and DPR, i.e., between virtual and native reconfiguration has been tackled in terms of area and timing performance. For modern six input LUT devices, the area overhead of VRCs is not as high as expected, although still using DPR is advantageous in terms of area. Regarding maximum working frequency, the VRC is in clear disadvantage, but it is still able to hold full-HD throughput. Nevertheless, deeper and further comparisons, including power consumption, are still needed. It is also worth to mention the effort needed to design the reconfiguration infrastructure needed for subframe reconfigurations. However, this effort has offered a useful set of tools that can be reused among different applications and for different devices.

Acknowledgments

This work was supported by the Spanish Ministry of Economy and Competitiveness under the project DREAMS (Dynamically Reconfigurable Embedded Platforms for Networked Context-Aware Multimedia Systems) with number TEC2011-28666-C04-02. The work of Lukas Sekanina was supported by the Czech Science Foundation project P103/10/1517 and IT4Innovations Centre of Excellence project CZ.1.05/1.1.00/02.0070.

References

[1] T. Higuchi, M. Iwata, D. Keymeulen, H. Sakanashi, M. Murakawa, I. Kajitani, E. Takahashi, K. Toda, N. Salami, N. Kajihara, and N. Otsu, "Real-World Applications of Analog and Digital Evolvable Hardware," *IEEE Trans. Evolutionary Computation,* vol. 3, no. 3, pp. 220-235, Sept. 1999.

[2] A. Stoica, "Evolvable Hardware for Autonomous Systems," *Proc. Congress Evolutionary Computation,* pp. 1-125, 2004.

[3] L. Sekanina, "Virtual Reconfigurable Circuits for Real-World Applications of Evolvable Hardware," *Proc. Fifth Int'l Conf. Evolvable Systems: From Biology to Hardware,* vol. 2606, pp. 186-197, 2003.

[4] A. Otero, R. Salvador, J. Mora, E. de la Torre, T. Riesgo, and L. Sekanina, "A Fast Reconfigurable 2D HW Core Architecture on FPGAs for Evolvable Self-Adaptive Systems," *Proc. NASA/ESA Conf. Adaptive Hardware and Systems,* pp. 336-43, 2011.

[5] S. Cagnoni, E. Lutton, and G. Olague, *Genetic and Evolutionary Computation for Image Processing and Analysis,* vol. 8. Hindawi Publishing Corporation, 2007.

[6] R. Salvador, A. Otero, J. Mora, E. de la Torre, T. Riesgo, and L. Sekanina, "Evolvable 2D Computing Matrix Model for Intrinsic Evolution in Commercial FPGAs with Native Reconfiguration Support," *Proc. NASA/ESA Conf. Adaptive Hardware and Systems,* pp. 184-191, 2011.

[7] R. Salvador, A. Otero, J. Mora, E. de la Torre, L. Sekanina, and T. Riesgo, "Fault Tolerance Analysis and Self-Healing Strategy of Autonomous, Evolvable Hardware Systems," *Proc. Int'l Conf. Reconfigurable Computing and FPGAs (ReConFig),* pp. 164-169, 2011.

[8] R. Salvador, A. Otero, J. Mora, E. de la Torre, T. Riesgo, and L. Sekanina, "Implementation Techniques for Evolvable Hw Systems: Virtual vs. Dynamic Reconfiguration," *Proc. Int'l Conf. Field Programmable Logic and Applications (FPL),* pp. 547-550, 2012.

[9] *Nonlinear Filters for Image Processing,* E.R. Dougherty and J.T. Astola, eds. Wiley, 1999.

[10] M. Sonka, V. Hlavac, and R. Boyle, *Image Processing: Analysis and Machine Vision.* Thomson-Eng., 1999.

[11] T. Higuchi and T. Niwa, "Evolving Hardware with Genetic Learning: A First Step towards Building a Darwin Machine," *Proc. Second Int'l Conf. Simulation of Adaptive Behavior,* pp. 417-424, 1993.

[12] A. Thompson, "Silicon Evolution," *Proc. Ann. Conf. Genetic Programming (GP '96),* pp. 444-452, 1996.

[13] R. Salomon, H. Widiger, and A. Tockhorn, "Rapid Evolution of Time-Efficient Packet Classifiers," *Proc. IEEE Congress Evolutionary Computation,* pp. 2793-2799, 2006.

[14] Z. Vasicek and L. Sekanina, "An Evolvable Hardware System in Xilinx Virtex II Pro FPGA," *Int'l J. Innovative Computing and Applications,* vol. 1, no.1, pp. 63-73, 2007.

[15] Z. Vasicek and L. Sekanina, "Hardware Accelerators for Cartesian Genetic Programming," *Proc. 11th European Conf. Genetic Programming,* pp. 230-241, 2008.

[16] K. Glette, "Design and Implementation of Scalable Online Evolvable Hardware Pattern Recognition Systems," PhD thesis, Univ. of Oslo, 2008.

[17] K. Glette, J. Torresen, and M. Yasunaga, "An Online EHW Pattern Recognition System Applied to Sonar Spectrum Classification," *Proc. Int'l Conf. Evolvable Systems: From Biology to Hardware,* pp. 1-12, 2007.

[18] J. Wang, C. Piao, and C. Lee, "Implementing Multi-VRC Cores to Evolve Combinational Logic Circuits in Parallel," *Proc. Int'l Conf. Evolvable Systems: From Biology to Hardware,* pp. 23-34, 2007.

[19] J. Wang, Q.S. Chen, and C.H. Lee, "Design and Implementation of a Virtual Re-Configurable Architecture for Different Applications of Intrinsic Evolvable Hardware," *IET Computers and Digital Techniques,* vol. 2, pp. 386-400, 2008.

[20] Z. Vasicek, M. Zadnik, L. Sekanina, and J. Tobola, "On Evolutionary Synthesis of Linear Transforms in FPGA," *Proc. Eighth Int'l Conf. Evolvable Systems: From Biology to Hardware,* pp. 141-152, 2008.

[21] Z. Vasicek and L. Sekanina, "Hardware Accelerator of Cartesian Genetic Programming with Multiple Fitness Units," *Computing and Informatics,* vol. 29, pp. 1359-1371, 2010.

[22] F. Cancare, M. Santambrogio, and D. Sciuto, "A Direct Bitstream Manipulation Approach for Virtex4-Based Evolvable Systems," *Proc. IEEE Int'l Symp. Circuits and Systems (ISCAS),* pp. 853-856, 2010.

[23] A. Upegui, "Dynamically Reconfigurable Bioinspired Hardware," PhD thesis, EPFL Lausanne, Thesis no. 3632, 2006.

[24] J. Torresen, G.A. Senland, and K. Glette, "Partial Reconfiguration Applied in an On-line Evolvable Pattern Recognition System," *Proc. NORCHIP,* pp. 61-64, 2008.

[25] K. Glette, J. Torresen, and M. Hovin, "Intermediate Level FPGA Reconfiguration for an Online EHW Pattern Recognition System," *Proc. NASA/ESA Conf. Adaptive Hardware and Systems,* pp. 19-26, 2009.

[26] J.F. Miller, *Cartesian Genetic Programming.* Springer-Verlag, 2011.

[27] A. Otero, E. de la Torre, and T. Riesgo, "Dreams: A Tool for the design of Dynamically Reconfigurable Embedded and Modular Systems," *Proc. Int'l Conf. Reconfigurable Computing and FPGAs (RECONFIG '12),* Dec. 2012.

[28] "Xilinx Modular Design Flow," http://www.xilinx.com/itp/xilinx7/books/data/docs/dev/dev0025 7.html, 2013.

Rubén Salvador received the BSc degree in telecommunication engineering from the Universidad Politecnica de Madrid (UPM), Spain, in 2001, and the MSc degree in electrical and electronic engineering from the Universidad de Alcala, Madrid, Spain, in 2004. Afterward, he received the MSc degree in industrial electronics also from UPM in 2007. He is currently working toward the PhD degree in industrial electronics at the Centre of Industrial Electronics (CEI), UPM. From 2005 to 2006, he was a research assistant in the Intelligent Vehicle Systems division at the University Institute for Automobile Research, INSIA-UPM, and until 2008, at CEI-UPM. In 2009, he was a visiting research fellow for four months in the Department of Computer Systems, Brno University of Technology, Czech Republic. Since March 2012, he has been a teaching assistant in the Department of Electronic Systems and Control (SEC), UPM, where he joined the Research Center on Software Technologies and Multimedia Systems for Sustainability (CITSEM). His research interests include evolvable hardware, high-performance reconfigurable and adaptive systems, evolutionary computing applications, and digital signal processing systems.

Andrés Otero received the telecommunication engineering degree from the Universidad de Vigo, Spain, in 2007, and the master's degree in industrial electronics from the Universidad Politecnica de Madrid in 2009. He is currently a researcher and working toward the PhD degree at the Center for Industrial Electronics, Universidad Politecnica de Madrid. His main research interests are reconfigurable computing, evolvable hardware, digital signal processing, and embedded system design.

Javier Mora received the BSc degree in industrial engineering from the Universidad Politécnica de Madrid (UPM), Spain, in 2011. He is currently working toward the MSc degree in industrial electronics at UPM. Since 2011, he has been a grant-holding researcher at the Centro de Electrónica Industrial (CEI), UPM. His current research area is in the field of fault-tolerant evolvable hardware.

Eduardo de la Torre received the MEng and PhD degrees from the Technical University of Madrid, Spain, in 1989 and 2000. He is currently an associate professor in the area of electronic technology at the Technical University of Madrid. His research has been centered in the last years in reconfigurable systems and architectures design for FPGA-based embedded systems. In the last five years, he has published more than 30 contributions in conferences and journals on the topic of reconfigurable systems, with emphasis on applying reconfiguration technology on restricted devices, but providing features such as autonomy, self-reconfiguration, self-adaptation, and self-healing.

Teresa Riesgo received the MSc and PhD degrees in industrial engineering from the Universidad Politecnica de Madrid (UPM) in 1989 and 1996, respectively. Since 2003, she has been a full professor of electronics at UPM. She has published a large number of papers in those fields and has participated and acted as main researcher in several European Union-funded projects. Since 2006, she has been involved in the International Relations Affairs of ETSII-UPM, and she has been vice-director of ETSII-UPM, vice president of the TIME network (www.time-association.org), and has represented UPM in different international agreements. Her research interests are focused on embedded-system design, wireless-sensor networks, reconfigurable systems, and power estimation in digital systems.

Lukas Sekanina (M'02-SM'12) received the MEng and PhD degrees from the Brno University of Technology, Czech Republic, in 1999 and 2002. He is currently a full professor with the Faculty of Information Technology, Brno University of Technology. His research interests include evolutionary design and evolvable hardware. He received the Fulbright scholarship to work with NASA Jet Propulsion Laboratory in Pasadena in 2004. He was a visiting lecturer with Pennsylvania State University and a visiting researcher with the University of Oslo in 2001. He has served as an associate editor of the *IEEE Transactions of Evolutionary Computation*, and editorial board member of *Genetic Programming and Evolvable Machines* journal, and *International Journal of Innovative Computing and Applications*. He coauthored more than 100 papers mainly on evolvable hardware. He is a senior member of the IEEE.