Evolvable 2D computing matrix model for intrinsic evolution in commercial FPGAs with native reconfiguration support

Rubén Salvador, Andrés Otero, Javier Mora,
Eduardo de la Torre, Teresa Riesgo
Centre of Industrial Electronics
Universidad Politécnica de Madrid
José Gutierrez Abascal, 2
28006, Madrid, Spain
Email: ruben.salvador@upm.es

Lukáš Sekanina
Faculty of Information Technology
Brno University of Technology
Božetěchova 2
612 66 Brno, Czech Republic
Email: sekanina@fit.vutbr.cz

Abstract

This paper addresses the modelling and validation of an evolvable hardware architecture which can be mapped on a 2D systolic structure implemented on commercial reconfigurable FPGAs. The adaptation capabilities of the architecture are exercised to validate its evolvability. The underlying proposal is the use of a library of reconfigurable components characterised by their partial bitstreams, which are used by the Evolutionary Algorithm to find a solution to a given task. Evolution of image noise filters is selected as the proof of concept application. Results show that computation speed of the resulting evolved circuit is higher than with the Virtual Reconfigurable Circuits approach, and this can be exploited on the evolution process by using dynamic reconfiguration.

1. Introduction

Unconstrained evolution in commercial FPGAs by direct bitstream manipulation is considered not possible because random safe modifications of the bitstream are not possible. Besides, its huge size turns the search space unmanageable, and, in addition, reconfiguration times using current manufacturers support are still high enough to prevent this technology to be embraced as a standard.

To overcome some of these limitations, evolvable hardware in commercial FPGAs has been mostly built up to date with the creation of an application specific reconfiguration layer on top of the FPGA known as Virtual Reconfigurable Circuit (VRC) [1]. Therefore, high reconfiguration speed and suitable computation granularity is achieved. However, the VRC introduces an area and delay overhead to the operation of the circuit as compared with a direct implementation of that functionality in the device fabric.

A different strategy is needed to implement evolvable circuits in FPGAs. Xilinx Dynamic and Partial Reconfiguration (DPR) design flow allows partial bitstreams to be dynamically loaded in the device. If there was a *library of reconfigurable components* defined as partial bitstreams, evolution could be performed *nearer* to the device fabric without virtualizing the reconfiguration mechanism in a VRC. This library of components can be seen as an *evolution* of the traditional ASIC *standard cells* but modified to support on-line hardware adaptation.

VRCs typically perform computations within an array of processing elements where data flows in one direction. Instead, our proposed architecture is defined as a highly regular and parallel two dimensional array of processing elements arranged as a systolic structure. The selection of this kind of 2D processing structure obeys to its widely known features in digital VLSI signal processing. However, after going through the state of the art the authors have not found any work studying whether this traditional architecture has been exercised to check its evolvability.

This work proposes such an architecture and its associated evolutionary framework aiming to demonstrate the architecture adaptation capabilities and the suitability of the proposal as a whole to be used for adaptive hardware in commercial FPGAs with native online reconfiguration for intrinsic evolution. Therefore, a detailed analysis and synthesis of the architecture is performed starting from an equivalent VRC. A software model is implemented and used to validate the architecture for noise removal in image preprocessing tasks.

The support needed for the FPGA reconfiguration task is introduced in [2] where a HW modular peripheral in charge

978-1-4577-0599-1/11/$26.00 ©2011 IEEE

of controlling the reconfiguration process with relocation capabilities is described. Presently, this block may achieve 250 MHz ICAP overclocking if combined with a fast link to an external memory as demonstrated also by other authors in [3], [4].

Next Section contains a description of the underlying motivation to tackle this work as our contribution to the current State of the Art, which is analysed in Section 3. Section 4 provides a full description and analysis of the architecture and the design steps given from the VRC paradigm and the implications it has on the genotypic level. We continue introducing the Evolutionary Algorithm (EA) implemented in Section 5 to finish showing the results obtained after running various tests in Section 6 before concluding the paper.

2. Motivation of this work

As it can be derived from the previous Section, the main motivation for this work is the integration of bio-inspired processing architectures into commercial FPGAs which contribute to solve the ever increasing demand on complexity and flexibility.

When dealing with highly distributed, networked, context-aware systems which may operate on very diverse environments, maybe even unknown or inaccessible at design time, it is very difficult to foresee all possible situations that may arise during system's lifetime. In addition, design and deployment methodologies for these systems are specific and application dependent. Therefore, providing them with adaptive behaviour would help in somehow simplifying the design while increasing re-usability and systems lifetime.

We envision a scenario in which a system with a set of tasks previously implemented in software is operating under certain conditions. At a given time, system maintainers (or the system itself under commands of a higher intelligence layer) may decide that running a hardware version of a particular task (or adding a new one to the initial set) would increase performance; however, let's assume that there is no hardware counterpart for this task. In such a case the device can trigger an adaptation phase to evolve a circuit which will eventually be able to perform that particular task. As a consequence, if evolution succeeds, a completely new circuit will have been automatically synthesized autonomously by the system, ideally without human intervention.

One of the main concerns in the evolvable hardware community to accomplish the implementation of these continuously adaptive systems, lies in the difficult task of supplying the system with an objective function which guides the change. However, in the suggested scenario, the system already has a functionally valid software task (or model) to be supplied as a goal to the EA whether when hardware acceleration is required by the system or in the case that any failure in the device happen to occur and a different need of adaptation arises. This approach can also be helpful if new tasks (new functionality required due to changing requirements) are added to the system.

One of the key technologies needed to succeed in this effort is a seamless use of DPR. However, device support is still limited in this field. What this proposal addresses is a use case of DPR in which a library of computational elements of different granularity is available at run-time in the form of partial bitstreams. Whether these are supplied by the manufacturers (as today are complex IP cores for integration at design time) or a seamless design flow is made available which facilitates its design, a standard, validated and functionally diverse library is needed, in a similar way as standard cells for ASIC design are used to build-up complex systems from smaller sub-components. With this on-line library, a suited EA can be implemented in the system to adapt its architecture by allocating-deallocating-reallocating components from the library in different positions of the device.

Our approach to the selection of an appropriate adaptive processing architecture for FPGAs may be seen from a slightly different point of view as that offered by most of the work which has been published up to date. Instead of defining an architecture tuned to the commonly used EAs to evolve some computational behaviour, our view is just the opposite; merging widely used and well performing processing architectures with an appropriately tuned EA so as to enable *adaptive processing-hardware* (as opposed to *processing adaptive-hardware*). In addition, it seems reasonable that those architectures with a higher probability to be successfully combined with bio-inspired adaptation strategies, are precisely those with a higher biological resemblance. For this reason we propose the use of a highly regular, parallel and interconnected two dimensional array of processing elements arranged like a systolic structure, with connectivity limited to the closer neighbours and with an inherent pipelining since this structure reminds typical (biological) cells layouts [5] while being widely used in VLSI signal processing.

3. Previous work on evolvable systems in FPGAs

Evolution of VRCs relies on Cartesian Genetic Programming (CGP) [6], which was created to allow the evolution of digital circuits at the gate level as an extension of Genetic Programming (GP). It describes a digital circuit as a directed graph, with a simple integer genotype which describes the functionality and connections of each node of the tree. The term *Cartesian* resembles the spatial

placement of the processing nodes in a grid where each node can be accessed through its Cartesian coordinates. It often uses a simple Evolutionary Strategy (ES) with small populations of 1 parent and between 4 and 10 children. Mutation at low rates is the only evolutionary operator since recombination does not seem to affect the search in an effective manner [7].

Further works extended CGP to functional level [8] where instead of logic gates, CGP operates with higher level components such as adders, comparators, shifters, etc. This proposal has been successfully applied to the evolution of image operators directly in FPGAs, whether the EA is running on a processor embedded in the device [9] or as a complete hardware implementation [10]. The mutation rate reported to be optimal for this task is 5% of the chromosome size, for a genotype length of 384 bits (VRC sized 8×4 functional blocks). The proposal found in [11] addresses an specific genotype-phenotype mapping inspired by enzyme biology which makes use of an *implicit context representation* so evolution of the system is independent of the position of genes within the chromosome.

Regarding the use of native reconfiguration in FPGAs, direct bitstream manipulation was initially proposed by Adrian Thompson who worked with currently obsolete XC6200 Xilinx chips where any (even randomly created) bitstreams were allowed to be used [12]. The XC6200 family was later replaced by the Virtex family in which the possibilities for direct evolution of configurations were restricted. Several attempts to implement evolvable systems were reported using JBits - a set of Java classes which provided an easy way to read and modify the bitstreams [13]. The JBits have not received enough support to become a widely accepted standard. Later platforms utilized the Internal Configuration Access Port (ICAP) that allowed the FPGA to be reconfigured internally. Upegui and Sanchez used the ICAP to manipulate only the LUT contents (in one dimension) while keeping a fixed routing [14]. Finally, Cancare, Santambrogio and Sciuto extended that concept to two dimensions exploring thus the capabilities of recent Virtex 4 devices [15]. However, only small circuits such as an 8-bit parity generator or 4-bit counter were evolved.

4. Proposed architecture and genotypic implications

This section addresses the design progress given towards the definition of a suitable architecture, starting from the VRC concept and adapting it progressively throughout the subsequent steps until an *optimised* architecture organization, from the reconfiguration point of view, is achieved. As different optimizations to the original architecture are proposed, consequent changes in the genetic representation of the individuals are analysed.

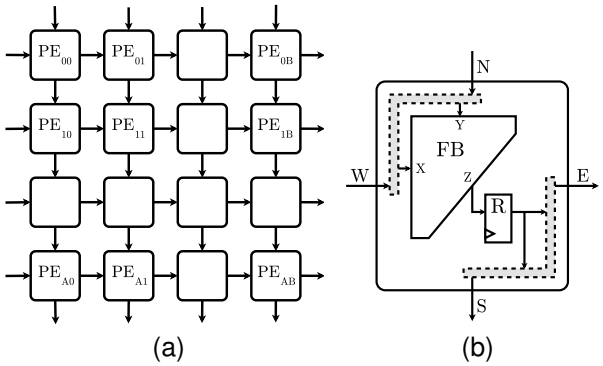

Figure 1. Proposed adaptive processing architecture (a) and internal configuration of a PE (b)

Figure 1a shows the proposed processing array, which is an $A \times B$ matrix of Processing Elements (PE) connected in a mesh-type fashion. It features a highly regular and parallel two dimensional processing array arranged like a systolic structure. Connections between blocks are fixed, but a certain data processing front adaptation is allowed, as will be shown below.

The final objective is to obtain an architecture with native FPGA reconfiguration, without the need of a redundant virtual reconfiguration layer. With the proposed library of partially reconfigurable and re-allocatable components we are able to cope with a similar degree of adaptation to the application at hand as compared to using VRCs.

Candidate circuit evaluation requires the region of the FPGA containing the processing matrix to be reconfigured. This process can be abstracted as replacing pieces in a puzzle. For each piece to allocate (PE to reconfigure) the reconfiguration engine indexes into the library as expressed by each gene in the chromosome, placing it in the correct position of the matrix (puzzle).

Getting inspiration from CGP as a valuable EA to evolve *similar* VRC-based architectures, a CGP-like encoding can be defined, where each gene can be encoded as a string of bits. Each gene acts here as a connection gene or as a functionality gene. Differently, in this work, since a library of partially reconfigurable components is available, another genotype can be defined in which each gene is an integer number pointing to an specific component of the library.

Contrary to previously proposed VRCs architectures, this 2D vertical and horizontal connection scheme is fixed to constrain the search space dimensionality. However, certain data processing front adaptation is allowed due to the proposed PE-FB connection mapping. The initial PE is composed of a Functional Block (FB), some additional routing logic and a flip-flop as shown in Figure 1b. Because of its already proved usefulness, the set of 16 functions chosen as a first approach is the same as the one defined

in [16].

4.1. Elaborating on the architecture definition

As explained previously, each element is defined not only by its functionality but also by the configuration of its connections. This means that the mapping from the inputs of the PE (W and N) to the inputs of the FB (X and Y) is specific for each element of the library. Therefore, if a PE is configured to the function F_i, PE_{F_i} (say, for example, *add*, PE_+), four different components (FB plus routing logic configuration) can be defined according to the configuration of the inputs, $\langle FB^1_+, FB^2_+, FB^3_+, FB^4_+ \rangle$. Each of these four different components maps directly to an element of the aforementioned library. The situation for this intermediate architecture (as it is being optimized) is shown in Figure 2a, which features a PE with 2 input multiplexers and additional routing logic in the output. The following mapping of the input connections applies:

$$N \rightarrow Y // W \rightarrow X \quad (1)$$

$$N \rightarrow X // W \rightarrow Y \quad (2)$$

$$N \rightarrow X // N \rightarrow Y \quad (3)$$

$$W \rightarrow X // W \rightarrow Y \quad (4)$$

Therefore, for each PE configured to a specific function F_i, there exist 4 different combinations according to the configuration of the input multiplexers. A similar analysis can be applied for the output if Z is routed to S, E or S and E simultaneously. This means another 3 options for each of the 4 input combinations which results in 12 possible different elements existing in the library for each defined function. This makes a total of 192 possible components to cover the full range of the 16 possible functions. Following this argument, each gene can be encoded using 8 bits so the implementation cost of the chromosome for the whole array would be $l = A \times B \times 8$ bits. If $A = 6, B = 6$ (for an array size similar to the state of the art) then $l = 288$ bits.

4.1.1. Optimizing the PE/FB connection mapping. If the previous routing proposal is carefully analysed some inconsistencies can be observed. For example, a PE in position (a, b) (column a, row b) may be evolved so that N is mapped to Y. However, the PE in position $(a, b-1)$ may be evolved to not route the output Z of the FB to the S port of the PE. To avoid this, the output routing logic is eliminated routing the FB output directly to both ports of the PE. This situation is shown in Figure 2b.

Now, for each PE configured to a specific function F_i, PE_{F_i}, there only exist 4 different elements according to the configuration of the input multiplexer, since the output routing has been eliminated. This makes a total of

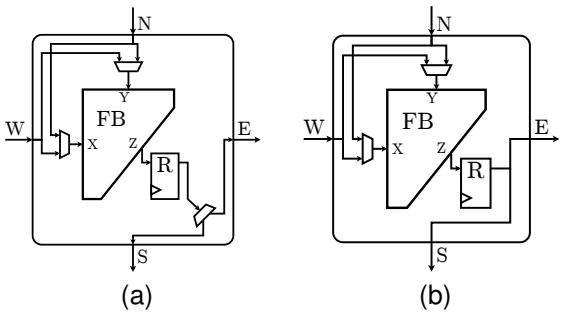

Figure 2. (a) shows the PE and its generic internal routing and (b) the PE with both outputs routed from the FB output.

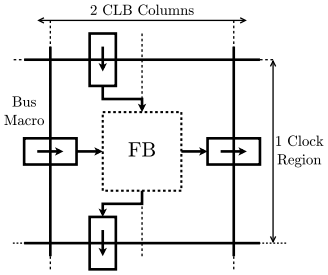

Figure 3. PE from the reconfiguration point of view

$16 \times 4 = 64$ possible components in the library for the 16 possible functions. In this case, each gene can be encoded with 6 bits, which yields $l = 6 \times 6 \times 6 = 216$ bits to encode the whole chromosome, reducing consequently the design (search) space. This connection optimization eliminates the need for the output routing logic, which besides saving resources, avoids the mentioned routing inconsistencies. However, constraints are imposed to evolution since data flow is somehow more restricted now.

The final objective is being able to allocate-deallocate-reallocate, at run time, different PEs in the array by means of DPR. To avoid the area overhead introduced by the multiplexers at the inputs of each FB, all of them are to be eliminated, so a different component will exist in the library of partial bitstreams for each possible combination of input routing connections and FB functionality, as explained previously. Figure 3 shows a PE from a DPR point of view. There is one *Bus Macro* [17] for each port of the PE (N, S, E, W), which works as the *anchoring point in the pieces of the puzzle*. In this way, one PE can be replaced by another one since all of them share a common connection interface defined by these Bus Macros. For this initial prototype, each PE has been oversized to occupy 2 CLB columns x 1 clock region in a Virtex-5 device.

Table 1. Set of components in the library.

Code	Function	Description
0	$N + W$	N + W (adder)
1	$N << 1$	N + N [a]
2	$W << 1$	W + W [a]
3	$N +^S W$	N + W with saturation
4	$N +^S N$	N + N with saturation [a]
5	$W +^S W$	W + W with saturation [a]
6	$(N + W) >> 1$	Average
7	255	Constant
8	$N >> 1$	Right shift N by 1
9	$W >> 1$	Right shift W by 1
10	N	Identity
11	W	Identity
12	$max(N, W)$	Maximum
13	$min(N, W)$	Minimum
14	$N -_S W$	Subtraction with saturation to 0
15	$W -_S N$	Subtraction with saturation to 0

[a] Improved implementation as a shifter

4.1.2. Optimizing by eliminating functional redundancies. Dividing the set of 16 FBs' functions according to its *arity*, 0, 1 and 2 operand functions are found on the set. In the case of 2 operand functions, we can further distinguish between commutative and non-commutative ones. Let consider again the example of the adder, PE_+, which as explained previously turns into 4 possible adder components in the library. However, since this is a symmetric operation, the two adders with an input configuration as (1) or (2) are just the same from a functional point of view. Therefore, they are merged into one single adder component, with default routing configuration as in (1); $N \rightarrow Y // W \rightarrow X$.

After adapting the set of 16 functions chosen to the architecture proposed and by applying the optimizations mentioned, the resulting set of library components raised to 29. This requires 5 bits to encode each gene, which yields $l = 6 \times 6 \times 5 = 180$ bits to encode the whole chromosome.

After some initial runs an additional reduction in the number of functions was done, choosing just those reported to have a higher degree of utilization for this task in [8]. The final set of 16 components included in the library can be found in Table 1, which reduces the chromosome length down to $l = 6 \times 6 \times 4 = 144$ bits.

4.1.3. Input/Output data strategy. While the previous analysis refers to the PEs themselves, the input and output data strategy at array level is analysed here. In relation to the output, there are $A+B$ output PEs (right/bottom borders of the array), so the decision of which is the correct one could be let to the evolution or fixed by design to any of the output PEs. As a first approach the right-most, lowest, PE is chosen as the matrix output since it seems to be a *reasonable* decision, given the natural data flow defined in the architecture, although some other data collection mechanisms at the output will be analysed.

Regarding the input, since the intended processing task is mask-based image noise filtering, a 3×3 window is used,

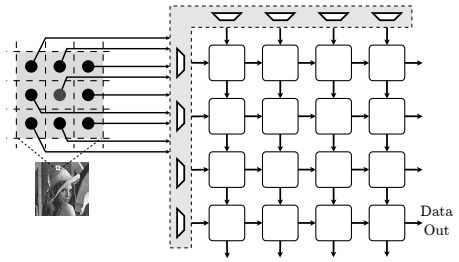

Figure 4. Input/Output data to the matrix

being let to evolution decide which pixels of the window are going to be connected to each of the input PEs (those situated in the left and upper borders). A new window will appear at the inputs in each clock cycle by moving one column to the right of the image. This means that each input PE needs an associated multiplexer. Figure 4 shows this input data strategy. For the selected array size of 6×6 PEs, 12 input PEs exist, each of them with an associated 16-bit input multiplexer able to select any of the pixels of the 3×3 window. In the genotype space, this results in 4 bits per gene, for a total of 12 input genes, which makes the chromosome grow in 48 bits.

5. Genetic representation of the problem

Previous section analysed the architecture and the implications that each refinement had on the chromosome length. Now the description of the EA is introduced. As explained previously, CGP encoding of the VRC concept has been the source of inspiration for this work, sharing many similarities with it. The chosen EA is a simple $(1+\lambda)$ Evolution Strategy in which each chromosome is composed of a set of integer numbers representing *input connection genes* and *functionality genes*:

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

where $InMux_i$ stands for the configuration of the input multiplexers of the matrix; PE_j is an index pointing to the library of components; and A and B are the height and width of the matrix of PEs respectively. The length l of the chromosome is therefore $l = (A + B) + A \times B$.

The initial population is generated randomly. After evaluation, which is described below, selection acts accordingly to CGP prescriptions. This means the fittest individual is selected as parent for the next population; elitism is enabled; and if two individuals score the same best fitness, diversity of the population is maintained by selecting the one which did not act as a parent in the previous generation (if this is the case). The new population consists of the selected parent and its mutants (no crossover operator). The mutation operator modifies k randomly selected genes from

this parent until the new population is complete. Uniform integer distribution (from 0 to 8 for input genes and from 0 to 15 for functionality genes) is used for this operator.

The evolution goal is to minimize the difference between the filtered image and the original image. Mean Absolute Error (MAE) is selected as fitness measure. To obtain the fitness value of a candidate filter a 3 step process is followed. First, the processing matrix is reconfigured with a candidate circuit; afterwards, the corrupted image is filtered; and finally the fitness value is calculated as:

$$MAE = \frac{1}{RC} \sum_{r=0}^{R\ 1} \sum_{c=0}^{C\ 1} |I(r,c) \quad K(r,c)| \tag{5}$$

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

Because of the inherent pipeline of the matrix, and due to the parallel fitness computation, the time of evaluation t_e for a single image can be expressed as:

$$t_e \approx (lat + (R \times C)) \frac{1}{f} \tag{6}$$

where lat stands for the initial latency of the matrix, whose calculation is tricky, because the way the data front propagates through the matrix changes with each candidate configuration. However, an approximation to it has been defined as the cycles it takes an input datum at PE_{00} to propagate until PE_{AB} which yields a reasonable expression for latency such as $lat = A + B \quad 1$. However, compared to the $R \times C$ product, lat can be neglected.

6. Experimental Set-Up and Results

To validate the proposed architecture and the EA, a high-level software model of the system shown in Figure 5 has been created using a mixed Python/C approach to combine ease of implementation with a relative processing speed (as compared to a HW implementation). In particular, the simulation results shown validate the EA as well as the 2D evolvable architecture and the fitness function utilized.

Different evolutionary runs have been done in order to identify a suitable initial set of parameters. Tests have been performed with matrix sizes ranging from 6×6 through 3×3. However, simulations of the 6×6 matrix were extremely slow, so the complete set of runs was not performed on it. Besides, no improvements compared to smaller arrays were observed in the runs performed so its results are not included here, as well as those of the 3×3 one, which were not able to evolve a good solution. Mutation rate k has been varied between 1% and 20% approximately.

As opposed to the proposed *reasonable* value for latency, initial runs seemed to offer better results for $lat = A+B \quad 2$, so this has been used for these tests. Table 2 shows the

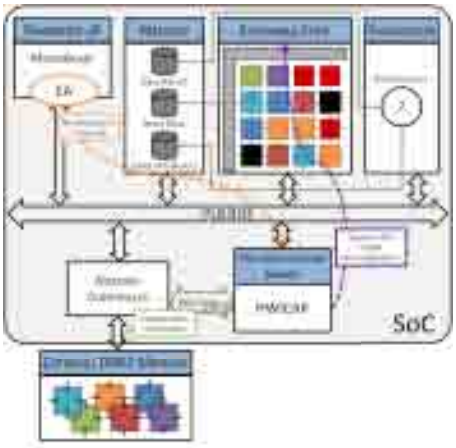

Figure 5. Proposed System Level Architecture

Table 2. MAE for different arrays and mutation rates

$A \times B$	k	$k(\%)$	**Best**	**Best Gen.**	**Avg.**	**Avg. Gen.**
5×5	1	2.86	2.9556	118	9.1326	150
$(l = 35)$	2	5.71	0.8111	30262	1.7614	20269
	3	8.57	0.4845	62345	1.3388	57192
	4	11.43	0.6607	33766	0.9875	40075
	5	14.29	0.8198	96625	1.1933	72991
4×4	2	8.33	1.2353	14870	2.4631	8062
$(l = 24)$	3	12.50	0.5640	32177	1.2798	28974
	4	16.67	0.7301	99292	1.0414	53675
	5	20.83	0.6553	56143	0.9812	71256

results obtained for various $(1+8)$-ES runs during 100000 generations. All tests were repeated 5 times.

Figures 6a–6c show the result of an average evolved circuit over Lena's image, which was used during evolution, while Figures 6d–6l show the result of applying that same circuit over a test set of 3 different images not seen during evolution to prove the generality of the solution.

6.1. Time of evolution

As it can be derived from Table 2, an average performing filter can be obtained within 50000 generations approximately. For the used population of $(1 + 8)$ that means an average number of 400000 evaluations, $AvgEvals$, are needed to obtain an acceptable filter. The time of *evolution* can then be expressed as:

$$t_{evol} = (t_e + t_{reconfig})\, AvgEvals \tag{7}$$

As stated in the Introduction section, if the enhanced ICAP overclocked at 250 MHz is used, approximately $12\ \mu s$ are needed to reconfigure a single PE. Since the maximum tested mutation rate is 5 elements per evaluation, a maximum of $60\ \mu s$ are needed for reconfiguration, while for a 256×256 pixels image the resulting time of *evaluation* (as in (6)) is $262\ \mu s$. Therefore, it can be concluded that

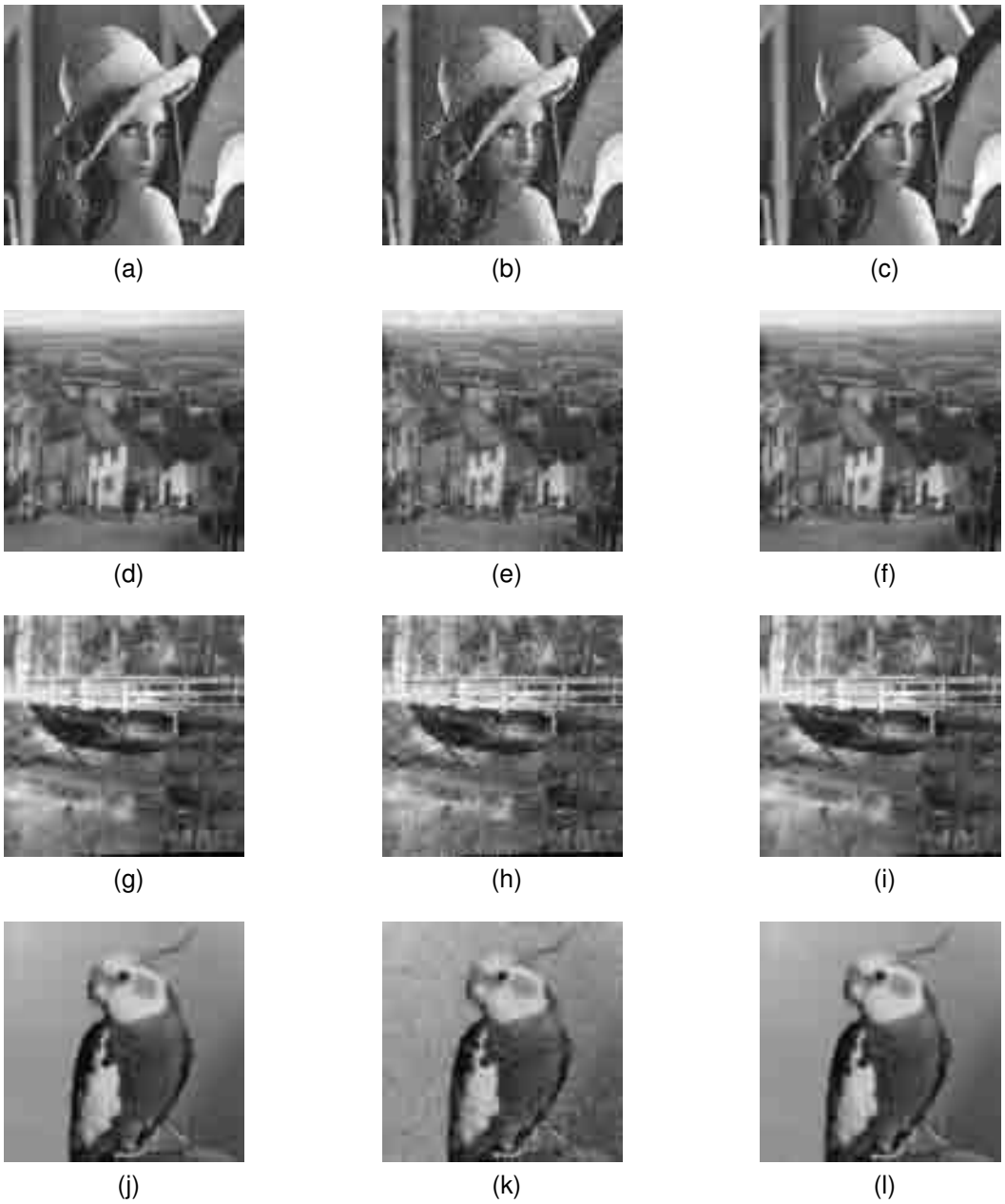

Figure 6. Result of applying an average evolved circuit: (a) through (c) to the train image used for evolution; (d) through (l) to an image test set to prove the generality of the solutions. Left column shows the original images; centre column the corrupted images; and left column the result of filtering

for the average 400000 evaluations needed, a total *evolution* time of 128 seconds are needed to obtain a working filter, which means around 3100 evaluations per second can be achieved.

7. Conclusions

This work proposes a 2D mesh-type, systolic, processing architecture widely used in VLSI signal processing which has been exercised to check its evolvability so as to analyse its feasibility as a solution for adaptive hardware in FPGAs with native reconfiguration support. Though it does not yet perform as well as other State of the Art proposals as VRCs

(but outperforms standard solutions as the median filter), it clearly shows its feasibility to be considered as a valid *adaptive processing-architecture*. Therefore, combined with our enhanced DPR engine, this complete evolvable framework represents a step forward in the implementation of adaptive systems in commercial FPGAs since it makes native reconfiguration possible, overcoming traditional VRCs drawbacks like area and delay overhead. Next step, which is already being accomplished, is the integration of the EA, the reconfigurable matrix and the DPR engine in the FPGA.

As future work on the EA side, we will focus on improving its search performance so the number of evaluations needed to find a working solution is reduced. An adaptive mutation rate scheme and crossover operators will be tested, as well as a fitness landscape analysis performed to better understand the search space. Also, a different EA such as a Genetic Algorithm will be tried out, which may be more convenient for this kind of combinatorial optimization problem. Besides, since matrix latency has arose as an important parameter to consider, it will be encoded as an extra integer gene in the chromosome to evolve alongside the solution.

Acknowledgement

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] L. Sekanina, "Virtual Reconfigurable Circuits for Real-World Applications of Evolvable Hardware," *Lecture Notes in Computer Science*, vol. 2003, no. 2606, pp. 186–197, 2003.

[2] A. Otero, A. Morales-Cas, J. Portilla, E. de la Torre, and T. Riesgo, "A Modular Peripheral to Support Self-Reconfiguration in SoCs," in *Digital System Design: Architectures, Methods and Tools (DSD), 2010 13th Euromicro Conference on*, 2010, pp. 88 –95.

[3] M. Shelburne, C. Patterson, P. Athanas, M. Jones, B. Martin, and R. Fong, "Metawire: Using fpga configuration circuitry to emulate a network-on-chip," *Computers Digital Techniques, IET*, vol. 4, no. 3, pp. 159 –169, 2010.

[4] C. Claus, R. Ahmed, F. Altenried, and W. Stechele, "Towards rapid dynamic partial reconfiguration in video-based driver assistance systems." in *ARC'10*, 2010, pp. 55–67.

[5] D. Mange, M. Sipper, A. Stauffer, and G. Tempesti, "Towards Robust Integrated Circuits: The Embryonics Approach," *Proceedings of IEEE*, vol. 88, no. 4, pp. 516–541, 2000.

[6] J. F. Miller and P. Thomson, "Cartesian Genetic Programming," in *Proceedings of the European Conference on Genetic Programming*. London, UK: Springer-Verlag, 2000, pp. 121–132. [Online]. Available: http://portal.acm.org/citation.cfm?id=646808.704075

[7] J. F. Miller, "An empirical study of the efficiency of learning boolean functions using a Cartesian Genetic Programming approach," in *Proceedings of the Genetic and Evolutionary Computation Conference*, W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith, Eds., vol. 2. Orlando, Florida, USA: Morgan Kaufmann, 1999, pp. 1135–1142.

[8] L. Sekanina, *Evolvable Components - From Theory to Hardware Implementations*, ser. Natural Computing Series. Springer Verlag, 2003.

[9] Z. Vasicek and L. Sekanina, "An Evolvable Hardware System in Xilinx Virtex II Pro FPGA," *International Journal of Innovative Computing and Applications*, vol. 1, no. 1, pp. 63–73, 2007.

[10] L. Sekanina and T. Martinek, *Evolving Image Operators Directly in Hardware*, ser. EURASIP Book Series on Signal Processing and Communications, Volume 8. Hindawi Publishing Corporation, 2007, pp. 93–112.

[11] Y. Zhang, S. L. Smith, and A. M. Tyrrell, "Intrinsic Evolvable Hardware in Digital Filter Design," in *Applications of Evolutionary Computing*, ser. Lecture Notes in Computer Science, G. R. Raidl, S. Cagnoni, J. Branke, D. W. Corne, R. Drechsler, Y. Jin, C. G. Johnson, P. Machado, E. Marchiori, F. Rothlauf, G. D. Smith, and G. Squillero, Eds. Springer Berlin - Heidelberg, 2004, vol. 3005, pp. 389–398.

[12] A. Thompson, "Silicon Evolution," in *Proc. of Genetic Programming GP'96*. MIT Press, 1996, pp. 444–452.

[13] G. Hollingworth, S. Smith, and A. Tyrrell, "The Intrinsic Evolution of Virtex Devices Through Internet Reconfigurable Logic," in *Proc. of the 3rd International Conference on Evolvable Systems: From Biology to Hardware ICES'00*, ser. LNCS, J. Miller, A. Thompson, and T. C. Fogarty, Eds., vol. 1801. Edinburgh, Scotland, UK: Springer, 2000, pp. 72–79.

[14] A. Upegui and E. Sanchez, "Evolving hardware with self-reconfigurable connectivity in xilinx fpgas," in *The 1st NASA/ESA Conference on Adaptive Hardware and Systems (AHS–2006)*. Los Alamitos, CA, USA: IEEE Computer Society, 2006, pp. 153–160.

[15] F. Cancare, M. Santambrogio, and D. Sciuto, "A direct bitstream manipulation approach for Virtex4-based evolvable systems ," in *Proceedings of 2010 IEEE International Symposium on of Circuits and Systems*. IEEE, 2010, pp. 853–856.

[16] Z. Vasicek and L. Sekanina, "Evaluation of a New Platform For Image Filter Evolution," in *Proc. of the 2007 NASA/ESA Conference on Adaptive Hardware and Systems*. IEEE Computer Society, 2007, pp. 577–584.

[17] *Xilinx Modular Design Flow*. [Online]. Available: http://www.xilinx.com/itp/xilinx7/books/data/docs/dev/dev0025_7.html