A Novel FPGA-based Evolvable Hardware System based on Multiple Processing Arrays

Ángel Gallego, Javier Mora, Andrés Otero, Rubén Salvador, Eduardo de la Torre and Teresa Riesgo

Abstract— **In this paper, an architecture based on a scalable and flexible set of Evolvable Processing arrays is presented. FPGA-native Dynamic Partial Reconfiguration (DPR) is used for evolution, which is done intrinsically, letting the system to adapt autonomously to variable run-time conditions, including the presence of transient and permanent faults. The architecture supports different modes of operation, namely: independent, parallel, cascaded or bypass mode. These modes of operation can be used during evolution time or during normal operation. The evolvability of the architecture is combined with fault-tolerance techniques, to enhance the platform with self-healing features, making it suitable for applications which require both high adaptability and reliability. Experimental results show that such a system may benefit from accelerated evolution times, increased performance and improved dependability, mainly by increasing fault tolerance for transient and permanent faults, as well as providing some fault identification possibilities. The evolvable HW array shown is tailored for window-based image processing applications.**

Index Terms—**Reconfigurability, adaptability; scalability; evolvable systems, evolvable hardware, fault tolerance, self-healing.**

I. Introduction

SRAM-based FPGA's can be reconfigured as many times as required, which is especially attractive if it is done autonomously and at run-time. Indeed, enhancing a system with Dynamic and Partial Reconfiguration (DPR) features may be a complex task, but it gives unrivalled flexibility and adaptability and, if used appropriately, it may contribute to improve system fault tolerance. DPR is based on replacing a portion of a working circuit by a different one, by writing a partial bitstream (PBS) into the corresponding positions of the device configuration memory. In the case of autonomous systems, PBSs can be taken from a library of presynthesized PBSs or generated in the device itself. However, autonomous embedded bitstream generation is complex, and only some basic transformations such as module reallocation or small functional and parametric modifications are affordable at run-time.

In this work, an evolvable HW (EHW) system is presented, which uses DPR for evolution. It is based on designing or transforming a circuit by using evolutionary techniques [1], i.e. an evolvable algorithm (EA) controls the generation of new circuits. The evolutionary loop is in charge of generating candidate circuits, evaluating their quality (fitness evaluation) and selecting the most appropriate candidate(s) as the parent(s) for the next generation. This technique opens the possibility of reaching solutions which cannot be conceived by conventional model-based design methodologies. If native FPGA reconfiguration is carried out at a fast enough rate, it can be used as the method to evolve one circuit to another one. This way, candidate's evaluation is performed on the programmable device itself, so that this evolution is noted as intrinsic. Intrinsic evolution has additional properties with respect to extrinsic or offline evolution. These include higher evolution speeds, but it should be mainly stressed the autonomy and self-adaptation features offered by this technique.

Besides adapting the circuit functionality to deal with run-time changing specifications, intrinsic evolution may be used to circumvent and recover from faults appearing in the reconfigurable fabric, due to aging or to high-energy particles reaching the device [2]. This way, if a permanent fault is produced, a new evolution process may generate candidates that avoid using such a portion of the circuit, recovering its functionality as much as possible. If this is done autonomously, the system gets self-healing properties.

When building intrinsic EHW systems on dynamically reconfigurable FPGAs, there exists different strategies, which range from direct bitstream manipulation, considered unfeasible due to the extremely large design space to explore, to coarse grain approaches [3], where complete functional blocks are reconfigured. As a trade off solution, authors presented in [4] an architecture, which is a 2-D mesh-type array of fine-grain Processing Elements (PEs), working in a systolic way. Each PE has a specific connectivity and functionality, and it can be selected from a library of presynthesized PBs stored in an external memory. Evolution is in charge of deciding which PE is reconfigured in each position of the array, until obtaining the desired functionality. Basic self-healing properties of this system were reported in [5].

In order to increase both the adaptability and the self-healing features of the basic EHW system presented in [4], authors propose in this work the replication of the evolvable array, building an evolutionary structure with a variable number of parallel processing structures arranged in a flexible manner, which includes independent, cascaded, parallel and bypass operation modes. In addition, novel evolutionary strategies and self-healing mechanisms are provided in this work, suited to the proposed architecture. Main benefits derived from the use of a variable number of arrays, as it will be reported throughout this paper, are:

- System evolution is accelerated due to the evaluation of candidates in parallel. In addition, a novel evolution strategy, called evolution by imitation, is proposed, that permits to evolve with no reference data sets, reducing the criticality of such component.
- Processing quality may be improved in several ways: during cascaded operation, better fitness values may be obtained by splitting up the task into several stages, with circuits adapted to each one. Also, in parallel mode, higher processing throughputs are achieved. Therefore, more complex tasks can be addressed.
- New fault-tolerance and fault-recovery strategies based on array processing redundancy are proposed. In particular, new TMR schemes can be implemented, so that the fitness function can be used as a fault diagnosis technique, useful for surviving with accumulated permanent faults.
- Bypass mode allows keeping operation while a single array is being re-evolved due to a new permanent fault.

A wide range of window-based digital image filters has been used for demonstration of these proposals.

The paper is structured as follows. An analysis of the state of the art is shown in section 2. The single array architecture and the transformations to make it scalable are shown in section 3. The modes of operation are presented in section 4, while self-healing mechanisms are tackled in section 5. Results on the benefits of their use and the impact on resource utilization are shown in section 6. Conclusions are drawn in section 7.

II. Analysis of the State of the Art

SRAM-based FPGA's are prone to faults due to many factors, especially those that work in space, where devices are affected by aging and high-energy radiation. Both produce two kinds of faults: transient faults or Single Event Upsets (SEU), and permanent faults or Local Permanent Damage (LPD) [6]. SEUs occur more frequently than LPDs, but LPDs should not be ignored, mainly in long-life missions, such as deep space exploration. This paper shows techniques to deal with both of them.

Self-healing is the capability of autonomous recovering from a fault, or a series of faults, trying to minimize system degradation effects. This property is one of the major processes of an autonomic computing system [7] [8], and it has a vital role to play in system's reliability. In order to achieve a self-healing architecture, a fault recovery mechanism or a combination of several elements must be implemented.

Fault recovery mechanisms in FPGAs can be classified in *offline* and *online* methods [9]. The former imply stopping the data processing while the healing process is active, while the latter involve being able to keep processing data while the system is faulty or being repaired, with the advantage of keeping the system running and therefore increasing the system's availability.

On the one hand, within the offline mechanisms, an interesting approach is the use of the inherent self-healing properties provided by the use of EAs. This has been analyzed in [5], [10] and [11], in combination with EHW. Apart from using the EA as a method for finding an optimized solution to solve a problem, this technique works as a self-healing technique because faulty designs are treated as sub-optimal solutions in the evolution, and therefore they are discarded in favor of a better solution, so the fault is avoided, or *healed*. Some authors also explore the approach of evolving not only the logic blocks, but also the routing blocks [12]. This may benefit if the fault is placed in a routing element, but it is done at a lower granularity level and a specific EA is needed in order to achieve admissible results.

On the other hand, examples of online mechanisms are redundant systems, such as *triple module redundancy* (TMR), which is widely used and allows mitigating faults by implementing the same logic three times and a voting system that compares the results obtained. Different approaches of TMR and variations are TMR/Simplex, TMR+Scrubbing, and the TMR + Lazy Scrubbing + Jiggling architecture proposed by Garvie et al. [13], which extend the avoidance of a fault (provided by TMR), and the recovery of a transient fault (using a scrubbing process, i.e. reading the configuration memory to check for faults, and re-writing it in case that any fault is found), allowing the system to be able to recover from an LPD by imitation of the others behavior. This is achieved by using an EA, which in this work is the same that is used to generate an adapted filter. Thus, an online fault recovery is achieved, because the system keeps running, but with an offline method, since the faulty element stops its processing task.

III. Architecture of the Evolvable Array

The single array evolvable system used as the starting point for this work was presented by the authors in [4].The first part of this section is therefore included to make this paper self-contained. Transformations performed to make it scalable, being capable to increase or decrease the number of arrays,are shown later.

A. The Single-Array Evolvable HW System

Figure 1 shows the SoPC architecture of a single-array evolvable system. Main elements are the reconfigurable circuit, the evolutionary algorithm and the reconfiguration engine. The array is the reconfigurable circuit, where the functionality of each Processing Element (PE) can be changed by DPR, taking the PBSs from an external DDR memory. Candidate circuits to be configured in the array are generated in the EA that runs on the embedded microprocessor (a MicroBlaze) where, after selecting a candidate, either randomly for the first generation or choosing the best candidate of the previous generation, makes a mutation in the genotype (the set of coded values that defines exactly one solution and allows to create the phenotype, i.e. the implementation of the circuit described by the genotype), and calls the reconfiguration engine to create a new circuit. Afterwards, the array takes either a reference image during evolution, or a real image taken, for instance, from a camera, in normal operation. A fitness function, the so-called *Mean*

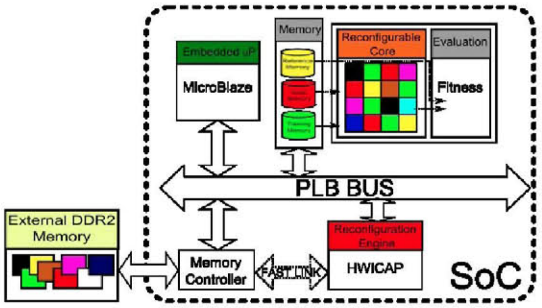

Fig. 1. *Internal architecture of a single array evolvable hardware system*

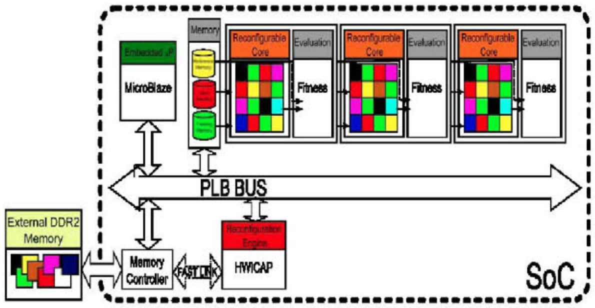

Fig. 2. *Internal architecture of a 3-stage evolvable hardware system*

Absolute Error (MAE), compares the quality of the resulting image with the expected result, and from here, a new candidate is selected.

The input training image and the reference image are stored in the flash memory, and they specify what filtering application is required. For instance, if a noisy image is set as training image, and the noise-free image is set as a reference, the EA will generate a noise reduction circuit. However, if the training image is the noise-free one, and the reference is set to the edge detected image, the circuit will converge to an edge-detection filter. This way, during system life-time new functionalities can be obtained, only by providing the system with the corresponding training and reference images.

Regarding the reconfigurable module, every PE within the array matrix can perform one operation with one or two inputs. Inputs are either the west (W) or the north (N) sides, or both, and data is always propagated, after a register that allows pipelined execution (improving speed), to both the south (S) and east (E) outputs. By eliminating redundancies and symmetries, the library of available PEs was reduced to 16 different elements, which allows the corresponding gene coding in 4 bits. For a 4x4 array, there are eight inputs, four in the north side and four in the west side. Every input has a 9-to-1 mux, letting the EA to select one out of the nine pixels of a sliding window which performs the computation of the central pixel in the output. The output of the array is one of the four outputs on the east side, and this selection is controlled also by the EA, which selects with a mux one of them.

The reconfiguration engine used in this system was presented by the authors in [14]. This module is capable of reading PBSs from an external memory or from the configuration memory itself, providing fast reconfiguration and relocation capabilities. Therefore, it may be used to insert, copy or move HW blocks within the reconfigurable fabric.

Regarding the evolutionary framework, getting inspiration from Cartesian Genetic Programming (CGP), a simple *$(1+\lambda)$ Evolution Strategy* with 1 parent and λ offspring has been implemented. More details can be found in [4].

B. Scalable Array Architecture Modifications

In order to make the system scalable, some modifications are required, as shown in Figure 2, where the multiple array architecture is presented. The modified architecture pretends to be effective for having a variable number of arrays. Enhanced. system dependability and functional adaptability to a wider number of applications, or with a better performance, are the main motivations for this architecture.

When designing an architecture for scalability, there exist different approaches. For instance, scalability can be static or dynamic, depending on whether the number of arrays is defined at design time or at run-time. In this work, scalability for a variable number of arrays was implemented using the same design principles as in [15], where generic dynamically scalable processing cores are presented. Main proposal of this work is that a change of dimensions of the architecture is tackled with a proportional change of the footprint of the core, leading to scalable footprints depending on performance or functional requirements. Thus, in this case, the more demanding are the system requirements in terms of self-adaptation and self-healing, the larger is the area occupied by the EHW system on the device. In addition, to reduce the cost of changing the dimensions of the architecture, processing cores are designed in a highly modular way, where each possible size is obtained by combining a different number of basic modules. In this case, each processing array with its corresponding controller, the structures to compute and to deal with the variable latency of the arrays, some FIFOs to align data and the fitness unit are envisaged as a unique module, so that the EHW architecture can grow by changing the number of those modules instantiated in the design. This basic module is referred as Array Control Block (ACB), and its internal architecture is shown in Figure 3.

The first ACB is connected to the static part through a vertical connector at the top, and it has a congruent connection in the bottom side to stack vertically with the next ACB. The connections between ACB and array are horizontal, so arrays also stack vertically. A self-addressing scheme was designed so that every control register in any ACB can be easily addressed by the EA in the MicroBlaze. The control registers allow different modes of operation of every individual array, as well as reading fitness and latency values.

The ACB structure is such that any array may be fed with a common input image, or an image coming from the previous array. The fitness computation block may compute the pixel aggregated MAE between the reference image and the output image of the array, but it may also be set to calculate MAE between the input and output images of the array, as well as MAE between the output and another output from an adjacent array. These settings enable different evolution modes, as it will be described in the next section. According to the design principles in [15], scalable arrays with multiple arrays can be directly built up by assembling the required number of these modules. In this version, the number of arrays is fixed during system lifetime, but in the future a dynamically scalable approach will be implemented.

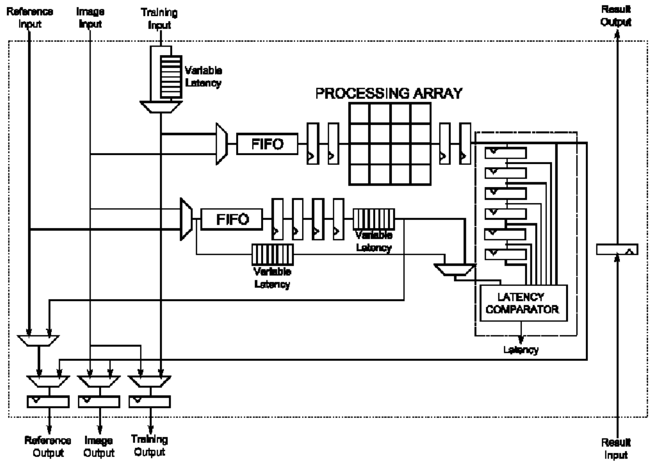

Fig. 3. *Internal architecture of a single stage of the Evolvable Hardware Platform*

Compared with the single array system, more logic resources are required, but in contrast, new operation modes are allowed, which results in benefits in terms of enhanced quality performance, platform adaptability and self-healing, as described in the next sections.

IV. EHW System Operation Modes

The strategy followed to enhance both the dependability and adaptability of the proposed EHW system, is based on providing its internal structure with such a flexibility that, besides the reconfiguration of the PEs within each array, is also possible to modify the connectivity between them, both during adaptation and mission times. Thus, different operation modes have been envisaged. Each mode offers different self-adaptation and self-healing capabilities, and therefore, it is tailored to deal with different processing requirements, as well as to work under different hazard conditions.

In the rest of the section, processing and evolution category modes are described, while drawbacks and benefits offered in each case are discussed in the results section.

A. Processing Modes

At mission time, the set of arrays can be arranged in four basic modes, which are a) Cascaded, b) Bypass, c) Parallel and d) Independent, as shown in Figure 4.

In *Cascaded mode*, the output of an array is taken through a 3 image lines FIFO to rebuild the 3x3 window, and fed to the next processing array. This scheme provides two types of functionality, which will be referred as collaborative or independent. They are shown below.

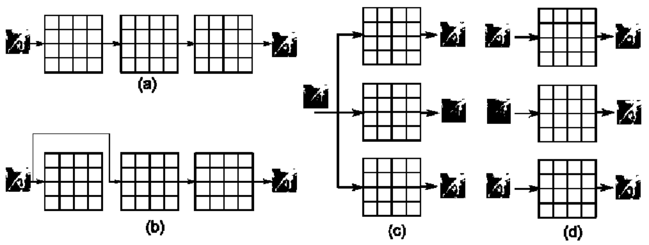

Fig. 4. *Processing Modes of the architecture: (a) Cascaded, (b) Bypass and (c) Parallel and (d) Independent*

The *Collaborative Cascaded* mode is based on splitting up the filtering task into subsequent stages, all of them trying to achieve a common target (the reference, zero-noise image). In this case, each filtering stage is different from the others, since each one is specialized for processing the output of the previous stage. Thus, the more stages in the chain, the more processing elements working together to achieve the common goal, and therefore, more complex tasks can be addressed.

Similarly, *Independent Cascaded* filters are also supported. In this mode, different filters are also used in each stage, but in this case, each one is in charge of a different task, such as noise removal, followed by a smoothing filters, and then edge detection. Therefore, instead of working together for the same purpose, each stage is specialized in a different task, and it will be obtained by evolving against different reference images.

On the other hand, *Bypass mode*, shown in Figure 4-b, is a variant of the Cascade mode, but one or more stages are disconnected and replaced by a bypass connection between its input and output. Bypassed array still receives its input data stream. This mode is the key of one of the self-healing strategies subsequently proposed in this work.

Parallel mode is based on arranging the processing arrays in such a way that all of them receive the same input, which is therefore filtered simultaneously. In this case, with three stages, different processing arrays may work as in a Triple Modular Redundancy (TMR) mode.

Finally, it is possible to configure all the arrays in *Independent mode*, to carry out the same or different processing tasks on the same or different input sources. This aims just at accelerating processing by means of several processing units in parallel, and therefore is not further analyzed in this work.

B. Evolution Modes

Evolution during system adaptation can be also carried out in different modes. The choice depends on the desired processing mode and the fault-tolerance strategy to be applied. Proposed evolution modes are referred as Independent, Parallel, Cascaded and Imitating.

Independent evolution is the simplest strategy. In this case, each array is evolved with its own reference, which allows adjusting them to different processing tasks. Therefore, this strategy may be used to obtain filters to work in the Independent processing mode, the parallel redundant mode, the independent cascade or parallel modes during mission. All arrays need to be evolved in a sequential manner. This situation has been already evaluated in [4].

On the other hand, *Parallel evolution*, shown in Figure 5-b, is based on the distribution of the offspring generated during each generation of the evolution phase among the different processing arrays, in order to reduce the time required to obtain a suitable solution. In this case, the reference image is delivered to the three arrays simultaneously. Each one evaluates a single candidate solution, and therefore, several fitness values are computed in parallel. Thus, it can be used in the same cases the independent evolution is used, but achieving better evolution times. Timing results are offered in the experimental results section.

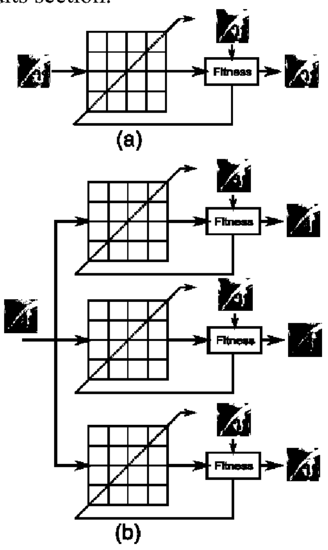

Fig. 5. *Independent and Parallel Evolution Mode*

Beyond the acceleration obtained applying parallel evolution, cascaded-specific evolution modes are also implemented in the platform.

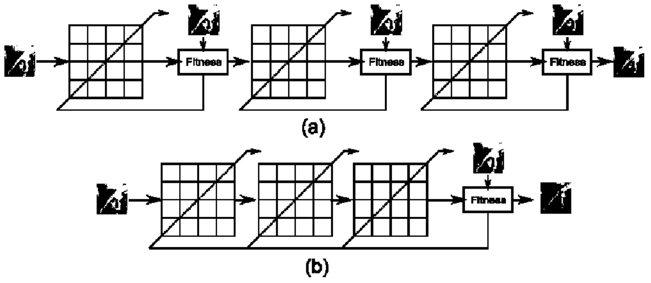

Fig. 6. *Cascaded Evolution Modes. In (a) cascaded evolution with separated fitness unit, and (b) cascaded evolution with a single fitness unit.*

In *Cascaded evolution modes*, each array in the sequence can be evolved considering the results of the rest of the arrays in the processing chain. More specifically, two cascaded modes have been envisaged, cascaded evolution with separate fitness computation, and cascaded evolution with merged fitness. In the case of *evolution with separate fitness units*, shown in Figure 6-a, each array is evolved considering its own fitness, but using the same reference image in all arrays. Furthermore, the output of the previous array is used as the input for the evolution of the subsequent array. On the other hand, evolution may be guided by a *single fitness unit*, shown in Figure 6-b, so all candidates are selected or rejected jointly. For each of these modes, two variants, simultaneous and sequential evolution, have been developed. In the case of *sequential cascaded evolution*, adaptation of array *i+1* is carried out once array *i* is finished. Differently, *simultaneous or interleaved cascaded evolution*, is based on moving forward a single generation in each array sequentially, and therefore, the adaptation of all of them is carried out together. In both cases, a different chromosome is kept for each array. These cascaded evolution modes drive to the Collaborative Cascade operation mode.

Another envisaged evolution mode is *Evolution by Imitation*, shown in Figure 7. This mode is one of the main proposals of this work, and it is based in the connection of a filter in bypass mode with respect to another, while the evolution of the bypassed filter is carried out evaluating the MAE between its output and output images of a neighboring filter. Ideally, this fitness metric should get close to zero, showing that one functionally equivalent filter was obtained from another working filter. Although this technique seems to be pointless, since just copying the genotype should produce the same result, it is a key technique to recover from a permanent fault, without using the reference image, which might have disappeared, damaged, or erased. So, a given filter can learn online from another one in the chain, just by imitating it. This is similar to the jiggling technique proposed in [13], but its application is not restricted to a TMR scheme for self-recovery.

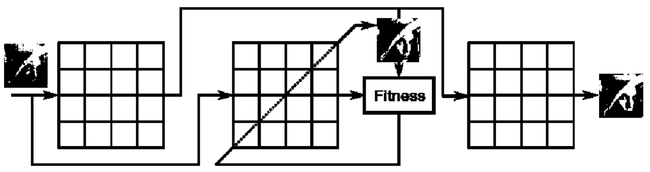

Fig. 7. *Evolution by imitation Mode*

V. Self-Healing Strategies

Besides the autonomous capability of the embedded platform proposed in this work to adapt itself to different processing tasks, it offers an inherent tolerance against injected faults. In the case of a single array, this feature has been already shown in [4], where a systematic fault analysis was carried out, injecting faults in each position of a single 4 × 4 processing array. Results showed that the system can self-recover from permanent faults by launching an evolution stage whenever such a fault is detected. Therefore, the same mechanism used to adapt the platform allows its self-recovery from permanent and accumulated faults. The number of supported faults depends on the characteristics of the filtering problem. On the other hand, transient faults, like SEUs, do not need to launch another evolutionary run, since it can be done by means of scrubbing. So, the detection of a permanent fault is obtained after detecting that the fault cannot be removed by a previous scrubbing operation.

In this work, since multiple evolvable arrays are able to work in different operation modes, more advanced self-healing strategies are proposed. Thus, techniques based on evolvable hardware are combined with other strategies, such as Triple Modular Redundancy. In the rest of the section, the proposed self-healing mechanisms are described both for cascaded and parallel operation modes.

A. Self-Healing strategy combined with fault recovery based on evolution by Imitation

Self-healing strategy corresponding to the cascaded operation modes is composed by the following steps:

a) Run initial evolution (Using either single or parallel mode) and select a working circuit, for each array in the platform.
b) Keep track of the individual fitness value of each array, by using a calibration image.
c) Run normally until next calibration.
d) Re-evaluate fitness.
e) If fitness from b) and d) in every array are equal, no fault is detected. →Go back to c).
f) If not, rewrite last reconfiguration (Scrubbing) in the damaged array.
g) Reevaluate fitness with pattern image.
h) If fitness from g) and b) are equal, then fault was transient. → Go back to c).
i) If fitness in g) > fitness from b) → Fault is permanent. Set faulty array in bypass mode, and either re-evolve with a reference image (if available), or launch an evolution by imitation process.

Error detection is carried out by means of the periodic application of calibration images, which must provide a known fitness value, in case the array is not damaged. However, having multiple arrays allows recovering from a fault without having training images available in the system, once the initial evolution has finished. This situation may appear in case training images are removed from memory to save resources, or if a fault appears in the memories storing the images. Differently, the damaged array is able to deal with the permanent fault by learning from the closer neighboring array, in order to recover from the fault. In order to do that, the array is configured in Bypass mode, with respect to the array it is learning from. This situation is shown in Figure 8.

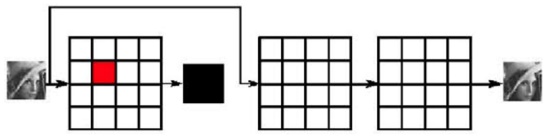

Fig. 8. *The faulty array is configured in bypass mode to learn by imitation from a neighboring array,*

This approach allows system to be recovered online, without stopping the filtering stream. This way, criticality of each PE under faults is reduced, since data stream is never stopped. In order to recover properly, the fault free filter should be processing the images accordingly to what the apprentice is desired to imitate.

B. Self-Healing based on TMR for Parallel Processing Mode

In this case, platform robustness could be improved by means of the parallel operation mode, which would allow implementing a TMR strategy, including a pixel voting strategy, as well as a fault recovery mechanism. This situation is shown in Figure 9. To achieve this, two different voter modules are implemented, depending on fitness comparisons or by pixel by pixel comparisons of the processed image outputs. Both voters are implemented in hardware, so the comparison would be at run-time. Fitness voter is able to detect, after each image filtering, if a fault has occurred. On the other hand, the output pixel voter is able to keep the system working with no fault impact. In this mode, only three parallel arrays are considered. The overall mechanism is the following:

a) Run initial evolution (Using either single or parallel mode) and select a working circuit. This circuit is configured in each one of the three arrays of the platform in parallel mode.
b) Compare online the individual fitness of each array, using the fitness voter module.
c) Run normally until a fitness divergence is found. If no fault is detected→Go back to b).
d) Rewrite last reconfiguration (Scrubbing) in the damaged array.
e) Reevaluate fitness with pattern image.
f) If fitness from d) and e) are equal, then fault was transient. → Go back to c).
g) If fitness in e) > fitness from d) → Fault is permanent. Launch an evolution by imitation process.
h) Ideally, if the imitating process reaches zero fitness value, the exactly same functionality is implemented in the faulty array. If not, the new configuration obtained can be pasted in every array to keep the validity of the TMR voter.

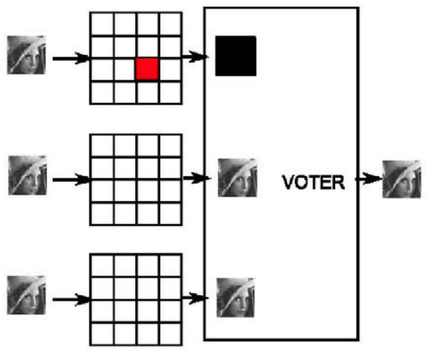

Fig. 9. *Fault-tolerant Strategies in parallel processing mode are shown.*

The self-healing strategy proposed for the parallel processing mode also offers benefits compared against the self-healing mechanism available in the case of a single array. For instance, it is able to keep data processing throughput under the presence of a single fault. Furthermore, it is able to detect faults autonomously, without requiring the use of an image for calibration, with the corresponding savings in processing time. Moreover, faults are mitigated due to the existence of the TMR voter in the output of the filtering stream. Therefore, when a single filter is misbehaving, a valid output can still be provided. A particular situation appears after the recovery from a permanent fault. In this case, expected fitness from the damaged filter may be different to the undamaged counterparts. To cope with this situation, a similarity threshold can be defined in the voter. In this case, an error is detected in case fitness disparity is out of the threshold. Compared to traditional TMR approaches, this technique allows autonomous fault surveillance even under a considerable number of permanent faults, which is one of the main contributions of this work. Pixel-level and fitness level voters work separately, such that, if the application supports that a faulty image may be processed, the pixel voter may be removed, keeping fitness based diagnosis as a lightweight mechanism for fault detection.

VI. Experimental Results

The validation of the proposed platform has been carried out considering both the adaptation capability of the evolvable arrays, as well as the fault-tolerance and self-healing features. Different operation modes and adaptation strategies are evaluated experimentally in this section. Also, a modified evolution strategy is proposed specifically for this multiple array system in order to improve evolution time taking into consideration the nature of the intrinsic evolution with native DPR. Experimental results provided in this work have been obtained with the system implemented in a medium size Xilinx Virtex-5 LX-110T FPGA.

A. Resource utilization

A snapshot of the floorplanning of the system is shown in Figure 10. In this case, three stages have been implemented. In future work, this number will be dynamically changed. Each PE occupies two CLB columns wide by one quarter of a clock region height (5 CLBs), each array occupies eight CLB columns of a clock region, which means a total of 160 CLBs. Reconfiguration time obtained with the reconfiguration engine used in this work is 67.53µs per PE. This result is obtained with the ICAP working at nominal 100MHz. Since each PE uses less than a clock region, configuration data allocated in the position of the PE has to be read back before reconfiguration. This process is carried out automatically by the Reconfiguration Engine, using its readback / relocation / writeback feature.

The control logic of the static part in charge of addressing and managing the ACB registers consumes 733 slices, requiring 1365 FFs and 1817 LUTs. Every ACB requires 754 slices, with 1642 FFs and 1528 LUTs.

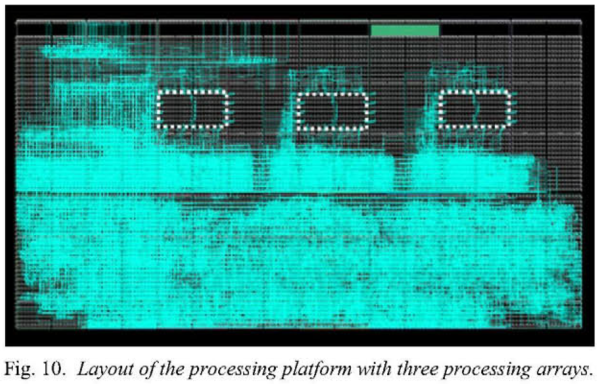

Fig. 10. *Layout of the processing platform with three processing arrays.*

B. Speed-up Experiments in Parallel Mode. New Evolutionary Algorithm

In the first experimental setup, speed up of independent versus parallel arrays is analyzed. In order to achieve better performance in the evolution time, the chromosomes mutations are done in groups of three at the same time, configuring the three arrays, one with each chromosome, instead of the classical evolution method, testing the chromosomes one by one. Nine chromosomes are generated in every generation, so the working diagram with one array or with three arrays is as shown in Figure 11. As shown in this figure, the only process that can be parallelized is the evaluation of the solution circuits, due to the fact that there is just one reconfiguration engine in the system. Mutation of the chromosomes is done in software, simultaneously to the evaluation process of the previous candidate(s), to improve the performance of the system.

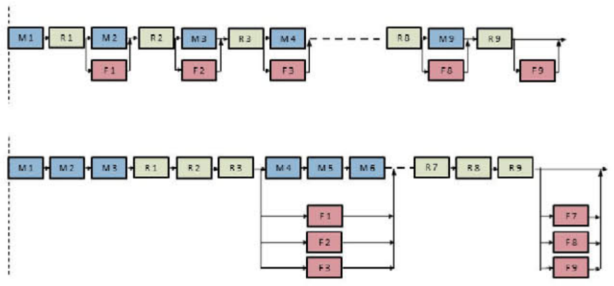

Fig. 11. *Diagram of the process carried out in each generation with one array (upper) and with three arrays (lower).M stands for mutation, R for reconfiguration, and F for fitness evaluation*

Figure 12 shows average evolution time for 50 runs of 100,000 generations each (these values will be kept constant throughout all experiments), using different mutation rates, for both single and parallel schemes. It can be observed that, the higher the mutation rate, the higher the reconfiguration time, and time differences are kept when increasing the mutation rate. So, a fixed time saving is achieved in the evolution process (around 50 seconds).

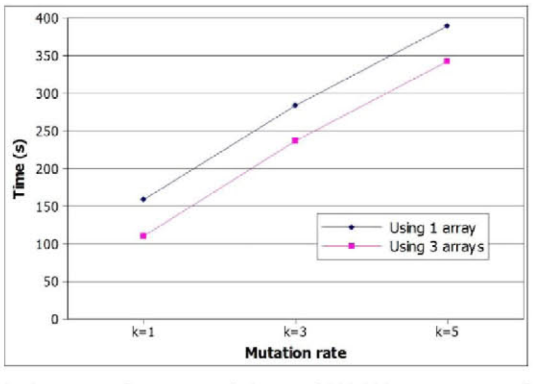

Fig. 12. *Average evolution time of 50 runs of 100,000 generations each, with different mutation rates (128x128 pixels).*

The speed up is limited since the reconfiguration time is higher than the evaluation time, and it consumes a high percentage of the time spent in every generation. However, if evaluation time was higher, which might happen, for instance, if images to be filtered are larger, benefits of Parallel evolution mode clearly increase. This case is shown in Figure 13, were the system has been modified in order to filter images of

256x256 pixels, four times the original size. In this case, more acceleration is achieved, increasing the time differences between the single array version versus the three arrays one. Time savings are also constant (around 200 seconds in this case), and higher than in the previous case, showing that the impact of reconfiguration time versus evaluation time is important to determine the speed improvement.

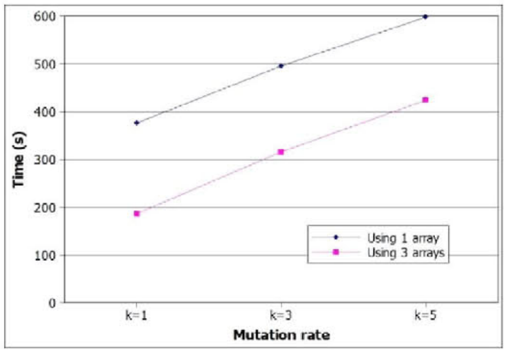

Fig. 13. *Average evolution time with different mutation rates for 256x256 pixels images (50 runs, 100,000 generations).*

However, looking at the two previous figures, evolution time always depends strongly on the mutation rate. In order to reduce this dependence, a new evolutionary strategy is originally proposed. It is as follows: the first parallel evaluation of every generation (in this case, the first three chromosomes) are created by mutating the selected chromosome from the previous generation with the usual mutation rate, but the other parallel evaluations of the same generation (six chromosomes) are created by mutating the chromosomes of the previously generated ones, but these mutations are always done with low mutation rate (k=1). Thus, every evaluated circuit is similar to the previous one, and so, fewer reconfigurations are carried out in every generation. Evolution time obtained with this new strategy, compared with the old one, is shown in Figure 14.

Furthermore, results it terms of fitness obtained with this strategy are equal or even better than the previously obtained results, as it is shown in Figure 15. It has to be noticed that the lower the fitness value is, the better the solution. Therefore, the new strategy, which was mainly created to reduce evolution time, also provides better results in terms of fitness, and so, on filtering quality.

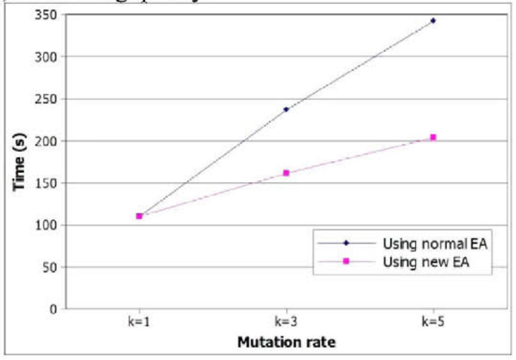

Fig. 14. *Comparison of average evolution time (50 runs, 100,000 generations) with different mutation rates using new and old EAs*

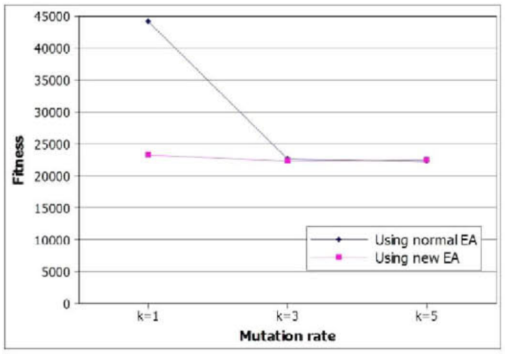

Fig. 15. Comparison of *average fitness results with different mutation rates*

C. Filtering quality experiments in Cascaded Mode

In order to show the benefits of using a cascaded evolution mode prior to cascade operation mode, comparisons between the collaborative cascaded modes and an iterative approach (every array stage holds the same circuit, that is, same chromosome) are shown in Figures 16 and 17. They show average and best fitness results obtained for every stage of the cascade filter.

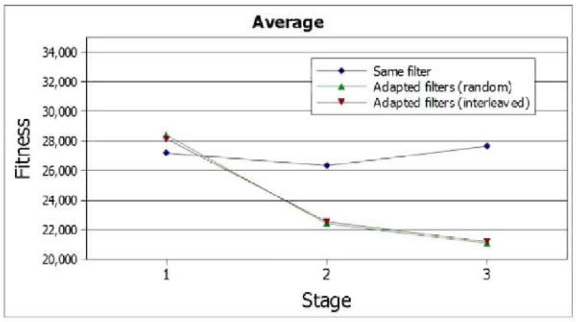

Fig. 16. *Average results comparison for cascaded modes*

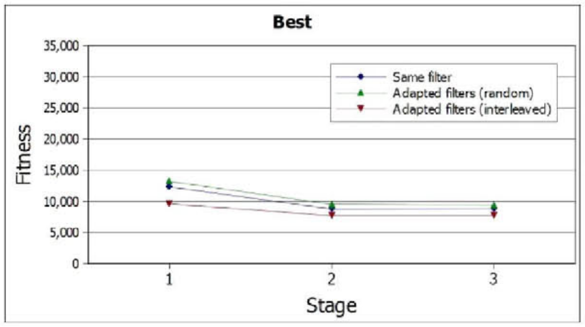

Fig. 17. *Best results comparison for cascaded modes.*

As it can be seen, using the same configuration in all filters yields a results improvement from the first stage to the second, but it gets worse in the third stage of the filter. On the other side, adaptive filters obtained with cascaded evolution modes have much higher improvements in all stages, getting better global results. This is because every filter stage is noise level specific. Comparing the two cascaded evolution approaches, that is, sequential cascaded evolution and interleaved cascaded evolution, results show that there is very little fitness difference between both modes.

An example of noisy input image and the result of a three stages adapted filter are shown in Figure 18. The input image has a salt & pepper noise with 40 % noise level, and the resulting image quality is very high, with a MAE fitness value of around 8000. It must be pointed out that the conventional reference filter for such type of noise is the median filter. It yields a MAE result which is far above this one, more than twice the value obtained for just one stage, and it is not cascadable.

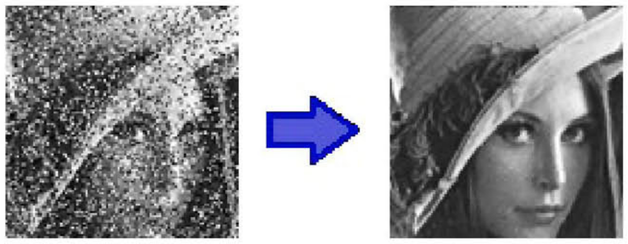

Fig. 18. *Input and output images of a three stages adapted cascaded filter.*

To take the decision about which operation mode is suited to each situation, different system parameters have to be taken into account. Thus, the selection must be motivated by the primary system goal, which may be processing throughput, in the case of parallel operation, or adaptability to more complex tasks, in the case of cascaded modes. The selection between redundant and collaborative cascaded, as has been described in the experimental results, depends on the specific filtering problem features. On the other hand, if multiple different tasks have to be implemented, both independent mode and independent cascaded modes are to be selected. Regarding the bypass mode, its main purpose is to work within a Self-healing strategy, for cascaded systems, as described next. The decision on the mode during evolution, has to be taken according to the selected operating mode.

D. Fault Tolerance and Self-Healing Experiments

Fault emulation is carried out using the same mechanism that is used during adaptation, that is, the DPR achieved by the reconfiguration engine. Thus, rather than simulating the fault, it is injected dynamically in the platform by means of the reconfiguration engine. Therefore, faults are generated reconfiguring dynamically the desired position of the array, with a modified bitstream corresponding to a dummy PE, which generates a random value in its output. This fault model will be referred as PE-level model, since a fault in any element inside a PE produces misbehavior in its output. Using a hardware based fault analysis, allows offering a systematic fault analysis, by injecting faults in every position in every array of the architecture.

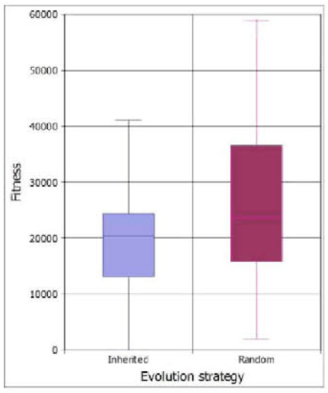

Fig. 19. *Results comparison between two different evolution strategies in the evolution by imitation.*

Using this fault injection mechanism, and recovering by imitation, we have observed that the imitation on faulty arrays performs better if the starting genotype from which evolution is started is the same as the non-faulty one, instead of a random generated one, as shows Figure 19. The fitness value in an imitation setup corresponds to the difference between the output image of the master and the output image of the faulty array. It should tend to zero (threshold is considered to be around 100 of MAE, while random values are about 3 orders of magnitude above this value), which is enough to say that both evolved systems are almost identical. With two permanent fault injections, or even more, a fitness reduction is still achieved, but the limitations imposed by the accumulated faults avoid the apprentice to work as well as the master.

In case it is applied to the TMR parallel operating mode with a fault in one of the arrays, a complete functional recovery is achieved in the best cases. In Figure 20, the complete strategy is shown.

The situation depicted in the figure corresponds to, first, three arrays working in parallel with the same results. Then a fault occurs in one array, which is detected by an increment in the fitness value. When that happens, an evolution by imitation process is launched, and after some generations, around 40,000, the faulty array is completely recovered. The combination with the scrubbing process, to determine if the fault is transient, is not shown in the figure.

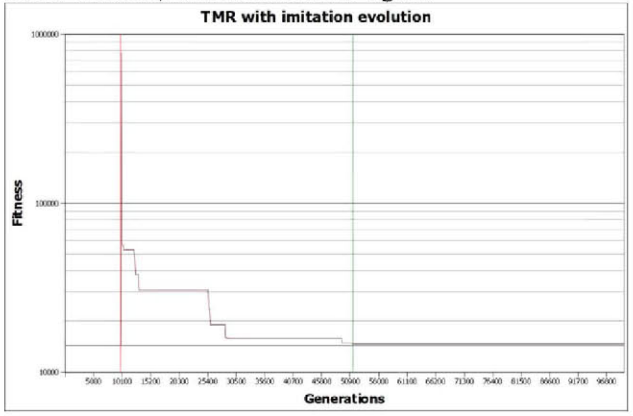

Fig. 20. *TMR mode with the injection of a fault and the recovery of the system.*

VII. Conclusions and Future Work

The evolvable hardware architecture proposed in this work offers a wide range of possibilities that enhance performance in various cooperative formats, both in evolution and operation modes. In particular, it has been proved that evolution time may be reduced by applying parallel evolution modes, for which a new evolutionary algorithm modification, based on two-level mutation is proposed, saving extra time with even better functional results. Also, the cascaded modes offer unrivaled quality, which could be adjusted by selecting a variable number of stages. Finally, the evolution by imitation is proposed at array level which, seamlessly combined with other fault mitigation and recovery techniques such as scrubbing or TMR, offer additional protection mechanisms against transient and permanent faults.

In future work, small additional modifications will make the arrays to be individually scalable. Also, after analyzing the criticality of all elements in the system, an overall fault resistance assessment, with realistic fault models, needs to be performed.

Acknowledgment

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.

References

[1] Sakanashi, H.; Iwata, M.; Keymulen, D.; Murakawa, M.; Kajitani, I.; Tanaka, M.; Higuchi, T.; , " IEEE International Conference on Evolvable hardware chips and their applications,", 1999, vol.5, no., pp.559-564

[2] A.M. Tyrrell, G. Hollingworth, S.L. Smith, "Evolutionary strategies and intrinsic fault tolerance" Proc. 3rd NASA/DoD Workshop on Evolvable Hardware. EH-2001, IEEE Comput. Soc, pp. 98-106.

[3] Torresen, J.; Senland, G.A.; Glette, K., "Partial Reconfiguration Applied in an On-line Evolvable Pattern Recognition System," NORCHIP, 2008. , vol., no., pp.61-64, 16-17 Nov. 2008

[4] Otero, A.; Salvador, R.; Mora, J.; de la Torre, E.; Riesgo, T.; Sekanina, L.; "A fast Reconfigurable 2D HW core architecture on FPGAs for evolvable Self-Adaptive Systems," 2011 NASA/ESA Conference on Adaptive Hardware and Systems (AHS).

[5] Salvador, R.; Otero, A.; Mora, J.; de la Torre, E.; Sekanina, L.; Riesgo, T.; "Fault Tolerance Analysis and Self-Healing Strategy of Autonomous, Evolvable Hardware Systems," International Conference on Reconfigurable Computing and FPGAs (ReConFig), 2011

[6] Fuller, E.;Caffrey, M.; Salazar, A.; Carmichael, C.;Fabula, J.;"Radiation characterization,and SEU mitigation, of the Virtex FPGA for space-based reconfigurable computing," Xilinx Application Note, 2000.

[7] Gericota, M.G.; Lemos, L.F.; Alves, G.R.; Ferreira, J.M.; "A Framework for Self-Healing Radiation-Tolerant Implementations on Reconfigurable FPGAs,"IEEEDesign and Diagnostics of Electronic Circuits and Systems, 2007. DDECS '07., vol., no., pp.1-6, 11-13 April 2007

[8] Al-Zawi, M.M.; Al-Jumeily, D.; Hussain, A.; Taleb-Bendiab, A.; "Autonomic Computing: Applications of Self-Healing Systems,"Developments in E-systems Engineering (DeSE), 2011

[9] Parris, M.G.; Sharma, C.A.; DeMara, R.F.; "Progress in Autonomous Fault Recovery of Field Programmable Gate Arrays," ACM Computing Surveys, 2010.

[10] Oreifej, R.; Sharma, C.; DeMara, R.F.; "Expediting GA-Based Evolution Using Group Testing Techniques for Reconfigurable Hardware," Int. Conf. on Reconfigurable Computing and FPGAs (ReConFig 2006), IEEE, 2006, pp. 1-8.

[11] A. Thompson, "Evolving fault tolerant systems" 1st International Conf. on Genetic Algorithms in Engineering Systems: Innovations and Applications (GALESIA), IEE, 1995, pp. 524-529.

[12] J. Lohn, G. Larchev, and R.F. DeMara, "Evolutionary Fault Recovery in a Virtex FPGA Using a Representation That Incorporates Routing" Proc. 17th International Parallel and Distributed Processing Symposium (IPDPS), 2003, p. 172.

[13] Garvie, M.; Thompson, A.; "Scrubbing away transient and Jiggling around the permanent: Long survival of FPGA systems through evolutionary self-repair,"On-Line Testing Symposium, 2004. IOLTS 2004. Proceedings. 10th IEEE International, pp. 155-160, 2004.

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

[15] Otero, A.; de la Torre, E.; Riesgo, T.; Krasteva, Y.E.; "Run-Time Scalable Systolic Coprocessors for Flexible Multimedia SoPCs," 2010 International Conference on Field Programmable Logic and Applications (FPL)