A Fast Reconfigurable 2D HW Core Architecture on FPGAs for Evolvable Self-Adaptive Systems

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

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

Abstract

Modern FPGAs with Dynamic and Partial Reconfiguration (DPR) feature allow the implementation of complex, yet flexible, hardware systems. Combining this flexibility with evolvable hardware techniques, real adaptive systems, able to reconfigure themselves according to environmental changes, can be envisaged.

In this paper, a highly regular and modular architecture combined with a fast reconfiguration mechanism is proposed, allowing the introduction of dynamic and partial reconfiguration in the evolvable hardware loop. Results and use case show that, following this approach, evolvable processing IP Cores can be built, providing intensive data processing capabilities, improving data and delay overheads with respect to previous proposals. Results also show that, in the worst case (maximum mutation rate), average reconfiguration time is 5 times lower than evaluation time.

1. Introduction

Embedded system design has a permanently increasing demand on complexity and flexibility, because of the market pressure and continuous emergence of new applications. Regarding complexity management, the methodological trend is to design systems by means of the integration of different IP cores created and validated independently, to reduce design time, and consequently, time to market. Cores are typically hardware modules designed to speed-up computation intensive tasks. On the other hand, there is a need of flexibility, in the sense of looking for systems with the capability of adapting their behaviour to time-varying conditions. This would help in increasing systems lifetime, and consequently, time in the market. The challenge is to combine hardware performance with the flexibility required by adaptive systems.

Making use of Dynamic and Partial Reconfiguration (DPR) features of modern SRAM-based FPGAs opens new possibilities towards 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 the inside of the device logic, which is a clear advantage in order to maximize integration.

Furthermore, FPGA-based self-reconfigurable systems can benefit from the Evolvable Hardware (EH) approach to adapt themselves when needed. Proposed by Higuchi [1] and De Garis [2] in 1993, EH research has split into various approaches, being online adaptive hardware the final objective of this work, which allows the automatic self-synthesis of new circuits by means of an Evolutionary Algorithm (EA) running in the system. However, integrating DPR with EH techniques suffers from the low speed of the reconfiguration ports. A well-known alternative is the use of Virtual Reconfigurable Circuits (VRC), a reconfigurable layer built on top of the reconfigurable fabric that reduces the complexity of the reconfiguration process, creating a kind of application specific programmable elements [8]. Main problems of VRCs are area and delay overheads, as well as power consumption, that might be acceptable during the adaptation step, but not during the subsequent processing phase. Differently, the main proposal of this work is the integration of a widely known 2D systolic processing architecture with an optimized DPR control engine which allows the implementation of adaptive (evolvable) processing-hardware with native reconfiguration support.

The architecture consists on a highly regular and parallel two dimensional mesh-type array of processing elements (PE) arranged like a systolic structure featuring intensive data processing capabilities in a broad range of fields. Each PE can be configured with a given functionality, which also permits to define which of the input signals are used and how, consequently allowing the adaptation of the data transmission front. Each possible combination of functions and connections is pre-synthesized and stored like an independent module. It is

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

during the adaptation phase that it is dynamically configured in the suitable position of the FPGA according to the EA decisions. This proposal avoids the inclusion of multiplexers as well as the simultaneous implementation of all the possible functions in each element, reducing the area and processing overhead compared with VRCs solutions. Due to the high modularity and regularity of the architecture, PEs which are already configured in the device can be readback from the very same configuration memory of the FPGA and reallocated to the desired place, reducing the external memory accesses. In addition, only those modules that have to be changed in the array will be changed, while those common for the original and the new circuit will be unaffected. Considering typical mutation rates in Cartesian Genetic Programming (CGP), which is the selected EA in this work, the number of PEs to be reconfigured will be extremely low between the evaluations of two candidates. The proposed architecture is generic, and its suitability to different processing tasks depends on the external library, which can be incremented even during system life time.

Together with the architecture, a peripheral module for low level control of the reconfiguration port is proposed to carry out the reconfiguration process. The main advantage of this reconfiguration engine is the relocation capability implemented in hardware, which also includes the readback/reallocation/writeback approach. It has been designed to be times faster than the evaluation phase of the candidate circuit, even beyond the maximum theoretical throughput of the reconfiguration port.

The rest of the paper is organized as follows. In the next section, some introductory details of DPR and EH are shown. Then, the proposed reconfigurable processing system, together with the architecture of the reconfigurable core as well as the reconfiguration engine are described. Section IV features a detailed description of the evolutionary framework of the system. Finally, in section V, implementation details and results are provided, and some conclusions are drawn in section VI.

2. Previous Concepts and Related Work

This section deals with previous concepts and work related with both partial reconfiguration and EH.

2.1. Dynamic and Partial Reconfiguration

To reconfigure an SRAM based FPGA, it is necessary to change the content of its configuration memory. The mechanism to change this content is to send some configuration commands and new configuration information through a reconfiguration port. It is common to have several configuration ports in most devices, however, in this work, self-reconfiguration is required to permit system self-adaptation, and so, the Internal Configuration Access Port (ICAP), a configuration port embedded in the device logic, is used. To control data transference to the configuration memory using the ICAP, Xilinx provides HWICAP [4], a peripheral that can be easily integrated in the system. This logic sends bitstreams directly to the port with no modifications. A configuration bitstream is a file that includes both the sequence of commands to control data transference through the reconfiguration port, and the configuration data itself.

Partially reconfigurable systems very often require to have relocation capability, that is, to configure the same reconfigurable module in different positions of the device, using a single bitstream as data source. For this, it is necessary to change the address included in the configuration file. HWICAP cannot perform this reallocation automatically, so it has to be carried out in software, including extra timing overhead, or hardware, where extra logic has to be added to the configuration controller. With this purpose, bitstream filters have been already provided in some works like [5].

The granularity of the reconfiguration depends on the configuration resources corresponding to a frame, the minimum addressable unit of information of the configuration memory. In the case of Virtex-5 devices, this granularity is one column of configurable logic, which corresponds to the height of one clock region. Thus, reconfiguration of two dimensional architectures is possible, as is exploited in this work.

2.2 Evolvable Hardware

EH usually involves the use of reconfigurable hardware devices whose configuration is controlled by an EA [1]. EH is regarded like a key technology to achieve adaptive systems, that is, hardware systems able to reconfigure themselves to adapt to environmental or requirements changes, or to recover from faults. In this work we focus on the architectural requirements needed to build up such systems relying on DPR, which is the focus of our EH Platform. Consequently, this section will be limited to EH architectures based on FPGAs. Specifically those where the full system is implemented in the FPGA, including the EA, the fitness calculation and the processing architecture itself will be considered.

Under these assumptions, the straight approach would be to directly evolve the FPGA bitstream. However, this is unfeasible due to the huge space to be explored, the risk of jeopardizing the device and some other practical issues like, for instance, the scalability problem. Consequently, other alternatives were examined and, among them, the most extended one is the use of virtual architectures with

inherent reconfigurability defined on top of the device fabric. This approach, proposed in [8] and [9], implies implementing all the possible configurations of the architecture simultaneously on the device, selecting the desired one using a set of multiplexers driven by the configuration registers. The reconfiguration process involves changing these registers, and can be done in just some clock cycles.

The main problem of virtual reconfigurable architectures is the area overhead caused by the inclusion of simultaneous implementations of every single function plus the required multiplexers for routing, reducing the maximum achievable frequency. Therefore, the use of run-time reconfiguration has been studied as an alternative to virtual reconfiguration. For instance, Upegui proposes in [10] two approaches. The first one consists in changing LUT contents to carry out parametric tuning. The second one is based on changing some configurable modules of a neural network, to adapt the processing structure. This approach is mainly constrained by its very specific purpose and the coarse granularity of the modules. Both approaches use the ICAP to perform reconfiguration. In [12], Torresen et al. propose a data classification system. Flexibility is achieved by changing the number of functional units used in each Category Detection Module (CDM) of the classifier, choosing between different pre-synthesized configurations that differ in the number of FUs used in each row and in each column, using DPR. However, only a reduced experimentation has been performed, achieving too slow results to create feasible EH systems. This is due to the large granularity of the reconfigurable regions (to change the number of FUs of the CDM, the full module has to be changed), and a limited use of the ICAP control, without further improvements. A different approach is proposed by the same group in [13] that exploits the SRL behavior of some Virtex devices LUTs to change its logic, without having to use the normal configuration interfaces, like the ICAP. In this case, the reconfiguration process is carried out directly shifting the configuration bits into the LUTs. This alternative achieves a higher reconfiguration speed, enough to fulfill the application requirements, reducing the overhead compared with VRCs. However, the solution is very device dependent, and reconfigurability is limited to adapt LUT functions. Furthermore, only 25% of the Virtex-5 LUTs show this behaviour. More recently, a direct bitstream manipulation approach is applied in a Cartesian Genetic Programming scheme [14], with static routing. In this case, reconfiguration is achieved by means of changing the values stored in the LUTs to change its functionality, as well as the communication channels. A hierarchical strategy is also introduced to deal with the size of the solution space.

3. 2D Reconfigurable Processing Architecture

Figure 1 shows the system level architecture of the proposed System-on-Chip (SoC) consisting on an embedded MicroBlaze processor responsible of controlling the whole system operation, a Reconfigurable processing Core (RC) and a Reconfiguration Engine (RE). All modules in the system have been included as peripherals attached to the Microblaze embedded processor using a commercial PLB bus that allows the configuration of some registers of each of the peripherals from the processor.

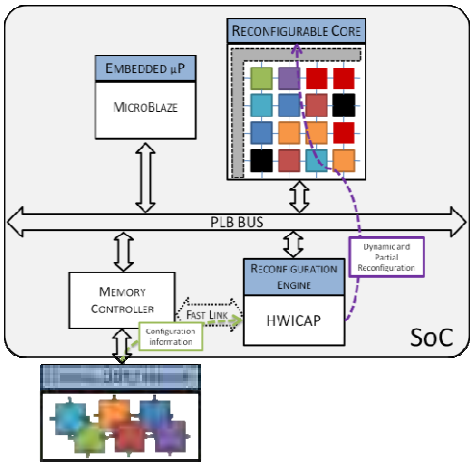

Figure 1. Overview of the System Architecture

The RE needs a memory module for its operation so an external DDR2 memory bank is used for this purpose. Since fast reconfiguration is a must, a dedicated NPI (Native Port Interface) [15] protocol has been used to connect the RE with the memory controller.

As introduced previously, the architecture of the RC is based on a 2D systolic array of PEs, known for its high performance and restrained use of resources. A major feature of our proposal is the possibility to change the functionality of the PEs by means of DPR. This confers the system upon adaptation capabilities. The following Sections offer an in-depth review of each module in the system.

3.1 RC Architecture Description

The proposed architecture is a two dimensional mesh-type systolic array of parallel PEs, focused on intensive data processing applications. Figure 2 shows the structure of the array, sized *AxB* (4x4 in Figure 2). Each PE is a basic computational unit able to perform a single operation on the data taken from their close neighbors, in one clock cycle. All the elements have two inputs, one in the north (N) and another in the west (W), and two outputs, one in

the south (S) and the other in the east (E). Communications among PEs have also been created following the systolic approach, that is, limiting them to the closer neighbors. Consequently, South output of one element is connected with the element immediately below, and the East output is connected to the West input of the element immediately on the right. Regarding data entrance, as shown in Figure 2, a single element has been included in the inputs of each PE in the north and west borders of the array. Each one of these elements can select one of the inputs of the 3×3 (9 pixels) neighborhood of the image pixel to be filtered. To collect processed data and send it back to the memory, just the output of the lowest, rightmost PE is considered. To allow the communication with the static part of the device, bus-macros have been included in the border of the architecture. Bus-macros are fixed-position communication structures that guarantee that the same signals use the same resources of the FPGA, allowing stable connections between reconfigurable and fixed regions on the device.

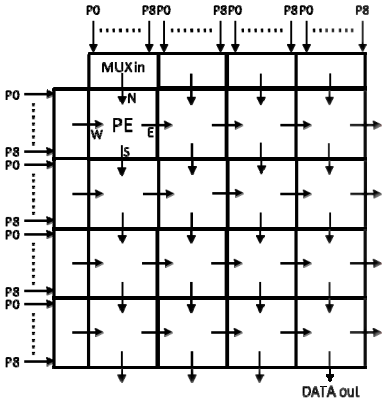

Figure 2 Two dimensional proposed architecture.

The internal structure of the PEs is shown in Figure 3. Each element includes a Functional Block (FB) that performs a basic operation with the input signals, and sends the result through both outputs. Therefore, S and E outputs of each element have always the same result at a given time. The considered FBs, which are shown in Table 1, have been chosen according to the proposals in [3], extending the ideas originally proposed for a similar VRC application. FBs involve one (x) or two (x, y) inputs and all of them have one output (z). The different PEs are generated combining the proposed functions with all the possible assignments between inputs of the element and inputs of the function, despite some exceptions due to redundancies, like for example, *N+W* and *W+N* as shown in the following paragraph. In addition, a register has been also included inside each PE with data pipelining purposes.

Therefore, PE operation could be classified as one of the following expressions $S=E=FB(N,W)z^{-1}$, $S=E=FB(N)z^{-1}$, or $S=E=FB(W)z^{-1}$. If a two-input operator is not commutative, then both *FB(N,W)* and *FB(W,N)* are provided as separate elements. Consequently, the total number of available elements is 16. Following the modular design flow, bus-macros have been instantiated in all the inputs and outputs of each PE.

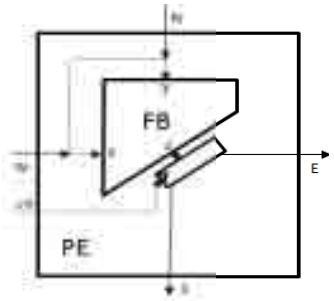

Figure 3 Structure of Processing Elements

In the taxonomy of systolic arrays, architectures can be classified according to the direction of data and results fronts' transmission. The proposed architecture allows all possibilities, since it depends on whether a module uses only the north input, the west one, or both inputs. Adaptation of each element is achieved by directly configuring the required PE in each position of the array, taking it from a library of pre-synthesized PEs. This process can be seen as replacing pieces in a puzzle. For each piece (PE to reconfigure) the RE places the required element as ordered by the processor in the correct position of the matrix (puzzle). To make this reconfigurable approach feasible, it is necessary to identify which logical resources of the FPGA are dedicated to each processing element of the array. As a result, the architecture has been created in a completely modular way, regarding its layout in the FPGA. Modularity also allows tackling the design and synthesis of each PE independently, using the Xilinx modular design flow [16]. The use of a standard commercial flow reduces the dependency with the specific details of the device, mainly compared with fine grain reconfiguration approaches [13]-[14], described in the previous section. In addition to commercial synthesis programs, an ad-hoc tool has also been created to extract the partial bitstream that corresponds to every PE. All modules of the architecture are defined using a rectangular area with the same dimensions. This approach might introduce an extra overhead in case of having PEs with a very different complexity, and consequently, resource consumption. However, this is not the case of the PEs used in this work, as will be justified in the results Section. On the contrary, this regularity allows relocation of the same bitstream to different basic PE positions in the FPGA fabric, reducing the number of necessary bitstreams.

Both modularity and regularity are the main strengths of

the array, allowing a finer reconfiguration grain, compared with existing state-of-the art modular solutions [12]. Consequently, reconfiguration time is reduced. This DPR with elements relocation is carried out using a special hardware block described in the next Section.

The proposed architecture can be easily extended to any other processing purposes, since new processing elements can be added to the library. In addition, elements included in this library can be reused among applications. As it will be explained in the Implementation section, the size of the implemented structure is 4×4, but it can be completely and easily scaled.

Table 1 Considered Functional Blocks

Code	Function	Description
0	x + y	Adder
1	x << 1	Left shift by 1
2	$x +_s y$	Adder with saturation
3	(x + y) >> 1	Average
4	255	Constant
5	x >> 1	Right shift by 1
6	x	Identity
7	max(x,y)	Maximum
8	min(x,y)	Minimum
9	$x -_s y$	Substraction with saturation to 0

3.2. Description of the Reconfiguration Engine

Description of the enhanced ICAP control used in the proposed framework includes the main aspects that enable fast reconfiguration to make online adaptation feasible, together with the regularity and modularity of the architecture featured above.

Bitstreams corresponding to the PEs included in the library of modules to compose the architecture do not include configuration commands. That is, only the body of the bitstream is stored, while both the header and the tail are eliminated. Consequently, final PE position in the device is not predefined. Differently, as shown in [11], header and tail info, as well as frame addresses, are added at run time. This strategy provides two advantages: i) A reduction of the bitstream size that allows reducing the whole data transference time from the external memory, and ii) faster relocation possibilities. Regarding the relocation feasibility, it allows having a unique bitstream for each PE that can be configured in any position of the array. This relocation is much faster than the previous proposals in the State of the Art, since those like [5], are based on bitstream parsers, instead of a runtime composition of the bitstream. This higher speed achievement allows increasing the frequency of the relocation process.

In addition, even though the maximum frequency of the Virtex-5 ICAP reported by the manufacturer is 100 MHz, we have proven the feasibility of overclocking it. In this work, we have experimentally achieved ICAP valid operation at 250MHz, including the online relocation process. Exploiting this, the overhead of the reconfiguration process is further reduced. Also, a direct and dedicated data link has been included from the external DDR2 memory to the reconfiguration module, using the NPI. This connection is able to feed the reconfiguration port with information fast enough to carry out fast reconfiguration, without introducing dead times.

Furthermore, analyzing the reconfiguration requirements of modular and regular architectures, in most cases, the same element has to be reallocated at different positions in the architecture. As a consequence, the possibility of pasting the same configuration module in different positions of the architecture, without having to read it further times from the external memory, has been developed. Moreover, this module might already be configured in other positions of the device. Consequently, also internal memories have been included to allow this readback/reallocation/writeback approach of full modules of the architecture. This approach eliminates the overhead of reading the configuration information from the external memory, greatly reducing the reconfiguration overhead so as to be appealing enough for its inclusion in evolvable systems.

RE includes a software API that simplifies the reconfiguration process. Further information about this API can be found in [11].

4. Evolutionary Framework

The previous Section featured a description of the general system architecture as well as the mechanism proposed for self-reconfiguration of the embedded processing core. This Section will therefore deal with the description of the force driving its adaptation, an Evolutionary Algorithm. Figure 4 shows the architecture of the proposed adaptive system outlining the HW/SW partitioning accomplished.

From the previously described Reconfigurable Architecture to this Self-Adaptive, evolvable, architecture, an EA running on the embedded MicroBlaze processor has been added, together with an Evaluation Module (EM) and a tightly coupled on-chip RAM memory. The objective of such architecture is to provide an adaptation framework so that the system can autonomously evolve a required *hardware processing task* at any given time. This processing task model is given as a couple of datasets, the *INPUTDATA* and the *IDEALRESULT*. The aim of this adaptation framework is therefore letting the system trigger a self-synthesis phase which will eventually evolve a functionally valid circuit.

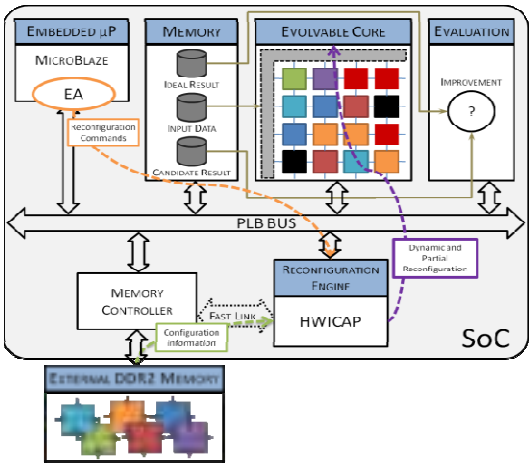

Figure 4 Evolvable FPGA-based SoC Architecture

Main activity of the embedded MicroBlaze processor besides running the EA deals with issuing the reconfiguration commands needed by the RE to configure the given candidate solution. Thanks to DPR, the EA can remain operating while the core is being reconfigured.

Getting inspiration from CGP as a valuable EA to evolve *similar* VRC-based Cartesian architectures, a simple *(1+λ) Evolution Strategy* with 1 parent and λ offspring has been implemented. The representation of the individuals encodes genes within the chromosome as a set of integer numbers representing *input connection genes* and *functionality genes*, which point to a particular PE of the library, as seen below:

$$< InMux_1, \cdots, InMux_{A+B}, PE_0, \cdots PE_{AB} >$$

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 of the chromosome is therefore $(A + B) + A \times B$. Evolutionary operators are defined according to CGP prescriptions. From a random initial population, selection chooses the fittest individual as parent for the next population (elitism enabled). In the case that two individuals score the same best fitness, diversity 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 is applied). For each mutant offspring, the mutation operator modifies k randomly selected genes from the parent. Uniform integer distribution (from 0 to 8 for input genes and from 0 to 15 for functionality genes) is used for this operator.

The evaluation of the candidate solutions comprises the following phases:

P1. ***Reconfiguration*** of the Core (conveniently renamed as Evolvable Core) with the candidate solution dictated by the EA.

P2. ***Execution*** of the processing task *(T)* configured in the Evolvable Core.

P3. ***Fitness*** computation (in the Evaluation Module) of the candidate solution actually configured in the Evolvable Core.

Thanks to the achieved parallelism, both phases, P2 and P3 are computed simultaneously. A conveniently defined Fitness function needs to be implemented in the system so that evolution finds its way to the required goal. This Fitness specification as well as the required training datasets definition have to be accomplished in a per-application basis. However, due to the bus-based structure of the system, where pluggable IP modules are attached as peripherals to the embedded processor, the implementation of different Evaluation Modules is straightforward. Regarding the datasets, these are provided from an external FLASH memory, but any other method can be used which supplies the required data to the system.

5. Implementation Results for an Evolvable Image Processing Core

Proof-of-concept system has been implemented in a Virtex-5 LX110T FPGA (a medium size Virtex-5) contained in XUPV5-LX110T Evaluation Platform. The results reported here correspond to an array sized *AxB=4x4* PEs. Layout of a single PE, including Bus-Macros, the clock signal, global enable and reset signals and the inputs/outputs (N, S, E and W) is shown in Figure 5. Each processing element occupies two CLB columns expanding one clock row, that is, 40 CLBs. Regarding these dimensions, this is a first prototype implementation that for simplicity in the implementation introduces too much area overhead for the computational complexity of the selected operations. According to the implementation results, PEs actually occupy from 7 to 10 slices, depending on the FB. Considering that a Frame includes configuration information of a full column of CLBs (20 CLBs), decision of creating homogenous modules, independently of their behaviour, do not introduce extra reconfiguration penalty.

Overhead due to the communications is also very high, since 6 of these slices are dedicated to bus-macro terminals. Maximum operating frequency of each PE, reported in the synthesis process using ISE 12.1 tool, is over 400 MHZ. However, the maximum frequency of the full array is 250 MHz after the mapping and implementation phases. This increase in the computation delay is due to the inter-connection of the different PEs through the Bus Macros and the interface to the static design area.

Evolution of image noise filters is selected as the proof of concept application. The evolution goal is therefore to

minimize the difference between the filtered image and the original image. Mean Absolute Error (MAE) is selected as fitness function:

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

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

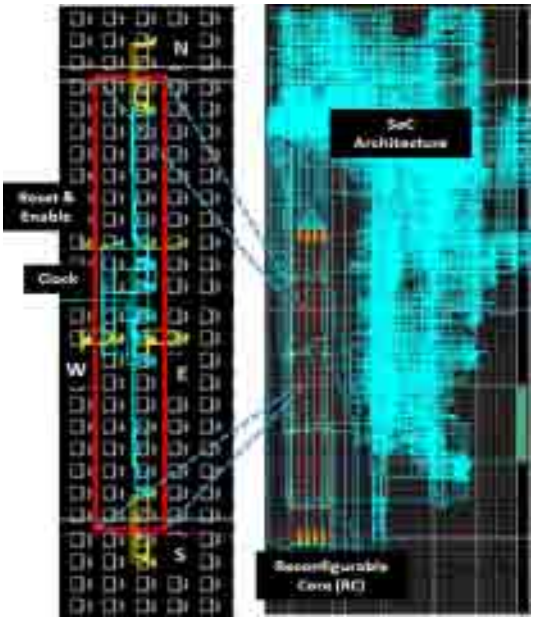

Figure 5 Processing element and a 4 × 4 array integrated in the SoC.

5.1. Time of reconfiguration

The first phase in the evaluation of the candidates, P1, involves DPR. This time depends on the area to be reconfigured and the frequency at which data is sent to the ICAP. Each CLB column requires the change of 36 frames in the configuration memory, that is, 72 frames have to be reconfigured in order to change a basic element. Since each frame is composed by 41 32-bits words, 2952 configuration words are required to describe each element. In the case of Xilinx HWICAP, these frames have to be transmitted using a shared PLB bus. However, introducing the direct NPI connection to the external memory eliminates this communication overhead, while releases the embedded processor to simultaneously execute a different task. Due to the burst support of the controller, bitstream memories included in the reconfiguration engine never run empty, allowing sending data continuously to the ICAP. Furthermore, timing overhead of the reallocation process is limited to the online introduction of some commands in the header of the bitstream. Considering this header and also tail overheads (including a pad frame), each processing element requires *2952 + 60 = 3012* words to be sent for each element. At the maximum reconfiguration speed reported by Xilinx, that is 100 MHz, around 30 µs are necessary to change each PE.

Consequently, for the typical low mutation rates in CGP, between 1 and 5 elements at a maximum have to be reconfigured between two candidates, which correspond to mutation rates under 20% for the considered array size. Thus, average reconfiguration times are between 30 µs and 150 µs. Embedded processor also introduces certain overhead when sending configuration commands to the RE, but that can be disregarded compared with reconfiguration times. As mentioned in Section 3.2 ICAP overclocking has also been explored in this work, achieving a maximum frequency of 250 MHz, which yields 12 µs per PE. Consequently, final reconfiguration time, t_{P1}, drops to a range between 12 µs and 60 µs.

5.2. Time of execution and fitness computation

As for the second and third phases in the evaluation of each candidate circuit, P2 and P3, the time t_{P2+P3} required to filter a 256 × 256 image and computing the resulting fitness, considering the initial array latency negligible in comparison to the image size, is 262 µs at 250 MHz (since P2 and P3 work in parallel as explained in Section 4, only a whole image has to be retrieved from memory to accomplish both phases). As a result, reconfiguration times are, at least, 5 times lower than the execution plus fitness computation time.

5.3. Overall time of evolution

Table 2 gathers information regarding the relation between average t_{P1} and t_{P2+P3} times for different mutation rates. Besides, the number of circuit evaluations per second is also computed and shown in the Table.

A software simulation has been accomplished to validate the proposal. For the selected (1+8)-ES, 100000 generations were simulated, yielding an average of 50000 generations needed to evolve a good circuit. According to these values, about 400,000 circuits are evaluated until a good result is achieved. Considering the previous timing analysis an average of 128s is needed to achieve a correct result.

Table 2 Timing results of the system

Mutation rate	t_{P2+P3}	Average t_{P1}	% Evaluation time*	Evaluations per second
3	262 µs	36 µs	13.95 %	3358
4		48 µs	18,60 %	3226
5		60 µs	23,25 %	3105

Values shown do not include the EA execution time itself, but system implementation is such that a new

candidate is generated from its parent while the previous one is being evaluated, so both computations are overlapped.

6. Conclusions and Future Work

In this work, a system architecture combining 2D data processing arrays and an enhanced DPR engine is proposed, showing the suitability to be integrated making up self-evolvable cores without the overhead of existing VRC-based solutions.

Obtained results show that for a 256×256 image, 3105 evaluations per second are achieved in the worst case, which clearly beats the State of the Art [3], where 775 evaluations per second are reported. This shows how the associated delay overhead due to the VRC is eliminated in this implementation, which operates at a maximum frequency five times higher.

Future work will address an improved implementation to optimize the area occupancy of each PE and the delay introduced by the Bus Macros, further improving the maximum operating frequency, even eliminating them as in [7]. In addition, the number of evaluations needed to obtain a working solution will try to be reduced, so a different EA, such as a Genetic Algorithm will be tested. This improvement would reduce the total time of evolution to tens of seconds.

We conclude stating that these results clearly show how DPR is already a good enough approach to tackle adaptation based on evolutionary techniques in embedded systems.

Acknowledgment

This work was supported by the Spanish Ministry of Science 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] Higuchi, T.; Iwata, M.; Keymeulen, D.; Sakanashi, H.; Murakawa, M.; Kajitani, I.; Takahashi, E.; Toda, K.; Salami, N.; Kajihara, N.; Otsu, N.; "Real-world applications of analog and digital evolvable hardware", IEEE Transactions on Evolutionary Computation, vol.3, no.3, pp.220-235, Sep 1999.

[2] H. De Garis, Evolvable Hardware: Genetic Programming of a Darwin Machine, McGraw-Hill, Artificial Neural Nets and Genetic Algorithms, Springer-Verlag, NY, 1993.

[3] 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

[4] "LogiCORE IP XPS HWICAP", http://www.xilinx.com/support/documentation/ip_documentation/xps_hwicap.pdf

[5] Corbetta, S.; Morandi, M.; Novati, M.; Santambrogio, M.D.; Sciuto, D.; Spoletini, P.; , "Internal and External Bitstream Relocation for Partial Dynamic Reconfiguration," EEE Transactions on Very Large Scale Integration (VLSI) Systems, I, vol.17, no.11, pp.1650-1654, Nov. 2009

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

[7] Otero, A., Yana E. Krasteva, Eduardo de la Torre and Teresa Riesgo. "Run-time Scalable Systolic Coprocessors for Flexible Multimedia SoPCs". Proceedings of the 20th International Conference on Field Programmable Logic and Applications, FPL 2010, Milan, Italy, Sep 2010.

[8] L. Sekanina, "Virtual reconfigurable circuits for real-world applications of evolvable hardware", in the 5th Conference on Evolvable Systems: From Biology to Hardware. LNCS vol. 2606 Berlin, Germany: Springer-Verlag, 2003, pp. 186–197.

[9] K. Glette and J. Torresen, "A flexible on-chip evolution system implemented on a Xilinx Virtex-II Pro device," In ICES 2005. LNCS, vol. 3637, pp. 66–75, Springer, Heidelberg (2005).

[10] A. Upegui. Dynamically Reconfigurable Bioinspired Hardware. PhD thesis, Ecole Polytechniqye Federale de Lausanne (EPFL), 2006. Thesis No .3632.

[11] Otero, A.; Morales-Cas, A.; Portilla, J.; de la Torre, E.; Riesgo, T.; "A Modular Peripheral to Support Self-Reconfiguration in SoCs," Euromicro Conference Digital System Design: Architectures, Methods and Tools (DSD), 2010 13th on , vol., no., pp.88-95

[12] 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

[13] Glette, K., Torresen, J. and Hovin, M., "Intermediate Level FPGA Reconfiguration for an Online EHW Pattern Recognition System". In the Proceedings of the 2009 NASA/ESA Conference on Adaptive Hardware System.

[14] Cancare, F.; Santambrogio, M.D.; Sciuto, D.; , "A direct bitstream manipulation approach for Virtex4-based evolvable systems," Proceedings of 2010 IEEE International Symposium on Circuits and Systems (ISCAS), , vol., no., pp.853-856

[15] "Multi-Port Memory Controller (MPMC)" http://www.xilinx.com/support/documentation/ip_documentation/mpmc.pdf

[16] *"Xilinx Modular Design Flow";* http://www.xilinx.com/itp/xilinx7/books/data/docs/dev/dev0025_7.html