IMPLEMENTATION TECHNIQUES FOR EVOLVABLE HW SYSTEMS: VIRTUAL VS. DYNAMIC RECONFIGURATION

Ruben Salvador, Andres Otero, Javier Mora
Eduardo de la Torre, Teresa Riesgo *
Centro de Electrónica Industrial
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

Adaptive hardware requires some reconfiguration capabilities. FPGAs with native dynamic partial reconfiguration (DPR) support pose a dilemma for system designers: whether to use native DPR or to build a virtual reconfigurable circuit (VRC) on top of the FPGA which allows selecting alternative functions by a multiplexing scheme. This solution allows much faster reconfiguration, but with higher resource overhead. This paper discusses the advantages of both implementations for a 2D image processing matrix. Results show how higher operating frequency is obtained for the matrix using DPR. However, this is compensated in the VRC during evolution due to the comparatively negligible reconfiguration time. Regarding area, the DPR implementation consumes slightly more resources due to the reconfiguration engine, but adds further more capabilities to the system.

1. INTRODUCTION

In the field of evolvable hardware systems, evolutionary algorithms (EA) are combined with reconfigurable devices to either automatically design or adapt hardware. Since all candidate circuits are evaluated in a reconfigurable device, the reconfiguration time and a suitable granularity of reconfiguration are the key factors that determine the overall performance of evolvable systems.

Initial works in evolvable hardware directly evolved the reconfiguration bitstream. However, this is not possible with current devices since random modifications of the bitstream may damage the device. After some initial attempts, like Xilinx JBits, two lines of research seem to be consolidating regarding reconfiguration techniques for evolvable systems in FPGAs. One of them makes use of the dynamic partial reconfiguration (DPR) capabilities of modern FPGAs; the other, known as Virtual Reconfigurable Circuit (VRC) [1], builds a virtual reconfiguration layer on top of the reconfigurable fabric using multiplexers, enabling the designer to create an application-specific reconfigurable platform consisting of application-specific processing elements (PEs).

Regarding DPR, it utilizes the Internal Configuration Access Port (ICAP) of modern Xilinx FPGAs that allows the FPGA to be reconfigured internally. Upegui and Sanchez evolved the LUT contents (in one dimension) while keeping a fixed routing [2]. This concept has been extended to two dimensions exploring thus the capabilities of recent Virtex 4 devices [3]. In [4] a data classification system able to change the number of functional units from a pre-synthesized set is proposed. Probably the most sophisticated ICAP-based (implemented in HW) evolvable platform for the Virtex 5 family has been introduced by the authors of this paper [5]. Its detailed description follows in Section 2. Its main advantage is the bitstream relocation capability into any compatible position of the FPGA.

In the case of VRCs, although reconfiguration is very fast, since it only involves writing a big register (the configuration memory) which controls a set of multiplexers, this approach suffers from a huge area overhead since it involves the implementation of every possible function in every virtual reconfigurable element. Besides, the multiplexers, used to select the desired functionality in each PE, increase circuit delay. This approach has been utilized by several research groups to solve different tasks [6], [7], [8].

Because of the radically different approach, various pros and cons can be observed in each method. So, what this paper tries to address is a comparison between both for a given reconfigurable processing architecture. In previous works, the authors proposed an FPGA-based self-adaptive platform based on a library of simple, dynamically reconfigurable, processing elements (PE) and an enhanced HWICAP reconfiguration engine [9], [5]. These PEs, which are arranged in a 2D processing array, are defined by their partial reconfiguration bitstream, which requires a somehow complex and very specific design process to be created. Besides, resource usage increases due to the extra

*. *Supported by Spanish Ministry of Science and Research, project DR.SIMON (TEC2008-06486-C02-01)*

†. *Supported by the IT4Innovations Centre of Excellence project CZ.1.05/1.1.00/02.0070*

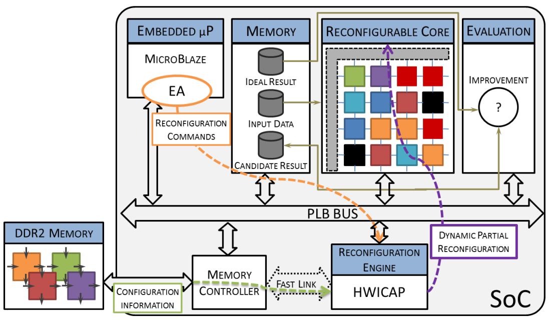

Fig. 1. System architecture holding the evolvable platform.

logic needed to perform DPR. For this work a VRC-based implementation of the processing array using VHDL has been accomplished so that a fair comparison in terms of performance and implementation area between both could be obtained for modern FPGAs. The reference application is image filtering since an EA is expected to efficiently deal with uncertainty coming from the unknown types of noise that might appear at the input signal.

The following section gives a brief overview of the system architecture before describing its computational core, an evolvable processing array, using both reconfiguration options, DPR and VRC. Finally, the results for both implementations are presented in section 3 in order to provide data for discussion. Paper is concluded in section 4.

2. EVOLVABLE PLATFORM DESCRIPTION

The proposed self-adaptive, evolvable, platform, built into a Virtex-5 FPGA device, is defined by a set of components (IP blocks) connected through a common bus interface. An adaptation engine acts upon the measured component performance trying to fulfil the adaptation requirements within a changing operating environment. An EA based on Cartesian genetic programming (CGP) was chosen for the task. Besides, since some kind of reconfiguration capability is required, two versions of the component's processing matrix featuring the two reconfiguration options mentioned previously have been implemented and evaluated.

Fig. 1 shows the proposed architecture, deeply analysed in [9], [5]. An embedded MicroBlaze microprocessor runs the system software that includes the control of the adaptive component, both at run and adaptation time. Hence, this microprocessor, among various other different tasks, runs the EA which proposes new candidate solutions of the component and issues the required commands to evaluate the population of candidate solutions. This process involves the reconfiguration of the array with these candidate solutions. A peripheral for HW fitness evaluation can also be observed, as well as a tightly coupled RAM memory, both of which serve in accelerating this task.

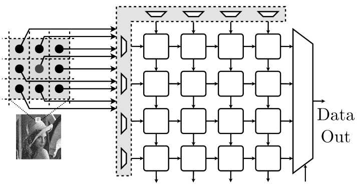

Fig. 2. Reconfigurable architecture of the processing array.

The processing architecture of the evolvable component is a highly regular and parallel two dimensional mesh-type array of $A \times B$ processing elements (PEs) organized as a systolic structure where inter-node connectivity is fixed and restricted to the four closest neighbours (North, South, East, West), as shown in Fig. 2. The input to the array is the same as in typical image convolution filters, a moving square window, sized 3×3. However, there is not a predefined routing from the window to the input PEs. By the contrary, each input PE has an associated 9-to-1 multiplexer so that circuit inputs may be chosen by the EA. The output of the array is obtained from any of the eastern (right-side) PEs, by using a 4 inputs multiplexer controlled by evolution. Each PE consists of a functional block (FB), some routing logic in the input and a register (R) in the output, which is the same for both East and South ports. The FB of each PE can be dynamically configured to perform one of the 16 functions shown in [9], such as *maximum, minimum, adding*, etc. operating over 8 bits.

The architecture proposed is a generic evolvable processing framework, and its suitability for different processing tasks depends on the chosen library. Adaptation is achieved by directly configuring the required PE in each position of the array. This can be seen as placing pieces in a puzzle. For each piece, or PE to (re-)configure, a reconfiguration engine (RE) places the required element as commanded by the processor in the assigned position of the matrix. This RE will act as the interface hiding the reconfiguration details as much as possible to the designer; therefore, the version of the component using DPR shall contain a Xilinx HWICAP which is not needed in the VRC.

The chromosome which encodes the candidate solutions, and therefore serves as the interface with the RE, is composed of a set of integer numbers representing *connection genes* for the input and output multiplexers and *functionality genes* as index pointing to the library for each PE. Based on preliminary simulations (see [9]), an array size of 4×4 PEs was selected, which yields a chromosome of 25 integer genes (1 output MUX, 8 input MUXs and 16 PEs).

2.1. Reconfigurable Array (RA)

The component based on DPR uses an enhanced HWICAP, which, unlike the standard Xilinx module, allows read-back and re-allocation, reducing this way external memory

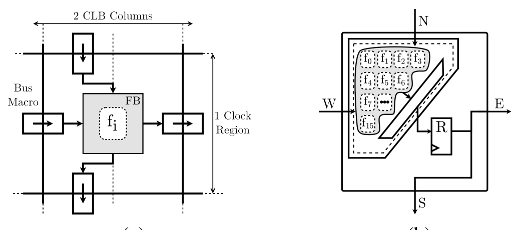

Fig. 3. (a) shows a PE in the DPR case (register not shown for simplicity) and (b) its abstraction in the VRC

accesses when moving/copying one PE from one position to another within the array. Besides, the ICAP was overclocked at up to 200 MHz and attached to an external DDR2 memory through a fast Xilinx NPI to accelerate the memory access process. Each FB is pre-synthesized and stored as an independent module in the library of reconfigurable PEs defined by their partial bitstreams. Unlike VRCs, fixed connections and a single function are implemented in each PE. Fig. 3a shows a PE from the reconfiguration point of view. There is one Bus Macro per port of the PE (N,S,E,W) so that all PEs share a common connection interface.

2.2. Virtual Reconfigurable Circuit

To mimic the behaviour of the reconfigurable library in the RA version, a kind of virtual reconfiguration library is synthesized locally into each PE by using a simple VDHL `case` statement containing the 16 functions mentioned previously, as shown in Fig. 3b. A direct consequence of this decision is the elimination of the input multiplexer at each FB (to select either N or W) in typical VRC implementations. The VRC is configured through the PLB bus, which involves the transfer of 98 bits (obtained in 4 simple 32-bit transfers).

3. IMPLEMENTATION RESULTS

The evolvable platform was implemented in a Virtex-5 LX110T FPGA included in Digilent's XUPV5 Evaluation Platform. Results are given for a 4×4 array. Each PE within the RA implementation occupies two CLB columns along one clock row, that is, 40 CLBs. According to individual synthesis results per PE (needed to extract the partial bitstream), these occupy from 7 to 10 slices. Overhead due to the communication is also very high, since 6 of these slices are dedicated to bus-macro terminals. Regarding the VRC implementation, a highly functional description was used, which yielded moderate synthesis results. The VRC was built aside the prototype RA implementation so some rough edges still need to be polished to obtain good synthesis results (mainly in terms of circuit delay, as a result of plugging the shared components together).

Table 1 shows the implementation results obtained for each version of the component. The module *Component* refers to the whole evolvable component as the higher entity to be considered in the comparison. No results are thus considered for the MicroBlaze implementation and associated resources needed for the whole design. In Table1, *Array* corresponds just with the evolvable 2D architecture of PEs, while *Misc* comprises all the associated logic and memory resources needed to control the array and feed its inputs. In this case, the resource usage reported for the *Array* entry in the RA version is based on the FPGA region that needs to be declared as a reconfigurable region so that the synthesizer does not use it for other purposes. Therefore, the figures do not indicate how many resources are actually needed to hold each PE implementation but those contained within that region. Finally, the enhanced HWICAP and the external memory controller resource usage are shown, which might be reused for other purposes in the system.

3.1. Timing Analysis

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

$$t_g \equiv \lambda \times (t_{offs} + t_{rec} + t_{filt} + t_{fit}) + t_{sel} \qquad (1)$$

where λ is the population size. Therefore, if N_g is the number of generations, the total time needed for evolution can be expressed as $t_{evo} \equiv t_g \times N_g$. Eq. (1) can be simplified since the execution of some tasks is overlapped; T_{offs} is done in SW and T_{eval} in HW, which takes longer for this application. Besides, T_{fit} and T_{sel} are also done in HW in parallel with T_{filt}. Therefore, $t_{evo} \approx \lambda \times N_g\,(t_{rec} + t_{filt}) \equiv N_{EVALS} \times t_{eval}$, where N_{EVALS} is the total number of evaluations. Due to the inherent pipeline of the matrix the array produces one pixel per clock cycle, so filtering time of one candidate circuit takes $R \times C$ clock cycles, where R and C are the number of rows and columns of the image.

Significant performance dissimilarities are expected due to the big differences between reconfiguration and filtering

Table 1. Usage results for the evolvable processing architecture.

Version	Module	Slices	Slice Regs	Slice LUTs	LUTRAM	DSP	BRAM
RA	**Component**	5763	12931	11276	1518	0	38
	Array	1280	5120	5120	1280	0	0
	Misc	1101	1932	1932	64	0	12
	HWICAP	1615	2765	2344	145	0	9
	Mem. Ctrl	1767	3114	1880	29	0	17
VRC	**Component**	2791	4224	5472	128	0	12
	Array	1096	215	2539	0	0	0
	Misc.	1567	3676	2300	128	0	12

Table 2. Reconfiguration, filtering and evolution times

	Time (μs)	
Task	RA@200MHz	VRC@100MHz
Reconfig. (1 PE)	15.92	0.4
Reconfig. (3 PE)	69.61[a]	
Filtering	82.12	163.84
Total evolution	122	132

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

times of each version. Table 2 shows a timing comparison along an evolutionary cycle of 100000 generations using a 128×128 size image and a mutation rate of 3. Maximum frequency for each version is also reported in Table 2. Regarding filtering time, it differs as a result of the extra delay of the VRC due to the multiplexers. However, RA reconfiguration will take longer, and will depend on the number of changed PEs (mutation rate in the EA).

3.2. Discussion on the results

As seen in Table 1 the RA uses slightly more than twice the resources of the VRC. More than half of it is due to the RE (HWICAP and Multi-Port Memory Controller, MPMC) needed to access the DDR2 memory containing the PE library. Therefore, the extra resources *wasted* in the VRC implementation are compensated by those needed to use DPR, so no real difference is observed here. However, if the bitstream size per PE is reduced (which is being tackled now by reducing the reserved logic for the RA), the need for an external memory could disappear, saving the resources consumed by the MPMC. Also, higher array size is in favour of RA, since variable size-dependent resource usage is smaller.

Regarding timing performance, reconfiguration time is fixed for different mutation rates in the VRC, since it just involves writing the configuration register. However, this time depends on the number of changed PEs for the RA, yielding a higher timing overhead in reconfiguration. Since filtering time is double in the VRC, the whole timing performance is very similar under these circumstances. The operating frequency of the final circuit is significantly higher in the RA case, yielding a higher throughput. In any case, a VRC would be able to achieve a frame rate of up to 48.23 images per second for full-HD resolution.

4. CONCLUSIONS

A comparison has been carried out between an evolvable architecture using both native DPR and a VRC. For modern 6 input LUT devices, the area overhead of VRCs is not as high as expected. In terms of maximum working frequency, the VRC is in clear disadvantage, but it is still able to hold full-HD throughput. Nevertheless, deeper and further comparisons, including power consumption, are still needed.

5. REFERENCES

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

[2] A. Upegui and E. Sanchez, “Evolving Hardware with Self-Reconfigurable Connectivity in Xilinx FPGAs,” in *1st NASA/ESA Conf. on Adaptive Hardware and Systems (AHS–2006)*. Los Alamitos, CA, USA: IEEE Computer Society, 2006, pp. 153–160.

[3] F. Cancare, M. Santambrogio, and D. Sciuto, “A direct bit-stream manipulation approach for Virtex4-based evolvable systems ,” in *Proc. of 2010 IEEE Int. Symp. on Circuits and Systems*. IEEE, 2010, pp. 853–856.

[4] J. Torresen, G. Senland, and K. Glette, “Partial Reconfiguration Applied in an On-line Evolvable Pattern Recognition System,” in *NORCHIP, 2008.*, nov. 2008, pp. 61 –64.

[5] A. Otero, R. Salvador, J. Mora, E. de la Torre, T. Riesgo, and L. Sekanina, “A fast Reconfigurable 2D HW core architecture on FPGAs for evolvable Self-Adaptive Systems,” in *2011 NASA/ESA Conf. on Adaptive Hardware and Systems (AHS)*, june 2011, pp. 336 –343.

[6] K. Glette, J. Torresen, and M. Yasunaga, “An Online EHW Pattern Recognition System Applied to Sonar Spectrum Classification,” in *Evolvable Systems: From Biology to Hardware*, ser. LNCS, vol. 4684. Springer Verlag, 2007, pp. 1–12.

[7] L. C. Wang J., Chen Q.S., “Design and implementation of a virtual reconfigurable architecture for different applications of intrinsic evolvable hardware,” *IET computers and digital techniques*, vol. 2, no. 5, pp. 386–400, 2008.

[8] Z. Vasicek and L. Sekanina, “Hardware Accelerator of Cartesian Genetic Programming with Multiple Fitness Units,” *Computing and Informatics*, vol. 29, no. 6, pp. 1359–1371, 2010.

[9] R. Salvador, A. Otero, J. Mora, E. de la Torre, T. Riesgo, and L. Sekanina, “Evolvable 2D computing matrix model for intrinsic evolution in commercial FPGAs with native reconfiguration support,” in *2011 NASA/ESA Conf. on Adaptive Hardware and Systems (AHS)*, june 2011, pp. 184 –191.