IOSR Journal of Electronics and Communication Engineering (IOSR-JECE)
e-ISSN: 2278-2834, p-ISSN: 2278-8735
PP 01-08
www.iosrjournals.org

Review for Adaptive Image processing using Reconfigurable Evolvable Hardware

Kaushik A.R.[1], Satale K. B.[2] & Deokar P. S.[3]
[1,2](E&TC Engg. Dept., KVNNIEER Nasik, SPP Univ., Pune(MS), India)
[3](E&TC Engg. Dept, MCEERC Nasik, Nashik, SPP Univ, Pune(MS), India)

Abstract : *Image recognition systems requiring a low recognition latency or high through-put could benefit from a hardware implementation. Furthermore, if the systems are applied in time-varying environments, evolvable hardware (EHW) would seem to be a promising approach. 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 generated by an embedded evolutionary algorithm and implemented by means of dynamic partial reconfiguration, enabling evaluation in the final hardware. The reconfiguration time and a suitable granularity of reconfiguration are the key factors that determine the overall performance of evolvable systems. Evolvable hardware system can be implemented on an array of processing elements (PEs). These arrays follow a systolic approach, and PEs does not contain extra logic thus performance speed is high. This Paper presents application of EHW in digital image filtering and edge detection. The evolved filters yield better quality than classic linear and nonlinear filters using mean absolute error as standard comparison metric*

Keywords - *Evolvable Hardware, Image Recognition, Latency, Processing Element, Reconfiguration*

1. INTRODUCTION

An electronic hardware can be seen as a set of basic electronic components interconnected in a certain way. Modifying such a circuit implies replacing some components or modifying some of its connections. Performing these modifications would be unfeasible without the concept of re-configurability. An electronic device is said to be configurable (or programmable) when its functionality is not pre-defined at fabrication-time, but can be further specified by a configuration bit stream. Reconfigurable devices permit configuration several times, supporting system upgrades and refinements in a relatively small time scale. In 1984 Xilinx launched to the market the first FPGA (Field Programmable Gate Array). The main advantage of FPGAs is the FPGAs' architecture scalability. FPGAs are programmable logic devices that permit the implementation of digital systems. FPGAs are typically composed of an array of uniform logic cells, interconnected through programmable interconnection matrices that can be configured to perform a given function by means of a configuration bit stream [1].

FPGAs programming is performed by means of a configuration bit stream, which is stored in a configuration memory. When the circuit has been completely placed and routed, a configuration bit stream is generated, which describes the functionality of every programmable element in the FPGA. Before programming the FPGA, it is non-operational (waiting for a bit stream), and the configuration memory is empty. This is said to be a static configuration given that for reconfiguring the device with a new bit stream, the configuration memory must be erased and the configuration process must be restarted. Some FPGAs allow the performing of partial reconfiguration where a reduced bit stream reconfigures only a given subset of internal components. Several issues arise when partially reconfiguring an FPGA, the routing being one of the main problems. When a portion of the FPGA is modified one must ensure that the modified section will still be compatible with the unmodified part. For instance, a processor with a reconfigurable coprocessor must ensure that when modifying the coprocessor logic and its respective routing resources, the new coprocessor will still be correctly attached to the original coprocessor interface [2].

Dynamic partial reconfiguration (DPR) is done while the device is active. Certain areas of the device can be reconfigured while other areas remain operational and unaffected by the reprogramming. Unlike a static partial reconfiguration, in DPR the system execution is not interrupted during the reconfiguration. However, the reconfiguring section will be unusable during the reconfiguration time. Self-reconfigurable systems result as a direct offspring of DPR systems [3].

Self re-configurable systems can modify their own hardware substrate, in order to provide high performance, high flexibility, and high autonomy. They exhibit high performance due to their hardware nature.

Their high flexibility is achieved thanks to the possibility of modifying their hardware at run-time by using DPR. The autonomy is assured by the capability of the platform to modify itself without any external intervention. Self-reconfigurable systems have been mainly used for reconfigurable coprocessors, where a repository of configurations is previously available, and an embedded processor reconfigures the platform with the coprocessor required at a given moment.

2. EXISTING SYSTEM

Existing systems includes brief study of VRC architecture and platform used for VRC, advantages and disadvantages of VRC in first part. It also includes the idea of genotype and phenotype mapping with the EA. Further it explains Dynamically Partial Reconfiguration using FPGA. Architecture of self-adaptive system with the reconfiguration core and reconfiguration element is explained briefly. Implementation uses IP core in FPGA platform, evolutionary algorithm used for self-adaptive system is reviewed.

2.1 Virtual Reconfiguration Architecture

The VRA (Virtual Reconfiguration Architecture), which is described in an HDL, is a second reconfiguration layer developed on the top of an FPGA. The key goals of our proposed VRA are to provide a much simpler intrinsic EHW platform – thus reducing the length of the genotype description and providing the feature of fast internal reconfiguration. Fig. 1 shows that our VRA consists of FE (Function Element) array and an EA. The top function of the FE array is configured using the chromosomes generated by EA, which is implemented on the same FPGA. The chromosome encodes the functions performed by each FE and the interconnection of the FE array. As the fitness calculation is also carried out in the same FPGA, we can benefit from pipeline processing allowing reasonable time of a candidate circuit evaluation. From this perspective, the approach utilizing a VRA offers many benefits, including: (1) In the VRA, the FE array is directly connected to the hardware implementation of the EA placed on the same FPGA allowing the bottle neck introduced by slow communication between FPGA and PC can be overcome.

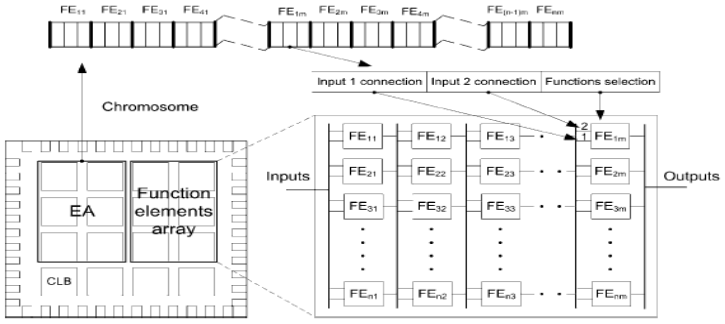

Fig 1 VRA and its Genotype–Phenotype Mapping

(2) As the VRA is available at the level of HDL source code, it can easily be modified and synthesized for various target platforms.

(3) The VRA can be designed exactly according to the requirements of a given problem. This feature means that the granularity of the FE array can exactly fit the needs of a given application [4].

2.1.1Genotype–Phenotype Mapping

Here, the encoding of a digital circuit into a genotype is a development of earlier models. As shown in Fig. 1, the phenotype is a connected two-input FE network in the two-dimensional FE array. This FE array consists of m columns from system input to output in which any FE may be either connected or is connected. Each FE in column l can receive its two inputs from any FE in column l 2 1. Each FE can have one of the functions predefined in a function set (such as AND, OR,XOR, addition, subtraction etc.).The genotype is a fixed-length linear binary bit string, which defines the interconnections of FEs and functions performed by FEs. The mapping process of the genotype to the phenotype is also illustrated in Fig. 1. The chromosome string encodes the FE array by using triplets in which the first two blanks refer to each of the FE's inputs employed, and the third is the achieved function from the available functions set. Compared with CGP employed by

Sekanina [6, 7] and Zhang et al. [5] in their virtual-reconfiguration-based intrinsic EHW in which an FE's input can be connected to the output of some FE's in its preceding columns or to some of the system inputs, our proposed two-dimensional geometric structure restricts the inputs of each FE to come only from the FEs in its previous one column to achieve the purpose of reducing genotype length. Similar to CGP, an obvious advantage of the proposed two-dimensional geometric-structure-based representation is that the genotype representation used is in dependent of the data type used in the phenotype. This feature brings us a generic and flexible platform. As for different applications, one would just need to change the data type, leaving the genotype unchanged. For instance, by changing the function set and the size of data paths in the FE array, our proposed model may be suitable for both gate level and function level evolutions with a comparable chromosome size.

2.1.2 Evolutionary Algorithm

The EA used to produce all of the applications is a simple form based on the 1 þl evolutionary strategy. The EA is only based on selection and mutation operators (MOs). No crossover is applied because several previous experiments [5, 11] have indicated that the crossover operators do not significantly improve the quality of the search. The flow diagram of the employed EA is as follows:

1. Randomly initializes a population of l individuals.
2. Evaluates all individuals.
3. Find the best individual.
4. Create l mutants of the best individual.
5. Create a new 1 þl population using l mutants and the best individual.

The algorithm repeats Steps 2–5 until a termination condition is achieved. The termination condition of EA in this paper is defined as:

(1) The predefined EA generation number is exhausted;
Or
(2) EA finds the expected solution.

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 [8].

2.2 Dynamic Partial Reconfiguration on FPGAs

FPGAs are programmable logic devices that permit the implementation of digital systems. They provide an array of logic cells that can be configured to perform a given function by means of a configuration bit stream. This bit stream is generated by a software tool, and usually contains the configuration information for all the internal components. Some FPGAs allow performing partial reconfiguration (PR), where a reduced bit stream reconfigures only a given subset of internal components. Dynamic Partial Reconfiguration (DPR) is done while the device is active: certain areas of the device can be reconfigured while other areas remain operational and unaffected by the reprogramming [8]. For the Xilinx's FPGA families Virtex, Virtex-E, Virtex-II, Spartan-II and Spartan-IIE there are three documented styles to perform DPR: small bits manipulation (SBM), multi-column PR, independent designs (ID) and multi-column PR, communication between designs (CBD).

Under the SBM style the designer manually edit low level changes. Using the FPGA Editor the designer can change the configuration of several kinds of components such as: look-up-table equations, internal RAM contents, I/O standards, multiplexers, flip-flop initialization and reset values. After editing the changes, a bit stream can be generated, containing only the differences between the before and the after designs. For complex designs, SBM results inaccurate due to the low-level edition and the lack of automation in the generation of the bit streams. ID and CBD allow the designer to split the whole system in modules. For each module, the designer must generate the configuration bit stream starting from an HDL description and going through the synthesis, mapping, placement, and routing procedures, independently of other modules. Placement and timing constraints are set separately for each module and for the whole system. Some of these modules may be reconfigurable and others fixed

2.3 Self Adaptive System

Self-reconfigurable systems result as a direct offspring of DPR systems. Self reconfigurable systems can modify their own hardware substrate, in order to provide high performance, high flexibility, and high autonomy. They exhibit high performance due to their hardware nature. Their high flexibility is achieved thanks to the possibility of modifying their hardware at run-time by using DPR. The autonomy is assured by the capability of the platform to modify itself without any external intervention. Self-reconfigurable systems have been mainly used for reconfigurable coprocessors, where a repository of configurations is previously available, and an embedded processor reconfigures the platform with the coprocessor required at a given moment [3].

The system consists of three main parts – a classification module, an evaluation module, and a processor. The complete system is implemented in a single FPGA. The processor is running the evolution algorithm and configures the other modules. The evaluation module is used for fitness computation and is based on evaluating only a small part of the classifier at a time since incremental evolution is applied. The classification module as shown in fig 2, however, would have to be complete unless time multiplexing is applied. Therefore this module with its typically large structure is difficult to make online adaptable without a large logic gate overhead.

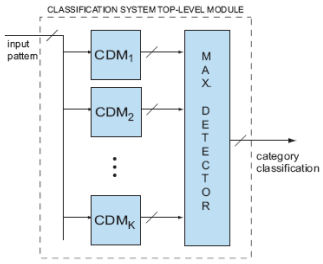

Fig.2: Evolvable Hardware Classification Module.

In [9] and [10] the authors have proposed an alternative to the VRC implementation, which uses native dynamic partial reconfiguration (DPR) of SRAM-based FPGAs to evolve a 2D fine-grain array of PEs. Each function to be mapped on the PEs is defined with a partial bit stream, thus avoiding VRC's drawbacks. The architecture has been optimized to reduce reconfiguration time providing an autonomous, self-reconfigurable, evolvable embedded system. Fig. 3 shows a block diagram of the architecture. The library of partial bit streams feeds an enhanced HWICAP (Xilinx Hardware Internal Configuration Access Port) module featuring

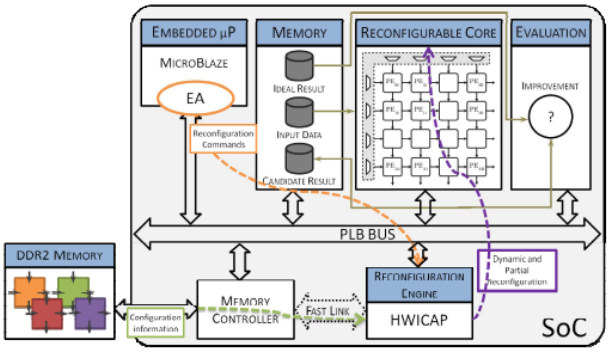

Fig.3 Overview of Self- Adaptive Platform

Blocks relocation capabilities and over-clocked at 250MHz. It is used to reconfigthe array according to the candidate solutions, as encoded in the chromosomes. EA operators have been defined to minimize the number of reconfigurations between evaluations.

The platform, which is based on FPGA SoC architecture, contains a set of IP cores, some of which are adaptive, connected through a common bus interface. An adaptive IP core is a standard IP core with the capability of adjusting its internal processing features as commanded by an adaptation engine. This adaptation engine acts upon the measured component performance trying to fulfill the requirements within a changing operating environment. In this proposal, the adaptation engine is an evolutionary algorithm. The main components are a reconfigurable core (RC) and a reconfiguration engine (RE). RC is the processing array, which is (re)configured by the RE during the adaptation process exploiting DPR. The combination of self-

reconfiguration by using the internal ICAP configuration port and an EA provides the system with the required adaptation capabilities. An embedded Micro Blaze processor issues the required reconfiguration commands to configthe RC to the candidate solutions proposed by the EA as shown in Fig. 3. A peripheral for HW fitness evaluation can also be observed, as well as a tightly coupled RAM memory, which also serves in the acceleration of the individuals' evaluation. Regarding the RE as well as the tools and methodology followed in this work, it can be said that a lesser device dependent solution is proposed. Whether it is true that this solution does depend on a particular device, it is also true that this proposal diminish this dependence. Basically, the tool used to build each PE bit stream automatically carries out, starting from the conventional net slits generated by vendor tools, all the necessary transformations to be consumed by the evolvable hardware platform. In case the target device changes, new partial bit streams can be automatically generated by just relaunching the tool. The RC architecture is a highly regular and parallel two dimensional mesh-type array of A_B PEs organized as a systolic structure where inter node connectivity is fixed and restricted to the four closest neighbors (North, South, East, West), as shown in Fig. 4. However, there is not a predefined routing from the window to the input PEs. By the contrary, each input PE has an associated 9-to-1 multiplexer so that circuit inputs may also be evolved.

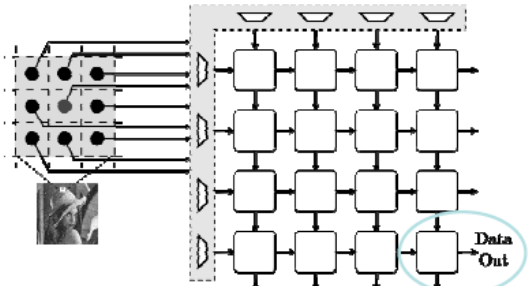

Fig 4:Parallel 2 Dimensional Array of Processing Elements

The output of the array is obtained from any of the east (right-side) PEs, by using a four inputs multiplexer controlled by evolution. Internally, each PE can be dynamically configured to have different functionality and input mapping. Therefore, although inter node connections are fixed, certain flexibility in the adaptation of the data transmission flow is allowed. Fig. 5 shows a conceptual view of a PE, which consists of a functional block (FB) and the associated routing in the input and a register (R) in the output. Each combination of functions and connections is pre synthesized and stored as an independent module in the PE library [7]. Unlike VRCs, fixed connections and a single function are implemented in each PE at a time, eliminating he area and timing penalties.

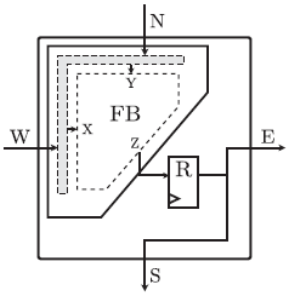

Fig 5: Conceptual Point of View of PE

Ability for different processing tasks depends on the chosen library. Adaptation is achieved by directly configuring the required PE in each position of the array, taking it from the library of pre-synthesized PEs. This process can be seen as placing pieces in a puzzle. For each piece (PE to reconfigure), the RE places the required element as commanded by the processor in the correct position of the matrix. Fig. 6 shows a PE from the reconfiguration point of view.

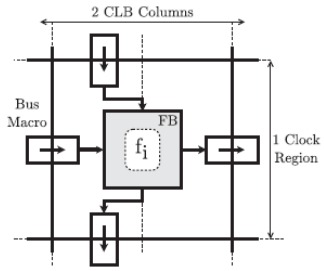

Fig 6: Recognition point of view of PE

Reconfiguration time is kept low because low mutation rates in the EA produce few PEs to reconfigure, as suggested in CGP, which is the reference EA selected for this work. Also, unlike the standard Xilinx module, this improved RE implemented in HW allows for read-back and reallocation to be performed, reducing this way external memory accesses when moving/copying one PE from one position to another within the array.

Table 1: Set of Components in Library

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

Besides, the ICAP was over clocked at up to 200 MHz and attached to an externalDDR2 memory through a fast Xilinx NPI to accelerate the process.

Each FB according to Table 1 is pre-synthesized and stored as an independent module in a library of reconfigurable processing elements defined by their partial bit streams, which is loaded during system start-up procedure in theDDR2 memory. There is one bus macro for each port of the PE (N,S,E,W), which works as the anchoring point in the pieces of the puzzle. This way, one PE can be safely replaced by another one since all of them share common connection interfaces.

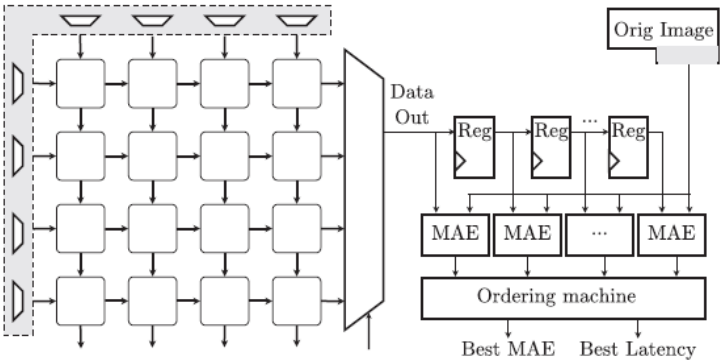

Fig 7: Modified Architecture for Self-Tuning Timing

The studied architecture has a fixed set of PEs and inter node connectivity, which could lead to define a static circuit latency. However, when a PE is reconfigured, the internal data-path is modified depending on which inputs that particular PE uses (as Table 2 shows). This means a change in the internal routing occurs. Besides, the output multiplexer controlled by the EA makes it impossible to forecast the selected output PE (note this behavior is intended to provide a better adaptability). Therefore, there is no analytical way to determine which the correct circuit latency is for this reason; a new architecture is proposed to take into account that a dynamically changing (unknown) circuit needs a dynamically changing latency, which is adjusted online as the circuit evolves. A first approximation could be to include the latency in the chromosome so that it might evolve along with the circuit structure. However, this would increase the search space. Since changing the circuit latency means sampling the output at a different moment, it is easy and not expensive (from the resources point of view) to sample at different moments, replicating the fitness computation units. This situation is shown in Fig. 7.

The new architecture contains several MAE computation modules and a best candidate selector that keeps track of the best associated latency. The input to each fitness unit (stream of filtered pixels) is sampled in a different clock cycle by the insertion of a simple shift register.

3 APPLICATIONS OF EHW

In contrast to conventional hardware where the structure is irreversibly fixed in the design process, evolvable hardware (EHW) is designed to adapt to changes in the environment, through its ability to reconfigits own hardware structure dynamically and autonomously.

- Real-World applications
  1. An analog EHW chip for cellular phones
  2. A clock-timing architecture for Giga hertz systems
  3. A neural network EHW chip capable of autonomous reconfiguration
  4. A data compression EHW chip for electro photographic printers,
  5. A gate-level EHW chip for use in prosthetic hands
  6. Robot navigation.
- General Applications
  1. Digital Image Processing for better image quality
  2. Image filtering: better noise adaption for different types of noise.
  3.

4. CONCLUSION

We can say that the concept of VRC has not been yet outperformed by DPR. DPR beats the VRC in the normal operation mode only.EHW fits the recently introduced concept of under designed and opportunistic computing. The area overhead of VRCs is not as high as expected, although still using DPR is advantageous in terms of area. Also study says, mapping evolvable hardware architectures that can evolve autonomously straight on the reconfigurable fabric of FPGAs, using DPR, is possible. Furthermore, evolution may adapt filter quality to varying noise intensities and types of noise. As compared with traditional filters, it produces much better

results for a wide range of noise densities and it can evolve filters for different tasks such as edge detection, including images corrupted with noise.

REFERENCES

[1] F. Cancare, M. Santambrogio, and D. Sciuto, "A Direct Bit stream Manipulation Approach for Virtex4-Based Evolvable Systems, "Proc. IEEE Int'l Symp. Circuits and Systems (ISCAS), pp. 853-856,2010.
[2] J. Torresen, G.A. Senland, and K. Glette, "Partial Reconfiguration Applied in an On-line Evolvable Pattern Recognition System," Proc. NORCHIP, pp. 61-64, 2008.
[3] A. Upegui, "Dynamically Reconfigurable Bio-inspired Hardware," PhD thesis, EPFL Lausanne, Thesis no. 3632, 2006.
[4] F. Cancare, M. Santambrogio, and D. Sciuto, "A Direct BitstreamManipulation Approach for Virtex4-Based Evolvable Systems,"Proc. IEEE Int'l Symp. Circuits and Systems (ISCAS), pp. 853-856,2010.
[5] K. Glette, "Design and Implementation of Scalable OnlineEvolvable Hardware Pattern Recognition Systems," PhD thesis,Univ. of Oslo, 2008.
[6] T. Higuchi and T. Niwa, "Evolving Hardware with GeneticLearning: A First Step towards Building a Darwin Machine," Proc.Second Int'l Conf. Simulation of Adaptive Behavior, pp. 417-424, 1993.
[7] A. Thompson, "Silicon Evolution," Proc. Ann. Conf. GeneticProgramming (GP '96), pp. 444-452, 1996.
[8] Z. Vasicek and L. Sekanina, "An Evolvable Hardware System inXilinx Virtex II Pro FPGA," Int'l J. Innovative Computing andApplications, vol. 1, no.1, pp. 63-73, 2007.
[9] R. Salvador, A. Otero, J. Mora, E. de la Torre, T. Riesgo, and L.Sekanina, "Implementation Techniques for Evolvable Hw Systems:Virtual vs. Dynamic Reconfiguration," Proc. Int'l Conf. FieldProgrammable Logic and Applications (FPL), pp. 547-550, 2012
[10] Nonlinear Filters for Image Processing, E.R. Dougherty and J.T.Astola, eds. Wiley, 1999