A Genetically Programmable Hybrid Virtual Reconfigurable Architecture for Image Filtering Applications

M. A. Almeida
Computer Science Dept.
UFSCar
S. Carlos, SP – Brazil

E. C. Pedrino
Computer Science Dept.
UFSCar
S. Carlos, SP – Brazil

M. C. Nicoletti
Computer Science Dept.
UFSCar & FACCAMP
SP, Brazil

***Abstract*—A new and efficient automatic hybrid method, called Hy-EH, based on Virtual Reconfigurable Architectures (VRAs) and implemented in Field Programmable Gate Arrays (FPGAs) is proposed, for a hardware-embedded construction of image filters. The method also encompass an evolutionary software system, which represents the chromosome as a bi-dimensional grid of function elements (FEs), entirely parameterized using the Verilog-HDL (Verilog Hardware Description Language), which is reconfigured using the MATLAB toolbox GPLAB, before its download into the FPGA. In the so-called intrinsic proposals, evolutionary processes take place internally to the hardware, in a pre-defined fixed way; in extrinsic proposals evolutionary processes happen externally to the hardware. The hybrid Hy-EH method, described in this paper allows for the intrinsic creation of a flexible-sized hardware, in an extrinsic way i.e., by means of an evolutionary process that happens externally to the hardware. Hy-EH is also a convenient choice as far as extrinsic methods are considered, since it does not depend on a proprietary solution for its implementation. A comparative analysis of using the Hy-EH versus an existing intrinsic proposal, in two well-known problems, has been conducted. Results show that by using Hy-EH there was little hardware complexity due to the optimized and more flexible use of shorter chromosomes.**

***Index Terms*—genetic programming; filtering images; hybrid reconfigurable architecture.**

I. Introduction

Evolvable hardware (EH) [1], [2], [3], [4], [5] is a branch of the evolutionary computation [6], [7], [8] area, focusing on evolving configurations of physical hardware, so its structures can be dynamically improved according to a pre-established performance criteria.

As an incipient research area, it has great potential for the development of innovative applications. Several works in evolvable hardware use genetic programming algorithms for implementing the evolution process ([9], [10]). A Genetic Programming (GP) algorithm [1], [11], [12] is an evolution-based algorithm whose main goal is to find a sequence of instructions (i.e., a program), from a pool of instructions initially given to the algorithm, which would perform a user-defined task. In this sense a GP algorithm can be approached as a Genetic Algorithm (GA) [13], [14], [15], [16], where each individual (chromosome) is a computer program, and its fitness is evaluated by the individual (i.e., a program) performing a given task.

EH has been lately explored using basically two strategies. In the extrinsic EH [17], software-based simulated evolution experiments help determining the best chromosome (for a particular hardware related task), which is then transferred into the reconfigurable hardware. In the intrinsic EH [18], reconfigurable circuit blocks, inside the hardware, are used and the process to evolve a best chromosome is accomplished within the hardware itself, by evaluating each individual in its own hardware, allowing for the process to explore the typical characteristics of the hardware, such as temperature or voltage levels, when calculating the fitness value, among others.

The intrinsic strategy [19], [16] has recently gained more visibility due to the emergence and development of reconfigurable-logic devices, such as FPGAs (Field Programmable Gate Arrays). However, the two strategies have limitations. The extrinsic, usually, produces larger chromosomes, when small size is a mandatory design aspect, for many EH applications; in the intrinsic case, in spite of using simpler GP routines, a large fraction of the FPGA resources is consumed by the GP built into the circuit. The intrinsic strategy also has little flexibility, considering that its circuit is fixed.

The remainder of this paper is organized as follows. Section II describes the proposal and implementation of a hybrid virtual reconfigurable arquitecture based on the use of genetic programming and aiming at image filtering applications. Section III gives some insights of a MatLab toolbox named GPLAB, used as a tool for inducing a chromosome (genetic program) for image filtering. Secton IV describes the use of the proposed system – Hy-EH – in two traditional image filtering problems. Results from Hy-EH and those obtained by another proposal, described in [19], are comparatively analysed in Section V. Conclusions and a few comments about the work are presented in Section VI.

II. The Hybrid Virtual Reconfigurable Architecture Genetically Programmable (Hy-EH)

In this paper an EH system based on VRA (Virtual Reconfigurable Architecture), and described in Verilog–HDL code, as a virtual layer for reconfiguring the FPGA, is proposed. The process the system implements can be described, in a general way, as: two images (noisy image and target image) are given as input to an instantiation of the GPLAB software, so to obtain a GP-based program that best filters the noisy image into the target image. The obtained program (best chromosome) is represented as a binary tree, as shown in Fig. (1), where the nodes represent operators and the leaves, pixels.

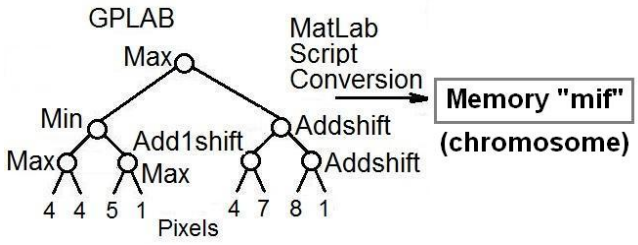

Fig. 1. GP-based program (chromosome), represented as a binary tree, that best filters a given image into a target image.

The binary tree is then translated into a hardware-based representation (.mif file in Quartus II 13.0), implemented as a memory cell, using an ad hoc MatLab based script (Fig. 2). So far the whole process is conducted in software.

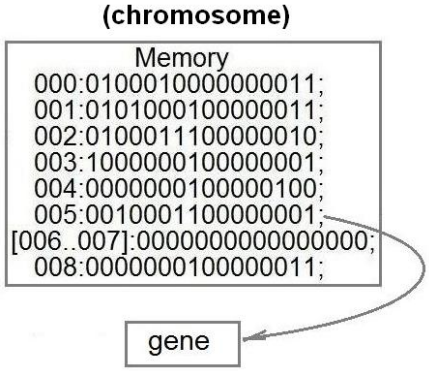

Fig. 2. A chromosome hardware-based representation, implemented as a memory cell (.mif file).

Next, a Verilog-HDL code, for sequentially reading each memory register (address) in the .mif file (Fig. (2)) and translating it into a Cartesian chromosome (virtual architecture is represented as a flexible bi-dimensional matrix – see Fig. (3)), is executed .

Each memory address describes a particular configuration of its corresponding matrix element (a gene). In the example shown in Fig. (1), where the resulting binary tree codifies to a 4×3 matrix (see Fig. (3) the configuration in address 5 (gene 5), '005:0010 0011 00000001', which is part of the memory cell, codifies a matrix element (FE for function element) by

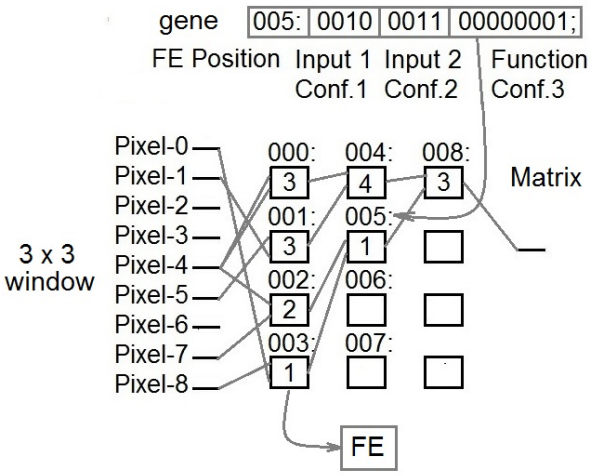

Fig. 3. Cartesian chromosome obtained from translating the memory registers in Fig. (2).

stating its inputs and its function. Its first nibble '0010' is a binary representation of the origin of its first input (i.e., the output of element 2) and the second nibble, '0011', is the origin of its second input (i.e., the output of element 3). The function (represented by the last byte, in this example) is the second function in Fig. (4).

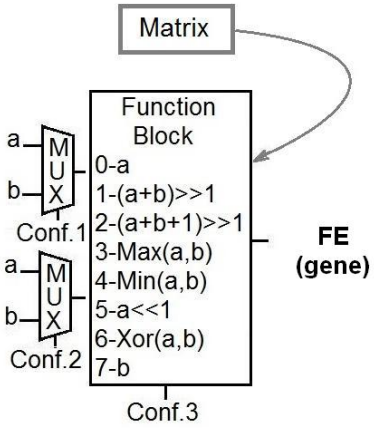

Fig. 4. The eight functions used when codifying chromosomes.

The matrix is organized such that the outputs of the FEs in its first column can be input operands for any FE in the next column, and so on. The FEs in the first column receive inputs from the environment and those in the last column provide outputs to the environment. The whole bi-dimensional matrix of FEs codifies a chromosome. Each FE, also known as gene (see Fig. (4), is associated with a set of basic functions and the selection of a particular function is conducted by a multiplexer

circuit (MUX), which is an inherent part of each gene in the chromosome. Functions can be of any type: logical, arithmetic, etc. FEs have been implemented as Verilog-HDL code. The chromosome employed in Hy-EH is represented as a bit string, that defines the functions and the connections of a specific FE in the matrix, as presented in Fig. (2) and Fig. (3).

The hardware comprises a development board having a FPGA from Altera [20]. For designing the circuit based on the code in the memory cell, a computer running the Altera software Quartus II 13.0 was used for developing a Verilog-HDL code representing the final circuit, which was then loaded into the FPGA. The code was developed to generate the FEs using a "for loop", dependent on the size of the matrix; it reads the memory and sequentially configures the FEs; each memory gene gives rise to a FE, as showed in Fig. (2) and Fig. (3). The same image given to the GPLAB is converted into a .mif file, via a MatLab script, so to make it readable under the Quartus II. Under Quartus was also defined a memory to be associated with the .mif file, a state machine (SM) to read the image from the input memory as well as a first-in-first-out (FIFO) structure to ease the reading process, as shown in Fig. (5).

The size of the FIFO structure depends on the image size as well as the chosen window size. For the experiments described in this work, a 3×3 pixel window was used. Each pixel read from the memory is buffered into the FIFO. After filling in the FIFO, at each clock cycle, a 3×3 window determines the 9 pixels to be supplied to the circuit (that represents the chromosome), as shown in Fig. (6). For an image of 256×256 size and a 3×3 pixel window, the FIFO structure has 2×256 pixels plus 3 extra pixels at the beginning of its third row. Fig. (6) shows how the information contained in the window is processed. Due to its hybrid approach, Hy-EH inherits benefits from both strategies: short chromosome, small matrix, lower hardware energy cost, a flexible circuit and, also, has high reusability, since its Verilog-HDL code allows for multi-platform use.

III. Technical background – The GPLAB Tool

The GPLAB tool [21] was configured as a GP set to find image filters. Among the several features made available by the GPLAB, the experiments used the full mode for tree initialization, the criteria for stopping the tree growth was depth $= 3$ (aiming at maintaining the chromosome with a fixed size and avoiding hardware variations), the population size was fixed with 200 individuals (chromosomes), the stopping criteria for the evolution process was 30 generations, an elitism of 1 individual was employed, and the genetic operators i.e., crossover and mutation, were used with adaptive rate. The crossover was applied to two parents, giving rise to two children and the mutation was applied to one individual, giving rise to another individual. The set of ad-hoc functions specifically designed for the application is briefly described in Table I, where 'a' and 'b' are the inputs for a gene.

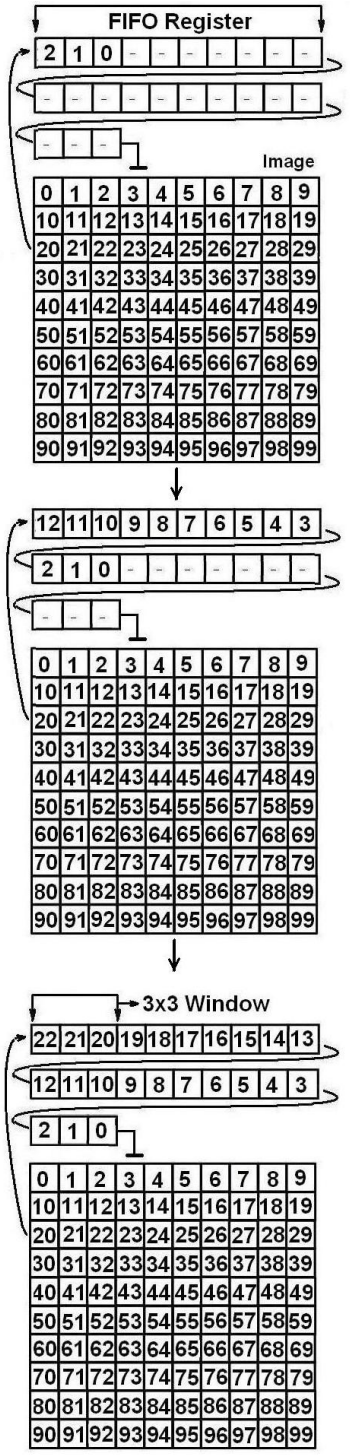

Fig. 5. The 3×3 window construction.

IV. Experiments

The fitness of each chromosome was determined by evaluating how close the resulting image, produced by applying the sequence of operators to the noisy image, is from the given target image. To measure the quality of the filtered image, the mean difference per pixel (MDPP), between the filtered and

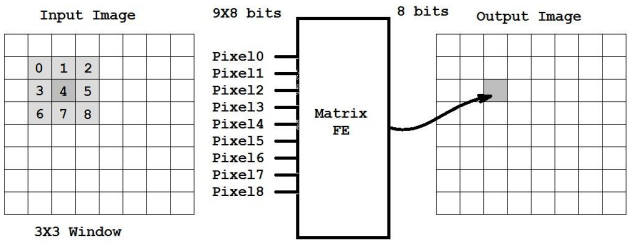

Fig. 6. Processing the window information.

TABLE I
THE EIGHT FUNCTIONS ASSOCIATED TO FES USED IN THE EXPERIMENTS.

Function	Description
0(000)	f_0(a,b) = a
1(001)	f_1(a,b) = add(a,b) >> 1
2(010)	f_2(a,b) = add(add(a,b),1) >> 1
3(011)	f_3(a,b) = max(a,b)
4(100)	f_4(a,b) = min(a,b)
5(101)	f_5(a,b) = a << 1 1
6(110)	f_6(a,b) = xor(a,b)
7(111)	f_7(a,b) = b

the target images, was calculated by Eq. (1) [19]. Considering target image sizes of $k \times k$, only a $(k-2) \times (k-2)$ region of the image is used, since the pixels on the image borders are not processed by the 3×3 window employed.

The best individuals are those with lower averages. For the GP learning process two different target images were used, in two different experiments. They are the'Lena' and 'Cameraman' target images, both with 256×256 grayscale pixels and 8 bits per pixel, shown on the left and on the right, respectively, in Fig. (7).

Fig. 7. Target images: Lena image (left) and Cameraman image (right).

Associated to each target image, a noisy image was artificially created, by adding to the target image a Gaussian noise with mean 0 and variance 0.008. The two images with the added noise are shown in Fig.(8). So, two learning experiments were run, each using a pair of images, namely: (lena_noisy, lena_ target) and (cameraman_noisy, cameraman_target).

For each pair of images, the GP was performed 100 times, as in [19], having as input both images; the sequence of operations represented by a chromosome was applied to the image with noise and its result compared with the target image,

Fig. 8. Noisy images: Lena image (left) and Cameraman image (right).

using Eq. (1) [22].

$$fitness_{MDFF} = \sum_{i=1}^{k-2} \sum_{i=1}^{k-2} |v(i,j) - w(i,j)| \qquad (1)$$

V. RESULTS AND DISCUSSION

The best individual of each evolutionary run was used for a further comparison with the results in [19]. The simulation of the best individual obtained by Hy-EH, in both images, are presented in Fig. (9).

Fig. 9. Filtered images by Hy-EH: the Lena image on the left and the Cameraman on the right.

The best individual found in the experiments in [19] was simulated in MatLab, using the same functions given in [19], for executing the filtering process. The simulation of the best individual obtained in [19], in both images, are presented in Fig. (10).

Fig. 10. Filtered images by a simulation of the best chromosome given in [19]: the Lena image on the left and the Cameraman on the right.

Table II shows a comparison of results related to FPGA costs, between a simulation with the Quartus II using the best chromosome given by Hy-EH and the results published in [19].

TABLE II
FP_GA COSTS (OCCUPATION). LEFT (HY-EH) ANDN RIGHT [19]

Cyclone II	Occupation	Virtex XCV2000E	Occupation
Logic Elem. (18,752)	1,633 (9 %)	Slices (19,200)	6,711 (35 %)

Table III shows the values of the common parameters used by both approaches, as evidence that Hy-EH employs more compact structures. The MDPP values of both best individuals found by Hy-EH and in [19] are in Table IV.

TABLE III
COMPARING PARAMETER VALUES.

Parameter	Hy–EH	Paper [19]
Size of FE matrix	4 rows, 3 columns	8 rows, 7 columns
Chromosome length	196 bits	441 bits

TABLE IV
COMPARING MDPP VALUES WHEN IMAGES HAVE GAUSSIAN NOISE.

Hy-EH		Paper [19]	
Cameraman	Lena	Cameraman	Lena
12.7097	12.3826	25.9728	20.2850

The filtered images (using both, the Hy-EH and a simulation of the use of the chromosome given in [19]) were compared, pixel by pixel, with the target images and their respective absolute difference of values were plotted. The simulation of the best individual obtained by the Hy-EH, in the Lena and the Cameraman images, are presented in Fig.(11) and Fig. (12), respectively. The simulation of the best individual obtained in [19], in the Lena and the Cameraman images, are presented in Fig.(13) and Fig. (14), respectively.

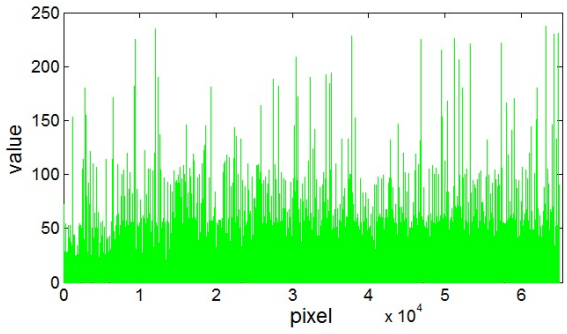

Fig. 11. Plotting of results produced by comparing, pixel by pixel, the Lena image target and the Lena image filtered by Hy-EH.

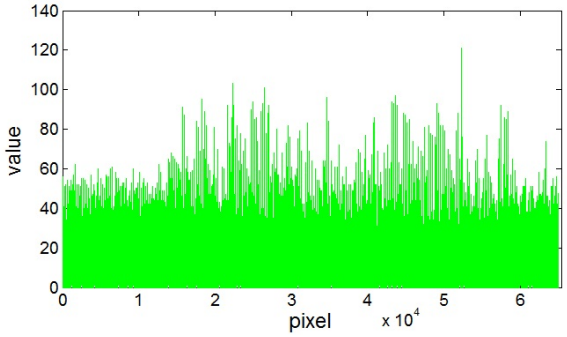

Fig. 12. Plotting of results produced by comparing, pixel by pixel, the Cameraman image target and the Cameraman image filtered by Hy-EH.

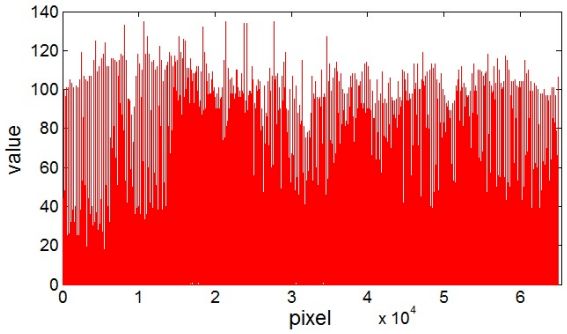

Fig. 13. Plotting of results produced by comparing, pixel by pixel, the Lena image target and the Lena image filtered using results given in [19].

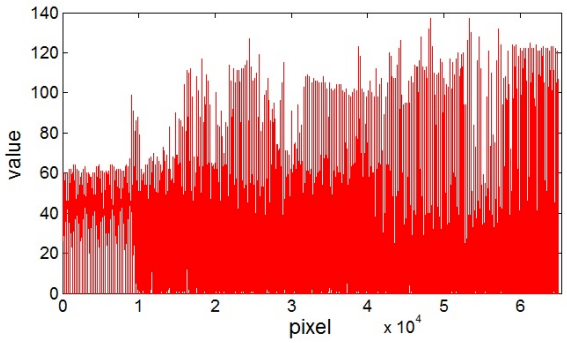

Fig. 14. Plotting of results produced by comparing, pixel by pixel, the Cameraman image target and the Cameraman image filtered using results given in [19].

The hardware simulation was performed with a 100 MHz clock, where the delay in processing each pixel was 40 ns. Taking into account 256×256 images, the resulting processing time was $256 \times 256 \times 40$ns = 2,622ms, allowing to process 380 frames per second. Assuming a resolution of 1024×1024, the image processing time will be 42ms, and the hardware would be able to process 23.8 frames per second.

VI. Conclusion

The Hy-EH proposed in this paper is a hybrid method for creating evolved hardware-based image filters, in an efficient way. It uses an extrinsic GP for obtaining optimized short chromosomes that codify image filters. The Hy-EH is a convenient choice since it does not depend on a proprietary solution for its implementation and it introduces much more flexibility into the process than the so called intrinsic methods, for the same task. See [23], [24] for details. As a continuation of the work a few more tasks have been planned; among them: (1) the method will be used with a larger set of images, so to confirm its feasibility and adequacy; (2) a comparison with standard image denoising methods, such as the median and the wiener filters [25] will be conducted.

Acknowledgment

We are grateful to the Brazilian funding agency FAPESP – Project No. 2015/23297-4. Our thanks go as well to UFSCar–DC, UFSCar-DF and FACCAMP, SP, Brazil.

References

[1] J. F. Miller, D. Job, and K. Vassilev, "Principles in the evolutionary design of digital circuits," *J. Genetic Program. Evolvable Mach.*, vol. 62, no. 1–3, pp. 8–35, 2000.

[2] R. Salvador, A. Otero, J. Mora, E. Torre, T. Riesgo, and L. Sekanina, "Self-reconfigurable evolvable hardware system for adaptive image processing," *IEEE Trransactions on Computers*, vol. 62, no. 8, pp. 1481–1493, 2013.

[3] F. Ni, Y. Li, X. Yang, F. Ni, and J. Xiang, "An orthogonal cartesian genetic programming algorithm for evolvable hardware," in *Proc. 2014 International Conference on Identification, Information and Knowledge in the Internet of Things (IIKI)*, 2014, pp. 220–224.

[4] R. Hrbacek and M. Sikulova, "Coevolutionary cartesian genetic programming in fpga," in *Proceedings of the Twelfth European Conference on the Synthesis and Simulation of Living Systems (ECAL2013)*, 2013, pp. 431–438.

[5] N. Nedjah and L. Mourelle, *Intelligent Data Engineering and Automated Learning*, N. Nedjah and L. Mourelle, Eds. Springer-Verlag, 2003, vol. 2690.

[6] D. B. Fogel, *Evolutionary Computation: the Fossil Record*. Wiley-IEEE Press, 1997.

[7] B. R. C. Alvareza, O. Cordon, and S. Damasa, "Evolutionary multiobjective optimization for mesh simplification of 3d open models," *Integrated Computer-Aided Engineering*, vol. 20, no. 4, pp. 375–390, 2013.

[8] J. F. Miller, P. Thomson, and T. Fogarty, *Genetic Algorithms and Evolution Strategies in Engineering and Computer Science*, D. Quagliarella and J. P. et al., Eds. Designing electronic circuits using evolutionary algorithms - arithmetic circuits: a case study: John Wiley and Sons, England, Chapter 7, 1997, pp. 105-131.

[9] Y. Wei, N. Wu, X. Zhang, and F. Zhou, "An online evolvable hardware system based on hardware ga and pla structure," in *Proceedings of the World Congress on Engineering and Computer Science (WCECS 2015)*, 2015, pp. 91–95.

[10] J. Wang, J. Liu, B. Feng, and G. Hou, "The dynamic evaluation strategy for evolvable hardware," in *Proc. of The 2015 Ninth International Conference on Frontier of Computer Science and Technology*, 2015, pp. 91–95.

[11] L. Sekanina and Z. Vasicek, "Approximate circuit design by means of evolvable hardware," in *Proc. of the 2013 IEEE International Conference on Evolvable Systems (ICES)*, 2013, pp. 21–28.

[12] J. Koza, *Genetic Programming*. MIT Press, 1992.

[13] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Publishing Company, Reading, MA, 1989.

[14] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs*. Springer-Verlag, 1992.

[15] H. Kim and H. Adeli, "Discrete cost optimization of composite floors using a floating point genetic algorithm," *Engineering Optimization*, vol. 33, no. 4, pp. 485–501, 2001.

[16] N. Siddique and H. Adeli, *Computational Intelligence - Synergies of Fuzzy Logic, Neural Networks and Evolutionary Computing*. Wiley, West Sussex, UK, 2013.

[17] I. Baradavka and T. Kalganova, "Assembling strategies in extrinsic evolvable hardware with bidirectional incremental evolution," in *Genetic Programming, Proceedings of EuroG'2003*. Springer-Verlag, 2003, pp. 276–285.

[18] Z. Jixiang, L. Yuanxiang, Z. Wei, X. Xuewen, and X. Xing, "Adaptive combinational logic circuits based on intrinsic evolvable hardware," in *Proc. of the 2009 IEEE Congress on Evolutionary Computation (CEC 2009)*, 2009, pp. 310–317.

[19] J. Wang, Q. S. Chen, and C. H. Lee, "Design and implementation of a virtual reconfigurable architecture for different applications of intrinsic evolvable hardware," *IET Computers & Digital Techniques*, vol. 2, no. 1–3, pp. 386–400, 2007.

[20] A. Corporation, *Cyclone II Device Handbook – Cyclone II Architeture*. Altera Co., 2007, vol. 1.

[21] S. Silva and J. Almeida, "Gplab – a genetic programming toolbox for matlab," in *Proceedings of the Nordic MatLab Conf, 2007*, 2007, pp. 271–278.

[22] L. Sekanina, "Virtual reconfigurable circuits for real-world applications of evolvable hardware," *Lecture Notres in Compute Science*, vol. 2606, pp. 186–197, 2003.

[23] E. C. Pedrino, J. H. Saito, and V. O. Roda, "A genetic programming approach to reconfigure a morphological image processing architecture," *International Journal of Reconfigurable Computing*, vol. 20, no. 3, pp. 712 494–712 503, 2010.

[24] E. Pedrino, J. Saito, V. O. Roda, E. R. R. Kato, M. L. T. Tronco, R. H. Tsunaki, O. Morandin, and M. C. Nicoletti, "A genetic programming based system for the automatic construction of image filters," *Integrated Computer-Aided Engineering*, vol. 20, no. 3, pp. 275–287, 2013.

[25] A. R. F. da Silva, "Wavelet de noising with evolutionary algorithms," *Digital Signal Processing*, vol. 15, no. 1–3, pp. 382–399, 2005.