Design and Implementation of Differential Evolution Algorithm on FPGA for Double-Precision Floating-Point Representation

Prometeo Cortés-Antonio[1], Josue Rangel-González[1], Luis A. Villa-Vargas[1], Marco Antonio Ramírez-Salinas[1], Herón Molina-Lozano[1], Ildar Batyrshin[2]

[1]National Polytechnic Institute, Center for Computing Research, México City, México; acorteoa09@sagitario.cic.ipn.mx, joshuab09@sagitario.cic.ipn.mx, lvilla@cic.ipn.mx, mars@cic.ipn.mx, hmolina@cic.ipn.mx

[2]Mexican Petroleum Institute, Eje Central Lázaro Cárdenas Norte 152, Col. San Bartolo Atepehuacan, Mexico, D.F., C.P. 07730, Mexico; batyr@imp.mx

Abstract: The paper presents the results of implementation of differential evolution algorithm on FPGA using floating point representation with double precision useful in real numeric problems. Verilog Hardware Description Language (HDL) was used for Altera hardware design. Schematics of the modules of differential evolution algorithm are presented. The performance of the design is evaluated through six different functions problems implemented in hardware.

Keywords: FPGA; Differential Evolution Algorithm; floating point

1 Introduction

Metaheuristic optimization algorithms such as Genetic Algorithms, Estimation of Distribution Algorithms, Differential Evolution Algorithms, Particle Swarm Optimization, Ant Colony Optimization etc. have been widely accepted in engineering, economics and biotechnology optimization problems because they are derivative free optimization methods that can be used for optimization of complex functions [1, 2].

Implementation of the Differential Evolution Algorithm (DEA) on software has been used in applications such as [3-10], where an optimization of parametric model is carried out in conventional computer equipment. However the applications where optimization is necessary in runtime, for example in online learning [11-13] and remote access [14, 15], require that DEA to be implemented in embedded systems such as FPGA device using evolvable hardware approach [16-18].

Several proposals for hardware implementation of evolutionary algorithms have been realized, such as Micro Algorithm [19-23] and Compact Genetic Algorithm (cGA) [24-28] with the aim of low resource consumption and minimal response time implementation. These algorithms lose the generality of solving problems of any kind, however such deployments have had success in combinatorial problems. However as it is shown in [29], cGA not always show good performance in solving nonlinear problems, and also complex linear problems. Furthermore if one considers the implementation of cGA presented in [30], where the probability vector is implemented using 8-bit integers, it is also clear that this implementation is limited by solution of only trivial problems.

It was shown in the seminal paper on Differential Evolution Algorithm [31] that this algorithm is very simple using only three evolutionary parameters and basic operations such as addition, subtraction, comparison, and its performance is comparable or even surpasses other evolutionary or heuristic algorithms. However, due to DEA used real value representation of variables and its operations are performed in floating point its hardware (FPGA) implementation in the time when this algorithm was published was not possible because FPGA in that time did not have the necessary resources for such implementations. Nowadays FPGA families have amazing abilities that make the implementation of such algorithms not only feasible, but also an excellent choice for designing evolutionary algorithms.

There are several design proposals for implementing evolutionary algorithms ranging from a dedicated system on only one chip until a cluster of FPGAs [32, 33] to perform concurrent computation, that can be useful for different applications.

The paper presents the design on EP4CE115F29C7 Altera FPGA device [34] for a Differential Evolution Algorithm with a number of function variables from 4 until 32 and population size from 16 to 128 using double-precision floating point representation. This work is divided into six sections. The next section gives theoretical bases of Differential Evolution Algorithms. A brief introduction to the Altera FPGA logic design is presented in Section 3. Section 4 presents the proposed design of the DEA showing schematics of each block that makes up the system. The results of resource consumption and latency time of the implementation are given in Section 5. Section 6 presents the conclusions and directions of future work.

2 Differential Evolution Algorithm

Differential Evolution Algorithm (DEA) belongs to the family of evolutionary algorithms, which has as aim to find the global optimum of a function over continuous space. In particular, and without loss of generality, this problem can be reduced to finding the minimum of a function:

$$minimize\ f(x) = f(x_1, x_1, \ldots, x_n) \qquad (1)$$

Where x is a n-dimensional vector and f is a real function of real valued arguments. DEA, proposed in [31], is an evolutionary algorithm that requires only three parameters CR (defining crossover and mutation operations that are mutually exclusive), F (scaling factor of the difference of two individuals) and NP (population size) to generate the evolutionary process for *n*-dimensional problem.

Differential Evolution Algorithm can be represented by a four-step process as shown in Fig. 1. Only the first step is performed once, the other steps are performed while an iterative process does not terminated by stop criteria.

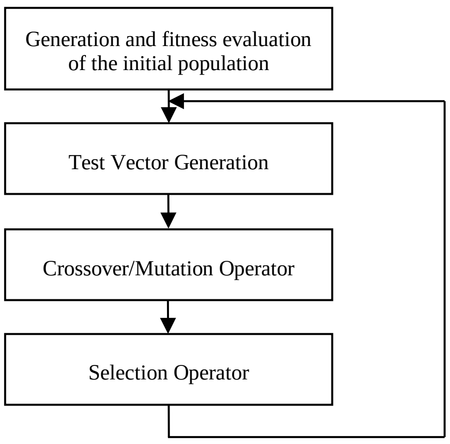

Generation and fitness evaluation of the initial population
Test Vector Generation
Crossover/Mutation Operator
Selection Operator

Figure 1
Bock diagram of Differential Evolution Algorithm

Complete pseudo-code is presented in Fig. 2, where the first 12 lines perform the block of generation and fitness evaluation of the initial population shown in Fig. 1, for dimensionality *D* and population size *NP*.

The algorithm contains three nested loops, where the outer loop is used to specify the stop condition, in this particular case it is determined by the parameter G_{max}(number of generations) but one can set other stop conditions such as minimum error or difference between sequential errors, etc.

The inner cycle indicates that for each individual in a generation with the probability defined by the parameter CR it is generated a new individual from three individuals chosen randomly, with indexes r_1, r_2 and r_3, using scale factor F, as described in line 21 of the algorithm. This cycle can be considered as a combination of crossover and mutation operations [31].

```
1  Begin
2  $G = 0$
3  $Create\ a\ random\ initial\ population$
4  $\boldsymbol{for}\ i = 1\ \boldsymbol{to}\ NP\ \boldsymbol{do}$
5    $\boldsymbol{for} j = 1\ \boldsymbol{to}\ \boldsymbol{D}$do
6        $x_{j,i}^{(G=0)} = x_j^{min} + rand_j[0,1].\left(x_j^{max} - x_j^{min}\right)$
7      $\boldsymbol{end\ for}$
8    $\boldsymbol{end\ for}$
9  $Evaluate\ Fitness\ Function\ for\ each\ individual\ of\ population$
10   $\boldsymbol{for} i = 1\ \boldsymbol{to}\ NP\ \boldsymbol{do}$
11       $f\left(x_i^{(G=0)}\right)$
12   $\boldsymbol{end\ for}$
13 $Test\ vector\ generation$
14 $\boldsymbol{for}\ G = 1\ \boldsymbol{to}\ MaxGen\ \boldsymbol{do}$
15   $\boldsymbol{for} i = 1\ \boldsymbol{to}\ NP\ \boldsymbol{do}$
16 $Select\ Randomly\ r1\ , r2, r3 \in [1, NP], r1 \neq r2 \neq r3 \neq i$
17 $Mutation\ and\ Crossover\ Process$
18       $jrand = randInt[1{:}\,D]$
19       $\boldsymbol{for} j = 1\ \boldsymbol{to} D \boldsymbol{do}$
20         $\boldsymbol{if}(rand[0,1] < CR \boldsymbol{or} j == jrand)\boldsymbol{then}$
21             $v_{i,j}^{(G+1)} = x_{i,r1}^{(G)} + F * \left(x_{i,r2}^{(G)} - x_{i,r3}^{(G)}\right)$
22         $\boldsymbol{else}$
23             $v_{i,j}^{(G+1)} = x_{i,j}^{(G)}$
24         $\boldsymbol{endif}$
25       $\boldsymbol{endfor}$
26     $\boldsymbol{endfor}$
27 $Seleccion$
28     $\boldsymbol{if}\left(f\left(v^{(G+1)}\right) \leq f\left(x_i^{(G)}\right)\right)\boldsymbol{then}$
29         $x_i^{(G+1)} = v_{i,j}^{(G+1)}$
30     $\boldsymbol{else}$
31         $x_i^{(G+1)} = x_i^{(G)}$
32     $\boldsymbol{endif}$
33   $\boldsymbol{endfor}$
34 $\boldsymbol{endFor}$
35 End
```

Figure 2

Pseudocode of Differential Evolution Algorithm

3 FPGA Device

The FPGA (Field-Programable Gate Array) is a device that is used to design a dedicated digital system or embedded platforms that perform specific tasks in a system. Its main characteristic is that it can be programmed several times, even after the system has been installed or finished to update its functionality. For this reason this device is very useful in evolved applications with dynamic environments.

Currently, FPGA has been used widely in several real applications of evolvable hardware, an emerging research area where intelligent computation techniques are implemented in digital system design that can be adaptive to environment changes, manage big data and process the information using intelligent techniques.

Altera FPGA [35] is a device consisted of programmable logic blocks (Logic Elements), and memory elements (Dedicated Logic Registers), which are interconnected to perform complex combinational and sequential functions. In addition it can contain specific resources such as embedded multipliers, SRAM, transceiver, or even hard intellectual propriety (IP) block and embedded processors for implementing SoC design. FPGA based system is implemented through modules describing basic digital logic circuits such as multiplexers, comparators, adders, registers, memory, and finite state machines use hardware description languages to perform specific and complex system tasks.

Altera provides a free library of parameterized intellectual property (IP) blocks called Megafunctions [36, 37]. The floating point Megafuctions implement hardware modules for performing customized floating point operations. Table 1 shows the resources used to perform the floating point arithmetic operations in Differential Evolution Algorithm implementation on EP4CE115F29C7 device for double precision floating point representation.

Table 1
Characteristics of tree floating point Altera Megafunction

MegafunctionName	Output Latency	Logic Elements	Logic Registers	Embedded multiplier	F_{MAX} (MHz)
FPMULT	5	552	530	18	102
FPCOMP	1	176	2	-	-
FPAddSub	7	1534	584	-	105

Differential evolution algorithm performs floating point operations only for generating the offspring individuals in mutation and crossover process; hence it needs only one module for floating point. Moreover, it is important to see that FPCOMP is a combinatorial module; because of the floating point comparator is the same that integer comparator. The complete hardware implementation of DEA is described in the next section.

4 Hardware Implementation of Differential Evolution Algorithm

The schematic hardware implementation of DEA consists of the following modules: i) PMem module to store individuals, ii) FXMem to store fitness function values, iii) fitness function module, iv) CrossOvermodule, v) four Random Number Generators and vi) Finite State Machine module to control the execution sequence of DEA. Fig. 3 presents all modules except of Finite State Machine module that controls all modules of the system. Fig. 3 depicts also the following registers: *i*,*j*,for addressing PMem and FXMem, three registers for storing indexes, three 64-bits registers for storing the values of X_{r1}, X_{r2} and X_{r3}attributes and a file register with D64-bits register for storing each attribute of offspring individual. Also some multiplexors and comparators are used that are not presented due to simplicity of the scheme. In the following the more detailed description of the modules will be given.

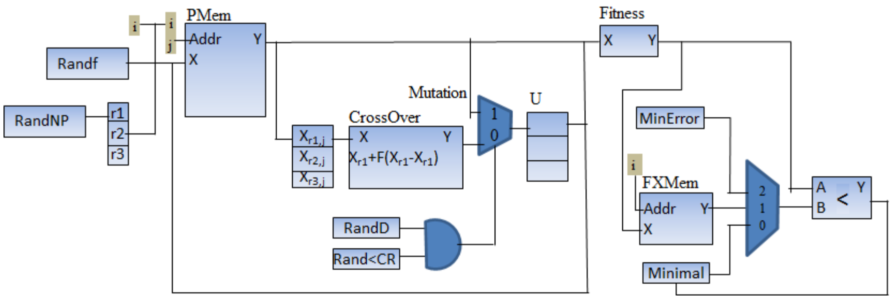


Figure 3

Complete hardware implementation of Differential Evolution Algorithm

4.1 Memories Modules

4.1.1 PMem Module

This module is implemented by using a RAM circuit for storing the population of current generation. Memory size is determined by population size parameter *NP*, and dimensionality *D*, the RAM size can be expressed as follows:

$$PMem[TAM] = NP * D \; words \tag{2}$$

If each word is specified by 8 bytes (64 bits), then the PMem size expressed in bytes is specified as follows:

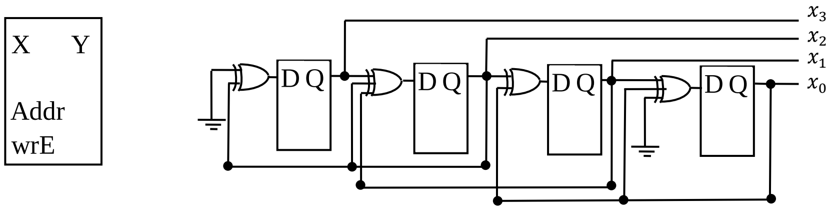
$$PMem[TAM] = NP * D * 8 \;\; bytes \tag{3}$$

4.1.2 FXMem Module

This module is implemented similarly to PMem with the difference that FXMem size is determined only by *NP* parameter, due to only one value is stored by individual. The expressions for FXMem[TAM] are:

$$FXMem[TAM] = NP\ words \tag{4}$$
$$FXMem[TAM] = NP * 8\ \ bytes \tag{5}$$

Fig. 4a shows the block diagram of RAM specifying all control pin.



a) Memory b) Random Number generator

Figure 4

Schemes of general memory modules and Random Number Generator of 4 cells

4.2 Random Number Generator (RNG)

Cellular Automata(CA) circuits have been used to create random numbers. The corresponding module works with two rules[38] where the first one is defined by:$a_i(t+1) = a_{i-1}(t) \oplus a_{i+1}(t)$ and the second one is defined by: $a_i(t+1) = a_{i-1}(t) \oplus a_i(t) \oplus a_{i+1}(t)$, where $a_i(t+1)$represents the next state of the *i*-cell, $a_{i-1}(t)$ represents the current state of the (*i*-1)-cell (left neighbor), $a_i(t)$ represents the current state of the *i*-cell, and $a_{i+1}(t)$ is the current state of the (i+1)-cell (right neighbor). An n-CA can generate 2^m-1 different pseudo random numbers where *m* is a number of the cells. The scheme for *m* = 4 is presented in Fig. 4b.

Implementation of DEA contains 3 different modules for generating integer values in intervals [0-NP], [0-D], [0-127] and one module for generating floating point values.

For design of a comparator with parameter CR taking values in interval [0,1] instead of floating point representation of parameter values it is used a digital representation in interval [0-127] by means of 7 cells of CA.

For floating point number generator used for generation of values of individuals.

4.3 Crossover Module

To implement the pseudo code shown in Fig. 2, line 21:

$$v_{i,j}^{(G+1)} = x_{i,r1}^{(G)} + F * \left(x_{i,r2}^{(G)} - x_{i,r3}^{(G)}\right)$$

where $x_{i,r1}^{(G)}$, $x_{i,r2}^{(G)}$, $x_{i,r3}^{(G)}$and F are floating point values,the following three binary floating point operations have been used:

$$R1 = \left(x_{i,r2}^{(G)} - x_{i,r3}^{(G)}\right)$$

$$R2 = F * R1$$

$$v_{i,j}^{(G+1)} = x_{i,r1}^{(G)} + R2$$

This sequence of operations is implemented using FPMult and FPAddSub Megafuntions shown in Table 1. Therefore for a complete crossover operation be performed, it should run 22 clock cycles. Fig. 5 shows scheme of Crossover module.

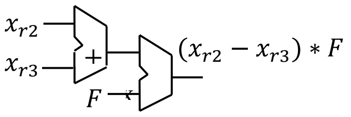
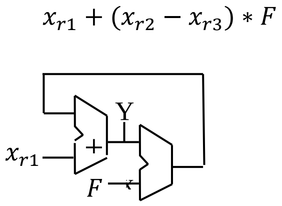
	 $x_{r1} + (x_{r2} - x_{r3}) * F$ Y + x_{r1} F
a) Crossover module. Stage 1	b) Crossover module. Stage 2

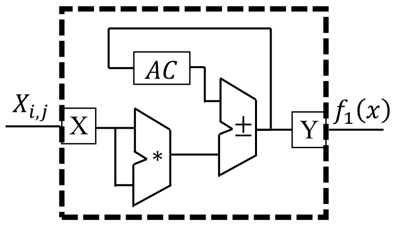
Figure 5
CrossOver module implementations

4.4 Fitness Module

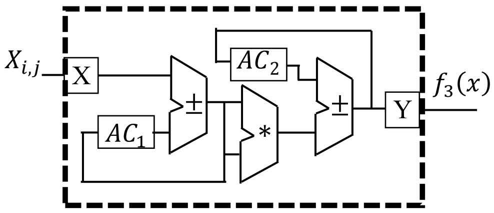
Fitness evaluation modules are dependent from specific applications therefore this modules are the only components that change from one application to another. In this paper we implement a set of six different benchmark mathematical functions traditionally used for evaluation of performance of metaheuristic algorithms (Table 2). The block diagram implementations of these functions are shown in Fig. 6.

Table 2
Benchmark mathematical functions

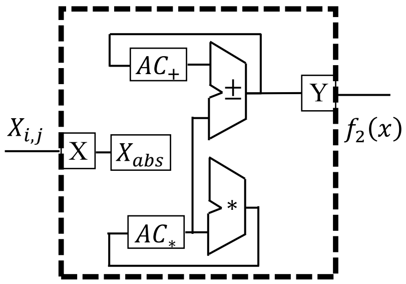
	Function	Ecuation
1	Sphere Mode	$f_1(x)=\sum_{i=1}^{D} x_i^2$
2	Schwefel's Problem 2.22	$f_2(x)=\sum_{i=1}^{D}\lvert x_i\rvert+\prod_{i=1}^{D}\lvert x_i\rvert$
3	Schwefel's Problem 1.2	$f_3(x)=\sum_{i=1}^{D}\left(\sum_{j=1}^{i} x_j\right)^2$
4	Schwefel's Problem 2.21	$f_4(x)=\max\{\lvert x_i\rvert, 1\le i\le D\}$
5	Generalized Rosenbrock's Function	$f_5(x)=\sum_{i=1}^{D}\left\lvert 100(x_{i+1-}x_i^2)^2+(x_i-1)^2\right\rvert$
6	Step Function	$f_6(x)=\sum_{i=1}^{D}\lvert(x_i+0.5)\rvert^2$



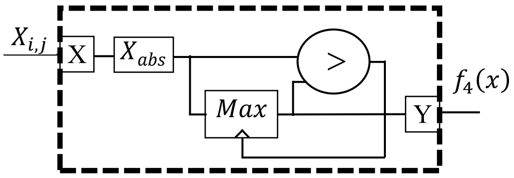
a)



b)

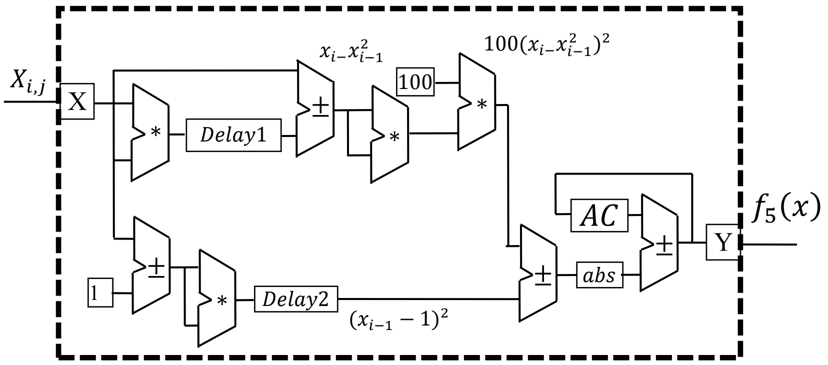


c)

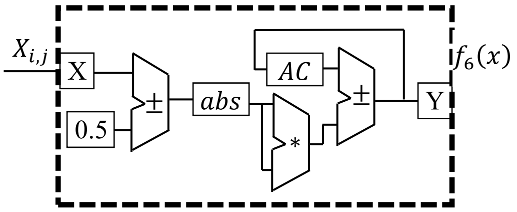


d)

Figure 6.1
Fitness Functions Implementations



e)



f)

Figure 6.2
Fitness Functions Implementations

4.5 Control Module

Each one of the considered modules contains a control inputs for deciding when to write or to read values on the register and which elements should be selected for a specific input. Control signals are managed by a control unit that performs the correct functionality of algorithm.

5 Results

Results presented in Table 3 show the resources consumed in the implementation of DEA with spherical objective function (f1) with *NP*=128 and *W*= 32 parameter values on EP4CE115F29C7 device of Cyclone IV E Altera Family. The Results column presents both the resources used in the implementation vs. the total available resources used by the device for different categories of resources. The column f1 of Table 4 shows what part of resources of Table 3 consumed by function f1. Total resources consumed in implementation of other benchmark functions used for evaluation of DEA performance also can be found in Table 4.

For evaluating the time performance of DEA implementation two tests over benchmark mathematical functions f_1-f_6 have been applied for two different *NP* and *W* parameter values. Table 5a shows results for parameter values *NP*= 128 and *W*= 32; Table 5b shows the results for *NP*= 16 and *W*= 4. The parameter values for CR and F used in simulations are taken from analysis presented in [39] where it was argued that these values are the best values for obtaining optimum with small number of generations. The parameter values presented for average number of generations (AveGen), average time (AveTime) and Error were obtained after the 20 running of the algorithm using an error $1e^{-12}$ or 20,000 generations as stop conditions.

Table 3

Resources consumed in DEA implementation

Category	Results
Total Combinational Functions (TCF)	10,330/114,480 (9%)
Dedicated Logic Registers (DLR)	5,366 /114,480 (5%)
Total Memory bits (TMb)	8480/3,981,312 (<1%)
Embedded Multiplier 9-bit elements (EM9)	72/512 (14%)
Fmax (MHz)	95 MHz

Table 4

Resources usedin implementation of objective functions

Category	f_1	f_2	f_3	f_4	f_5	f_6
TCF	2405	2915	4397	213	8501	4281
DLR	1182	1494	1929	74	4551	1880
TMb	71	54	140	0	293	113
EM9	18	18	18	0	72	18
F_{max}(MHz)	95	95	70	122	71	80
Latency	8clk*W	11clk* W+10clk	12clk* W	3clk* W	17clk* W	12clk* W

Table 5

Consuming time for different objective functions

a) *NP*=128,*W*=32

f(x)	CR	F	AveGen	AveTime(s)	Error
$f_1(x)$	0.9	0.7	11571	26.67	$1.00E^{-12}$
$f_2(x)$	0.2	0.7	2802	7.7484	$1.00E^{-12}$
$f_3(x)$	0.9	0.7	20000	442.0546	0.0911
$f_4(x)$	0.8	0.7	20000	91.8609	0.089
$f_5(x)$	0.9	0.7	20000	119.9898	$1.45E^{-06}$
$f_6(x)$	0	0.7	1521	7.4476	$2.57E^{-13}$

b) *NP*=16, *W*=4

f(x)	CR	F	AveGen	AveTime	Error
$f_1(x)$	0.9	0.7	121	6 ms	1.00E-12
$f_2(x)$	0.2	0.7	250	16 ms	1.00E-12
$f_3(x)$	0.9	0.7	144	75 ms	1.00E-12
$f_4(x)$	0.8	0.7	501	53 ms	1.00E-12
$f_5(x)$	0.9	0.7	1683	.24 s	1.00E-12
$f_6(x)$	0	0.7	101	12 ms	1.00E-12

Conclusions

The paper presented the design of Differential Evolution Algorithm on Altera FPGA following a sequential flow and using three parameter values defining crossover and mutation operations, scaling factor and population size.

The design does not exploit parallelism approach because we think that this technique depends of specific application. However we can mention that parallelism is more adequate in fitness functions module because it is a temporal bottleneck of many applications and its implementation is straightforward.

The paper describes an implementation of the basic version of EDA considered in [31].There exist several modifications of Differential Evolution Algorithm, with the following principal variations: a) the change of the number of individuals that participate in the crossover process, which are incrementing in even numbers for better exploring of the space; b) the use of the best individual of the population as a principal ancestor for better exploiting his local neighborhood into search space; c) the way in which the crossover-mutation operator is implemented. For more details about modified DEA see [39, 40]. The FPGA implementation of these variations of DEA are straightforward.

The paper contains the original results of research that were not submitted to other journals or conferences.

References

[1] H. Nejat Pishkenari, S. H. Mahboobib, A. Alasty, "Optimum Synthesis of Fuzzy Logic Controller for Trajectory Tracking by Differential Evolution", Scientia Iranica, Iran, pp. 261-267, April, 2011

[2] Shing-Tai Pan, "Evolutionary Computation on Programmable Robust IIR Filter Pole-Placement Design", Instrumentation and Measurement, Vol. 60 , pp. 1469-1479, April 2011

[3] A. Chandra, S. Chattopadhyay, "Novel Approach of Designing Multiplier-less Finite Impulse Response Filter using Differential Evolution Algorithm", Intelligent Systems and Applications, Vol. 4, pp. 54-62, June 2012

[4] A. Chandra, S. Chattopadhyay, "Role of Mutation Strategies of Differential Evolution Algorithm in Designing Hardware Efficient Multiplier-less Low-pass FIR Filter", Journal of Multimedia, Vol. 7, No. 5, pp. 353-363, October 2012

[5] A. Hiendro, "Multiple Switching Patterns for SHEPWM Inverters Using Differential Evolution Algorithms", International Journal of Power Electronics and Drive System, Vol. 1, pp. 94-103, December 2011

[6] C. Cheng-Hung, L. Cheng-Jian, Member, L. Chin-Teng, "Nonlinear System Control Using Adaptive Neural Fuzzy Networks Based on a Modified Differential Evolution", Systems, Man, and Cybernetics, IEEE, Vol. 39, pp. 459-473 , July, 2009

[7] C. J. François, et al, "FPGA Implementation of Genetic Algorithm for UAV Real-Time Path Planning", Intelligent and Robotic Systems, Vol. 54, pp. 495-510, March 2009

[8] D. Zaharie, D. Petcu, "Parallel Implementation of Multi-Population Differential Evolution", Concurrent Information Processing and Computing, IOS, Press, pp. 223-232, 2005

[9] V. Tirronen, et al., "An Enhanced Memetic Differential Evolution in Filter Design for Defect Detection in Paper Production", Evolutionary Computation, Vol. 16, No. 4, pp. 529-555, 2008

[10] W. Kwedlo, K. Bandurski, "A Parallel Differential Evolution Algorithm for Neural Network Training", Parallel Computing in Electrical Engineering, pp. 319-324, Sept. 2006

[11] H. Shayani, P. J. Bentley, A. M. Tyrrell, "Hardware Implementation of a Bio-plausible Neuron Model for Evolution and Growth of Spiking Neural Networks on FPGA", Adaptive Hardware and Systems, NASA/ESA, pp. 236-243, 2008

[12] J. M. Sánchez-Pérez, et, "Genetic Algorithms Using Parallelism and FPGAs: The TSP as Case Study", Parallel Processing, Portland, Oregon, USA, pp. 573-579, June, 2005

[13] R. Lovassy, L. T. Kóczy, L. G, "Function Approximation Performance of Fuzzy Neural Networks", Acta Polytechnica Hungarica, Vol. 7, No. 4, pp. 25-38, 2010

[14] E. Magdaleno, M. Rodríguez, F. Pérez, D. Hernández and E. García, "A FPGA Embedded Web Server for Remote Monitoring and Control of Smart Sensors Networks", sensors, Vol. 14, pp. 416-430, 2014

[15] R. Patel, A. Rajawat, R. N. Yadav, "Remote Access of Peripherals using Web Server on FPGA Platform", International Conference on Recent Trends in Information, Telecommunication and Computing, India, pp. 274-276, 2010

[16] E. Sanchez, M. Tomassini, "Towards Evolvable Hardware", Lecture Notes in Computer Science, Springer, Vol. 1062, 1995

[17] K. Hwang, S. Cho, "Improving Evolvable Hardware by Applying the Speciation Technique", Applied Soft Computing, Vol. 9, pp. 254-263, 2009

[18] Y. Thoma and E. Sanchez, "A Reconfigurable Chip for Evolvable Hardware", GECCO, Springer-Verlag Berlin Heidelberg, Vol. 3102, pp. 816-827, 2004

[19] K. Krishnakumar, "Micro-Genetic Algorithms for Stationary and non-Stationary Function Optimization", Intelligent Control and Adaptive Systems, Vol. 1196, pp. 289-296, 1989

[20] F. Viveros-Jiménez, E. Mezura-Montes, A. Gelbukh, "Elitistic Evolution: a Novel Micro-Population Approach for global optimization problems", Eighth Mexican International Conference on Artificial Intelligence, IEEE, México, pp. 15-20, 2009

[21] C. A. Coello-Coello, G. Tosano-Pulido, "A Miro-Geneti Algorithm for Multiobjetive Optimization", Evolutionary Multi-Criterion Optimization, Switzerland, pp. 126-140, 2001

[22] Wu, D., Gan, D. and Jiang, J. N, "An Improved Micro-Particle Swarm Optimization Algorithm and Its Application in Transient Stability Constrained Optimal Power Flow", International Transactions on Electrical Energy Systems, Vol. 24, pp. 395-411, 2012

[23] Huang T, Mohan AS, "Micro-Particle Swarm Optimizer for Solving High Dimensional Optimization Problems", Applied Mathematics and Computation, Vol. 181, pp. 1148-1154, 2006

[24] C. Ying-ping, C. Chao-Hong, "Enabling the Extended Compact Genetic Algorithm for Real-Parameter Optimization by Using Adaptive Discretization", Evolutionary Computation, Vol. 18, No. 2, pp. 199-228, 2010

[25] E. Mininno at al, "Compact Differential Evolution", IEEE Transactions on Evolutionary Computation, Vol. 15, pp. 32-54, February 2011

[26] J. I. Hidalgo, et al., "A Parallel Compact Genetic Algorithm for Multi-FPGA Partitioning", Parallel and Distributed Processing, Mantova, Italy, pp. 113-120, February, 2001

[27] K. H. Tsoi, K. H. Leung, P. H. W. Leong, "Compact FPGA-based True and Pseudo Random Number Generators", 11th Field-Programmable Custom Computing Machines, Napa, California, USA, pp. 51-61 , April, 2003

[28] Y. Jewajinda, P. Chongstitvatana, "FPGA Implementation of a Cellular Compact Genetic Algorithm", Adaptive Hardware and Systems, NASA/ESA, pp. 385-390, 2008

[29] R. Rastegar, A. Hariri, "A Step Forward in Studying the Compact Genetic Algorithm", Evolutionary Computation, Vol. 14, No. 3, pp. 277-289, August, 2006

[30] C. Aporntewan, P. Chongstitvatana, "A Hardware Implementation of the Compact Genetic Algorithm", Evolutionary Computation, Seoul, Korea, Vol. 1, pp. 624-629, May, 2001

[31] R. Storn and K. Price, "Differential Evolution - A Simple and Efficient Adaptive Scheme for Global Optimization over Continuous Spaces", Journal of Global Optimization, Vol. 11, pp. 341-359, March, 1995

[32] A. León-Javier, M. A. Moreno-Armendáriz, N. Cruz-Cortés, "Designing a Compact Genetic Algorithm with Minimal FPGA Resources", Advances in Computational Intelligence, Springer, Vol. 116, pp. 349-357, 2009

[33] A. Swarnalatha, A. P. Shanthi, "Optimization of Single Variable Functions Using Complete Hardware Evolution", Applied Soft Computing, Vol. 12, pp. 1322-1329, 2012

[34] Altera Inc, ftp://ftp.altera.com/up/pub/Altera_Material/12.1/Boards/DE2-115/DE2_115_User_Manual.pdf

[35] Altera Inc, "http://www.altera.com/literature/hb/cyclone-iv/cyclone4-handbook.pdf"

[36] Altera Inc, "http://www.altera.com/literature/ug/ug_intro_to_megafunctions.pdf"

[37] Altera Inc, "http://www.altera.com/literature/ug/ug_altfp_mfug.pdf".

[38] P. D. Hortensius, R. D. McLeod and H. C. Card, "Parallel Random Number Generation for VLSI Systems Using Cellular Automata," IEEE Transactions on Computers, Vol. 38, No. 10, pp. 1466-1473, 1989

[39] E. Mezura, J. Velázquez y C. A. Coello, "A Comparative Study of Differential Evolution Variants for Global Optimization", 8th annual conference on GECCO, USA, pp. 485-492, 2006

[40] A. K. Qin, V. L. Huang, and P. N. Suganthan, "Differential Evolution Algorithm with Strategy Adaptation for Global Numerical Optimization", IEEE Transactions on Evolutionary Computation, Vol. 12, No. 2, pp. 398-417, April, 2009