FPGA Implementation of Parallel Particle Swarm Optimization Algorithm and Compared with Genetic Algorithm

BEN AMEUR Mohamed sadek
Laboratory of Microelectronic, university of Monastir,
Monastir, Tunisia

SAKLY Anis
National Engineering School of Monastir,
Monastir, Tunisia

***Abstract*—In this paper, a digital implementation of Particle Swarm Optimization algorithm (PSO) is developed for implementation on Field Programmable Gate Array (FPGA). PSO is a recent intelligent heuristic search method in which the mechanism of algorithm is inspired by the swarming of biological populations. PSO is similar to the Genetic Algorithm (GA). In fact, both of them use a combination of deterministic and probabilistic rules. The experimental results of this algorithm are effective to evaluate the performance of the PSO compared to GA and other PSO algorithm. New digital solutions are available to generate a hardware implementation of PSO Algorithms. Thus, we developed a hardware architecture based on Finite state machine (FSM) and implemented into FPGA to solve some dispatch computing problems over other circuits based on swarm intelligence. Moreover, the inherent parallelism of these new hardware solutions with a large computational capacity makes the running time negligible regardless the complexity of the processing.**

Keywords*—PSO algorithm; GA; FPGA; Finite state machine; hardware

I. INTRODUCTION

Over the last decade, several meta-heuristic algorithms are proposed to solve hard and complex optimization problems. The effectiveness of this algorithm give satisfaction to solve the most difficult problems for many algorithms related for various optimization problems. The proposed architecture is tested on some benchmarks functions. We have also analyzed the operators of GAs to describe how the performance of each one can be enhanced by incorporating some features of the other. We used standard benchmarks functions to make comparison between the two algorithms. In fact, PSO algorithm use the technique [1] that explores all the search space to fix parameters that minimizes or maximizes a problem. So, the ability and the simplicity to solve complex problems make the studies active in this area compared with many others optimization techniques [2] [3].

This research attempts to present that PSO has a good effectiveness to find the best global optimal solution as the GA but with a better computing efficiency (less using of resource hardware and execution time). The main objective of this paper is to compare the computational efficiency of our optimized PSO with GA and other PSO algorithms using a set of benchmark test problems. The results of this optimization algorithm could prove to be important for the future study of PSO. The organization of the paper is described as follow: The first chapter briefly introduces the general steps performing the mechanism of PSO. Especially, a brief introduction of pseudo random number generator [4]. The next section describes the background functional architecture which performs the GA and PSO algorithm. In chapter 3, a description of the architecture used in the hardware implementation of PSO and genetic algorithm; the second part illustrates the experimental results of some benchmarks functions applied into the PSO algorithm and compared with GA and others PSO algorithms. Finally, we conclude our work and we make some implications and directions for future studies.

II. PARTICLE SWARM OPTIMIZATION

In Particle Swarm Optimization algorithm we can say that each « bird » may be a solution through a search space. Birds are called particles and to explore all the search space, each particle is evaluated by the fitness function and to manage the flying of the swarm to the prey, they use velocities module. Each particle flies around the solution by following the optimum position of particles [5][6]. All particles are associated with points in the search space and their positions are depending on their own solution and of their neighbors. Some particles come into play randomly in every iteration through this environment; they look the assessment of themselves and their neighbors [7]. Then, they follow successful particles of the given problem. PSO algorithm give satisfactory results in solving many dispatch problems related to biology medical, finance, 3d graphics, image processing and others. [8], but it is hard to choose the setting parameters because it is too complicated to find the best setting of a desired application. So, we have to set first, several parameters of the PSO algorithm: [9] [10]:

- Position and velocity equations of particles
- Number of particles in the search space
- The Gbest fitness achieved.
- Positions of particles having the best solution of all.
- Number of iteration

In the beginning we generate a random population after that we search for the best solution after each iteration. Then, the particles update their positions using two best solutions. The first one is the best solution towards the problem and it is

named « lbest ». The other optimal solution is followed by the PSO algorithm and obtained by any particles from the population and it is named « Gbest ».

A. The random number generator

Programming PSO algorithms requires the use of random generator; there are several methods to generate a random numbers. In fact it is impossible to generate a random number based on algorithms that's why they are called pseudo random number. The random generators programs are particularly suitable for implementation and effective. Most pseudo random algorithms try to produce outputs that are uniformly distributed. A common class generator uses a linear congruence. Others are inspired by the Fibonacci sequence by adding the two previous values. Most popular and fast algorithms were created in 1948 D. H. Lehmer introduced linear generators congruentiels and will eventually become extremely popular.

In our algorithm we used the bloc of the pseudo random generator [13] at the initial position of particles and in the velocity vector. We choose the frequently used pseudo-random generator called the linear congruent of Lehmer:

$$F_{n+1} = (A * F_n + B) \bmod C \quad (1)$$

Where: - F_{n+1} : is the random number obtained from the function F

- F_n : is the previous number obtained
- A and B : are multiplicative and additive value, respectively
- C : the modulo number

B. Position and velocity equations

Velocity equation allows changing the position of a desired particle and generally, the objective of using PSO algorithm is to indicate by their positions the distance to the best particle. So, these equations are updated throughout the race of iterations using the equations below:

$$V_i(t+1) = W_{moy} + c_1 r_1 \left(lbest_i - x_i(t)\right) + c_2 r_2 (Gbest - x_i)) \quad (2)$$

$$X_i(t) = X_i(t) + V_i(t) \quad (3)$$

xi(t) is the particle position at time t and vi is the velocity of particle at the instant t(i), w is parameters, c_1 and c_2 are constant coefficients, r_1 and r_2 are random numbers at each iteration, « Gbest » is the optimal solution found until now and « lbest » is the best solution found by the particle i. So, generally the velocity vector allows directing the research process and reflects the sociability of the particles.

The convergence to the optimum solution can be fixed by a number of iterations depending on the fitness or when the variation tends to zero (like sphere function) or when it tends to the best minimized solution. Here some parameters that comes into play:

- The number of population.
- The size of the neighborhood.
- The dimension of the search space.
- The values of the coefficients.
- The maximum speed.

Each iteration allows the particles to move as a function of three components:

- Its current speed
- Its local best solution
- The global best solution in its neighborhood.

TABLE I. EXAMPLE OF SOME SELECTED PARTICLES

Particles	iteration			
	1	2	3	n
$\mathbf{X_1}$	0010011111110010	0010011111110001	0010011111110000	…
$\mathbf{X_2}$	0010011111100011	0010011111100011	0010011111100011	…
$\mathbf{X_3}$	0010011111000001	0010011111000001	0010011111000001	…
$\mathbf{X_n}$	X_{n1}	X_{n2}	X_{n3}	…

In this table we present a sample from the sphere function, we can easily see that the "lbest" of particle x_1 is located in it_3 and the "lbest" of particle x_2 and x_3 are located in it_2 but the "gbest" is x_3 and located in it_2.

The global minimum for the sphere function is clearly located at $x_i = 0$, in each iteration we pick the "lbest" and we save the results into memory in order to compare its value with the new position of particles in the next iteration.

III. ARCHITECTURE OF GA AND PSO ALGORITHMS

A. GA architecture

To optimize a problem in GA, we have to explore all the searching state in order to maximize (or minimize) a chosen function. So, the use of genetic algorithm is suitable for a quick exploration of an area. The organizational chart that describes the architecture of GA is shown by the following figure.

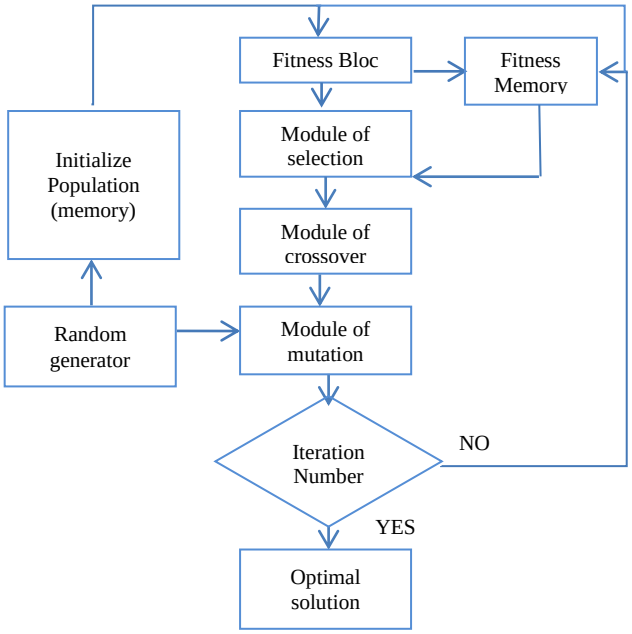

Fig. 1. Architecture Of The GA

B. PSO architecture

The architecture of our optimized PSO algorithm is presented in the following figure:

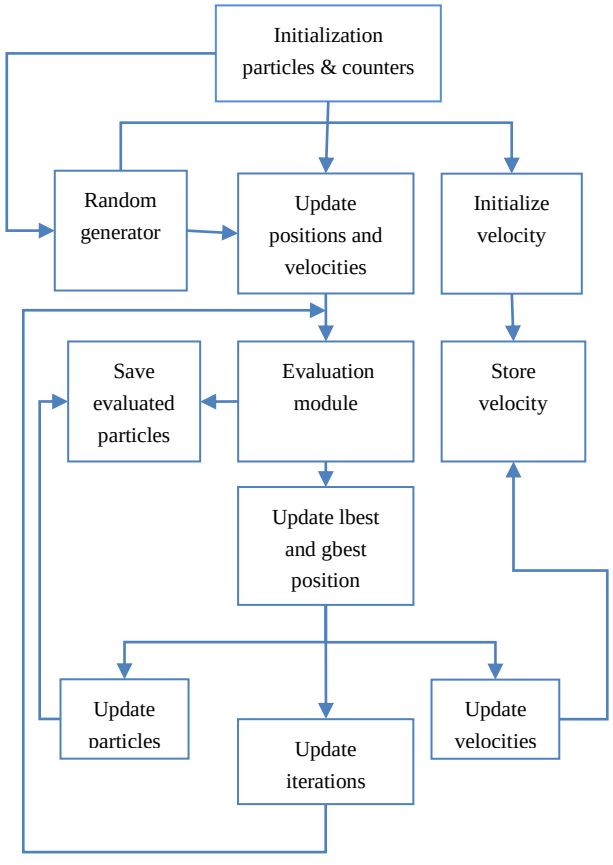

Fig. 2. Internal architecture of the PSO algorithm

For hardware implementation of PSO algorithm, the architecture is decomposed into five operations that are performed on each particle: update the position, evaluate the fitness, update the particle's best position, update the global best position and update the velocity.

We can demonstrate from the two architectures that the two algorithms share some common points. In fact the two algorithms begin with a random population in the search space and both of them use fitness module to evaluate the generation.

Both of them update the generation and search for an optimal value using the pseudo random number but the two of them does not guarantee the success. However, the Particle Swarm Optimization doesn't have crossover and mutation operators. Indeed, PSO update its particles using the velocity module.

C. The FSM

In our paper a dynamic parallel PSO is implemented to be applied into large optimization problem and compared with GA and others PSO algorithms. The FSM is used to exploit all type of parallelism to find the optimum solution in a reduced portion of times. The dynamical process of FSM is represented in figure 3, in fact every state may have at every time a position of many possible finite states. Firstly, we must propose a number of fixed states; every transition may have one or more around states. In this way, states which have only one state and have no possible transitions we named the final states.

The algorithm performs the updating of the optimum fitness number after the evaluations for all the particles. Here, when we update their positions and velocities we can obtain a good convergence rates after evaluating each particle. In a dynamic parallel computing, the main factor of performance is the communication latency after each transition between states. The goal of parallel dynamic computing is to produce optimal results even when we use multiple processors to reduce the running time. In this architecture we used pair memory modules to compound the bandwidth and thus, we can ameliorate the capabilities of our algorithm and we cannot do this only if we use Dual Channel bloc RAM. In that way we can access to the data memory in two modes write or read at the same frequency. There are problems with the dual RAM. In fact, the reading time of the content of memory is delayed by one clock comparative to the last reading. The description of the 8 states is presented in the sequel:

- S0: Initialize parameters, signals and counters of PSO algorithm and goto S1
- S1: Generate initial population and their velocities using random generator and goto S2 or S3
- S2: Save positions and velocities value into memory (RAM)
- S3: Evaluate particles using fitness module and goto S4 or S5
- S4: Save evaluated value into Bloc RAM and goto S6

- S5: Test gbest If fit(i) <Global-best(i) then update Global-best and if the number of iteration is achieved then go to final state else go to State S7
- S6: Test lbest If fit(i) < local-best(i) then update local-best(i) then, go to S2 and return to state S4
- S7: Update particles positions and velocities
- S8: Update the number of iteration if iteration not achieved then go to state S3 else go to final state (SF)
- SF: Display the optimum solution.

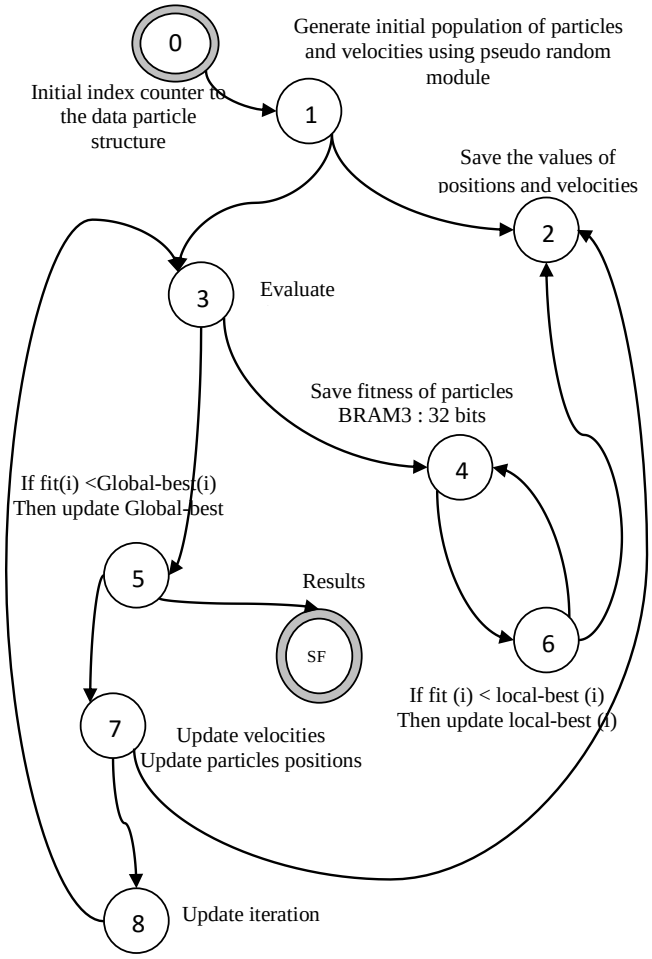

Fig. 3. The finate state machine of the PSO algorithm

Luckily, new advances in processor technology are capable and available to compute a complex program and use low cost power beyond clusters of mid-range performance computers. So, the dynamic process implemented in the particle swarm optimization could be separated in two states which update position and velocity of each particle using dynamic process with the goal to reduce the processing time.

In our paper, the soul of the parallel processing was used to generate a dynamic PSO algorithm and the aim of using parallel computing to the PSO algorithm, is to speed up the algorithm processing using a uniform distribution method to achieve optimum solutions with a significant execution time. Figure 3 present the finite state machine of the global control module; especially, it presents step by step the code of the PSO in order to keep the algorithm more practical.

D. Benchmark test functions

The Most researchers use a number of population size between 10 to 50 for the performance comparison between algorithms, here we fixed the population at 20 chromosome for the GA and the same for PSO algorithm. To test the PSO and to compare its performance with other algorithm, we used some standard benchmark functions which are described as below:

- Sphere function
- Rosen-brock function
- Rastrigin function
- Zakharov function

Some well-known benchmark functions have been selected for comparing the two implementations. So, to test and compare the performance of our proposed PSO algorithm we used unimodal and multimodal functions. These functions are described as below:

$$f_1(x) = \sum_{i=1}^{n} xi \quad (4)$$

$$f_2(x,y) = \sum_{i=1}^{n-1} \ (x_i{}^2 + y_i{}^2) \quad (5)$$

$$f_3(x,y) = \sum_{i=1}^{n-1} \ [(1-x_i)^2 + 100(y_i - x_i{}^2)^2] \quad (6)$$

$$f_4(x) = 10n - \sum_{i=n}^{n} \ (x^2{}_i - 10\cos(2\pi x_i)) \quad (7)$$

$$f_5(x) = x^2{}_i + (0.5ix_i)^2 + (0.5ix_i)^4 \quad (8)$$

IV. VALIDATION EXAMPLES

Most researchers use a number of population size between 10 to 50 for the performance comparison of GA and PSO, the swarm size used for the PSO is the same as the population size used in GA and is fixed at 20 particles in the PSO swarm and 20 chromosomes in GA population. In the GA all variables of each individual are represented with binary strings of '0' and '1' that are referred to as chromosomes. Like genetic algorithm, PSO begins with a random population and to perform its exploration, GA use three operators (crossover, selection and mutation) to propagate its population from iteration to another.

A. The sphere function

$$f_1(x) = \sum_{i=1}^{n} x_i \quad (9)$$

Sphere function is useful to evaluate the characteristics of our optimization algorithms, such as the robustness and the convergence velocity. This function has a local minimum and it is unimodal and continuous. The interval of search space is between [-1,1]. Figure 4 present the results of simulation using modelsim of the sphere function.

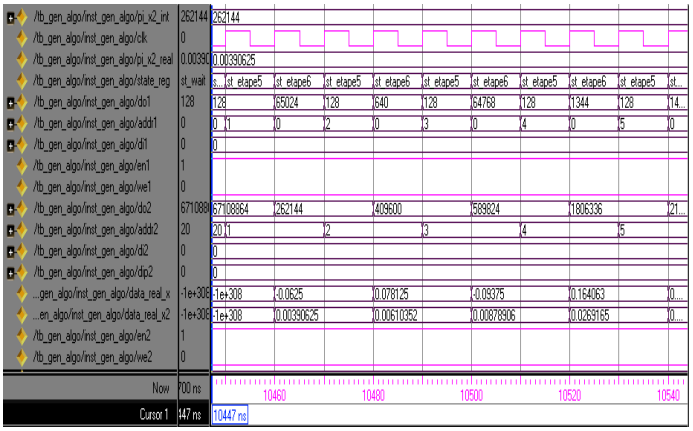

Fig. 4. Simulation results of function f1

The detailed results describe that our solution converges to zero from iteration to another.

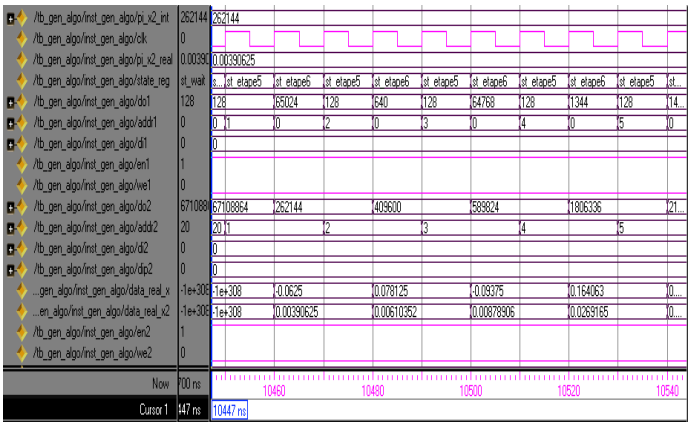

Fig. 5. Simulation results of function f2

These particles work together in a parallel dynamic state to get the best solution of any function. They update position and velocity even if the algorithm has a lot of particles and this cannot make a hard impact on the global execution time speed. Indeed, the number of particles in this algorithm is limited by the size of embedded features of FPGA. The following tables present the number of LUT (Look up Table), bloc RAM and all the resource materials used in this function.

TABLE II. DEVICE UTILIZATION SUMMARY OF PSO

(PSO) Sphere function			
logic	***used***	***available***	***utilisation***
slices	225	1920	11%
flip flops	214	3840	5%
4 inputs	354	3840	9%
IOBs	10	173	5%
BRAM	5	12	41%
multiplexers	4	12	33%
GCLKS	3	8	37%

TABLE III. DEVICE UTILIZATION SUMMARY OF GA

(GA) Sphere function			
logic	***used***	***available***	***utilisation***
slices	581	1920	30%
flip flops	600	3840	15%
4 inputs	864	3840	22%
IOBs	23	173	13%
BRAM	2	12	16%
multiplexers	8	12	66%
GCLKS	5	8	62%

In the following figure, we can easily see the difference between the two algorithms, here the PSO algorithm give better optimization in the use of hardware resources than the Genetic Algorithm.

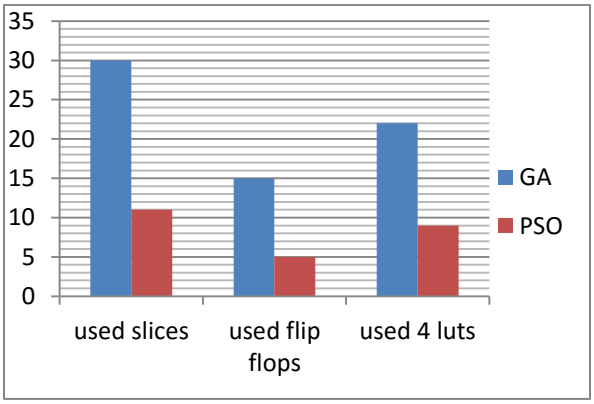

Fig. 6. Comparison of hardware resource between PSO and GA

We implemented the sphere function with two algorithms, GA and PSO using Spartan 3 from Xilinx, and then we can realize that the processing time of one iteration of PSO algorithm gives higher operation speed for optimization problems rather than genetic algorithm. The following table describes this.

TABLE IV. PROCESSING TIME OF ONE ITERATION

algorithm	***pso***	***genetic***
Execution of one Iteration (clock cycle)	1180	9740

B. The rastrigin function

This function is described below:

$$f_4(x) = 10n - \sum_{i=n}^{n} ({x_i}^2 - 10\cos(2\pi x_i)) \qquad (10)$$

The Rastrigin function contains several local minima. But it has just one global minimum and it is highly multimodal and the location of the minima is distributed regular.

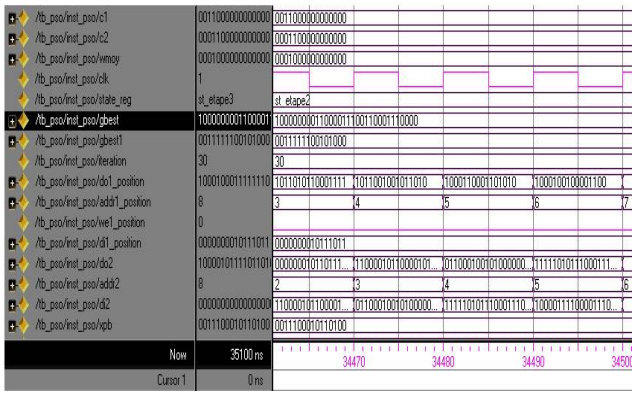

Fig. 7. Simulation results with modelsim

The synthesis results of the whole system are shown in the following tables:

TABLE V. DEVICE UTILIZATION SUMMARY OF PSO

(PSO) rastrigin function			
logic	***used***	***available***	***utilisation***
slices	307	1920	15%
flip flops	259	3840	6%
4 inputs	547	3840	14%
IOBs	26	173	15%
BRAM	10	12	83%
multiplexers	2	12	16%
GCLKS	3	8	37%

TABLE VI. DEVICE UTILIZATION SUMMARY OF GA

astrigin function			
logic	***used***	***available***	***utilisation***
slices	1265	1920	65%
flip flops	842	3840	21%
inputs	2231	3840	58%
IOBs	2	173	1%
BRAM	3	12	25%
multiplexers	4	12	33%
GCLKS	8	8	100%

We can easily see that GA require a lot of hardware resource while the PSO algorithm use less number of slice and flip flop as it shows the following figure.

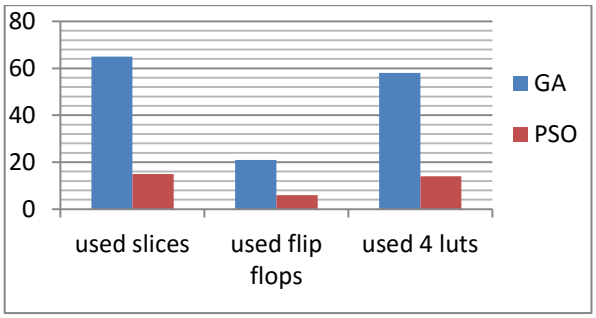

Fig. 8. a comparison of hardware resource used in the two algorithms

C. Rosenbrock function

The function of rosenbrock is a non-convex benchmark of two variables which is used to test some mathematical optimization problems. It was introduced in 1960 by Howard H. Rosenbrock and it is known by the banana function name.

In this function the global minimum of search algorithms converge easily. The function is described as follow:

$$f_3(x,y) = \sum_{i=1}^{n} [(1 - x_i)^2 + 100(y_i - x_i^2)^2] \qquad (11)$$

The global minimum is obtained at point (x, y) = (1, 1), for which the function is 0. A different coefficient is sometimes given in the second term, but that doesn't have a great affect in the position of the global minima.

D. Zakharov function

We used another benchmark which is the zakharov function whose global minimum occurs at x = (0):

$$f_5(x) = \sum_{i=n}^{n} [x_i^2 + (0.5ix_i)^2 + (0.5ix_i)^4] \qquad (12)$$

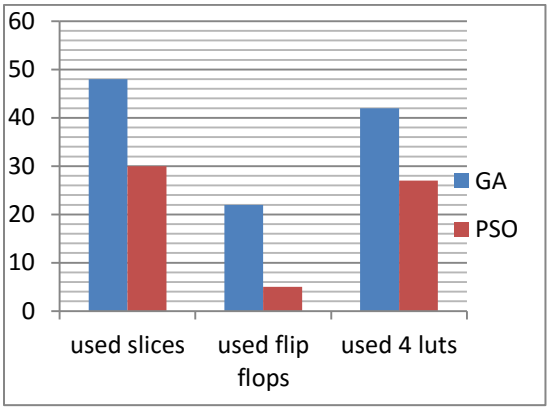

Fig. 9. Comparison between PSO and GA of rosenbrock

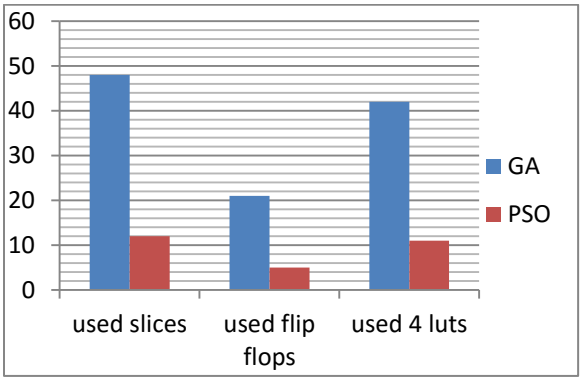

Fig. 10. Comparison between PSO and GA of zakharov function

V. EXPERIMENTAL RESULTS

The platform of Spartan-3 FPGA is from Xilinx. The Spartan3 is one of the best low cost generation of FPGAs and the board can offers a choice of many platforms which deliver a unique cost optimization balanced between programmable logic, connectivity and hardware applications. It creates a PROM file and this latter can be written to the non volatile memory of Spartan-3. The platform of Spartan3 board includes the following elements (Figure 11):

- 200k of gate in a 256-ball thin Ball Grid Array package)

- 4,320 logic cell and equivalents
- 12 x 18K of bit block RAMs (216K bits)
- 12 of hardware multipliers (18x18)
- 4 Digital extern clock (DCMs)
- A lot of I/O signals and it is up to 173
- Three "40" pin expansion connectors
- PS/2 mouse/keyboard port, VGA port and serial port.

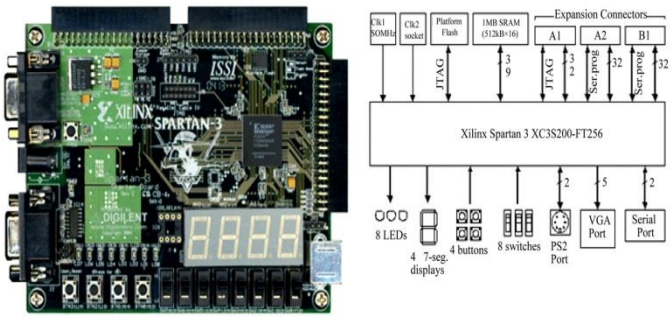

Fig. 11. The Block Diagram of SPARTAN-3

To make a comparison of this algorithm to deliver better solution in a significant time especially, its robustness and speed, we have tested it against other meta-heuristic algorithms, like genetic algorithms and another PSO algorithm. For GA, we used the basic model with elitism method and a probability of mutation equal 5%. The simulations have been carried out using spartran-3 of Xilinx with 50MHz. We have also fixed the population n = 20 for all simulations. The results are favorable and proved that Real BAT can be effective for many problems related to any algorithms used. The experiment results was carried out at minimum 5 % which allow judging whether the results of the PSO are acceptable and optimized in execution time compared to the best results of other algorithms.

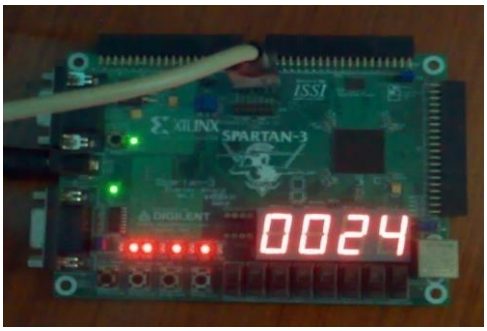

Fig. 12. Display of the number of iteration to achieve the optimal solution

VI. Related Work

Since its invention, many researchers have worked on the PSO algorithm [11] and how to accelerate its performance to give a good convergence and to reduce the use of hardware resource for embedded applications. In this section we will present some works lean on parallelization algorithms proposed by other researches. In fact, there are many interesting improvements using PSO algorithm for several applications; al.Reynolds [12] suggested a smart technique for modified PSO algorithm using neural networks. His technique is based on a deterministic approach while the particles update their positions to simplify the hardware implementation because the standard PSO algorithm has been implemented to use random generators only for the operations of update and to reduce the hardware resource Upegui and Peña [13] use a discrete recombination of PSO algorithm called (PSODR), that's allow to decrease the time of computing of the velocity module. It is clear that these modified PSO algorithm allows generating competitive results compared to those of the basic PSO algorithm [14]. Moreover another works on the PSODR algorithm are proposed by Bratton and Blackwell with simplified models of the PSODR algorithm are analyzed and proposed by Blackwell and Bratton [15] with effective results and promising.

Many researches presented a modified variant of PSO either to reduce the materials resource or to eliminate explicit problem related directly on the architecture of PSO. That's why we developed a modified architecture using finite state machine to program a parallel algorithm that could give effective results to solve several problems [16]. Thus, we fixed the representations of the data by 20 particles to bearing several purpose of applications.

A comparison performance of PSO algorithms on some processors platforms are represented in the following table. We choose two different processors platforms, the Xilinx xc3s500 [17] and the Xilinx Micro-Blaze soft processor core for the Sphere test function.

TABLE VII. Comparison of Other Platforms

Plat_form	**Xilinx xc3S500**		**Xilinx microblaze**		**Spartran xc3S200**	
	Average nb. Iter. (st.dev)	Average .Exe. time (s) (st.dev)	Average .nb. iter. (st.dev)	Average .Exe. time(s)(st.dev)	Average. nb. iter.(st.dev)	Average.Exe. time(s) (st.dev)
Sphere function	338 (30.9)	0.28 (0.03)	382 (27.0)	10.4 (0.65)	420 (88.9)	0,024 (0.001)

The random number generator plays a big role in the implementation of the two algorithms. That's why we can obtain some difference in the number of iterations even we use the same equation of random generator and the same initial seed used for the three tests. In order to evaluate the performance of our proposed PSO algorithm, we consider and compare two implementations of the PSO process: the first one is our algorithm and the second use the processor Xilinx MicroBlaze [18].

TABLE VIII. DEVICE UTILIZATION SUMMARY OF THE PSO ALGORITHM ON SPARTAN XC3S200 AND XC3S500

	Tested function (sphere)	**Number of slices**	**Block BRAM**	**MULT18x18s**
Other PSO algorithm [18]	Xilinx xc3s500	1523 (32.7%)	7 (35%)	8 (40%)
proposed algorithm	Xilinx xc3s200	225 (11%)	5 (41%)	3 (37%)

VII. CONCLUSION

In this work, we developped a hardware implementation on FPGA of a Particle Swarm Optimization algorithm. The effectiveness of our PSO algorithm has been tested on several benchmark functions for many degrees of parallelism. This architecture exploits all the parallelism to allow updating the particle positions and velocities to get a good performance of the fitness function using a finite state machine and implemented as hardware on Xilinx spartran 3 (xc3s200). In this algorithm we used a FSM to exploit all the parallelisms that make the program converge very quickly. The FSM allow updating the positions and velocities of particles and after that we can take independently the result of the better optimized fitness from the position of particles. In this paper the simulation results demonstrate that all the states and modules can be executed at the same time and the execution time can be reduced a lot.

The proposed PSO algorithm proves that it has a favorable convergence speed compared to the other meta-heuristic algorithms and the complexity of the algorithm depends on the size of design space, it means the number of allocated particles and the complexity of the problem. So, the PSO's robustness is attached to its enhanced ability to achieve a satisfaction between two requirements, the numbers of used memory and the processing time of algorithm to solve complex problems.

REFERENCES

[1] P.K. Tripathi, S. Bandyopadhyay, K.S. Pal, Multi-objective particle swarm optimization with time variant inertia and acceleration coefficients, Information Sciences 177 (22) (2007) 5033–5049.

[2] A.S. Mohais, R. Mohais, C. Ward, Earthquake classifying neural networks trained with dynamic neighborhood PSOs, in: Proceedings of the 9th Annual Conference on Genetic and, Evolutionary Computation (GECCO'07), pp. 110–117.

[3] G.S. Tewolde, D.M. Hanna, R.E. Haskell, and Multi-swarm parallel PSO: hardware implementation, in: Proceedings of the 2009 IEEE Symposium on Swarm (2009).

[4] D.H. Lehmer, Mathematical methods in large-scale computing units, Ann. Computing Lab. Harvard Univ. 141-146, 1951.

[5] R.M. Calazan, N. Nedjah, L.M. Mourelle, Parallel coprocessor for PSO, International Journal of High Performance Systems Architecture (4) (2011) 233–240.

[6] EDA Industry Working Groups, VHDL – Very High Speed Integrated Circuits Hardware Description Language, September 201

[7] N. Nedjah, L.S. Coelho, L.M. Mourelle, Multi-Objective Swarm Intelligent Systems – Theory & Experiences, vol. 261, Springer, Berlin, (2010).

[8] A. Ratnaweera, S.K. Halgamuge, H.C. Watson, Self-organizing hierarchical particle swarm optimizer with time-varying acceleration coefficients, IEEE Transaction on System, Man and Cybernetics 8 (3) (2004) 240–255.

[9] Hardware/software co-design for particle swarm optimization algorithm Shih-An Li, Chen-Chien Hsu b, Ching-Chang Wong, Chia-Jun Yu, Information Sciences 181, Elsevier (2011) 4582–4596

[10] A hardware accelerator for Particle Swarm Optimization, Rogério M. Calazan, Nadia Nedjah, Luiza M. Mourelle, Applied Soft Computing, Elsevier (2013)

[11] A hardware accelerator for Particle Swarm Optimization Rogério M. Calazan, Nadia Nedjah,Luiza M. Mourelle Applied Soft Computing xxx (2013).

[12] P.D. Reynolds, R.W. Duren, M.L. Trumbo, R.J. Marks, II, FPGA Implementation of Particle swarm optimization for inversion of large neural networks, in: Proceedings 2005 IEEE Swarm Intelligence Symposium (2005).

[13] J. Peña, A. Upegui, A population-oriented architecture for particle swarms, in: Second NASA/ESA Conference on Adaptive Hardware and Systems, (AHS 2007), pp. 563–570.

[14] J. Peña, A. Upegui, E. Sanchez, Particle swarm optimization with discrete recombination: an online optimizer for evolvable hardware, in: First NASA/ESA Conference on Adaptive Hardware and Systems, (AHS 2006), pp. 163–170.

[15] D. Bratton, T. Blackwell, Understanding particle swarms through simplification: a study of recombinant PSO, in: Proceedings of the 9th 1013 Annual Conference on Genetic and, Evolutionary Computation (GECCO'07), pp. 2621–2627.

[16] Y. Maeda, N. Matsushita, Simultaneous perturbation particle swarm 1035optimization using FPGA, in: Proceedings of International Joint Conference 1036 on Neural Networks (IJCNN 2007), pp. 2695–2700.

[17] G.S. Tewolde, D.M. Hanna, R.E. Haskell, Multi-swarm parallel PSO: hardware 1054 implementation, in: Proceedings of the 2009 IEEE Symposium on Swarm Intelligence (SIS09), Nashville, TN, pp. 60–66.

[18] A modular and efficient hardware architecture for particle swarm, optimization algorithm Girma S. Tewolde Q1, Darrin M. Hanna, Richard E. Haskell Microprocessors and Microsystems xxx, Elsevier (2012).