# High-Speed FPGA-based of the Particle Swarm Optimization using HLS Tool

Ali Al Bataineh[1], Devinder Kaur[3]
Department of Electrical Engineering and Computer Science
University of Toledo Ohio
USA

Amin Jarrah[2]
Department of Computer Engineering Hijjawi
Faculty for Engineering Technology
Yarmouk University, Irbid, Jordan

***Abstract*—The Particle Swarm Optimization (PSO) is a heuristic search method inspired by different biological populations on their swarming or collaborative behavior. This novel work has implemented PSO for the Travelling Salesman Problem (TSP) in high-level synthesis to reduce the computational time latency. The high-level synthesis design generates an estimation of the hardware resources needed to implement the PSO algorithm for TSP on FPGA. The targeted FPGA of this algorithm is the Xilinx Zynq family. The algorithm has been implemented for getting the best route between 5 given cities with given distances. The research has used 7 number of particles for a different number of iterations for generating the best route between those 5 cities. The overall latency has been reduced due to the applied optimization techniques. This paper also implemented and parallelized the same algorithm on CPU Intel I7 Processor; the result shows the FPGA implementation gives better results than CPU on the comparison of performance.**

***Keywords*—FPGA; High Level Synthesis; Particle Swarm Optimization; Travelling Salesman Problem (TSP)***

## I. INTRODUCTION

Particle Swarm Optimization (PSO) is an algorithm which is been adapted from the unpredictable flight of the bird's flock. This algorithm was proposed by Eberhart and Kennedy in 1995. This algorithm is also called as population-based stochastic optimization technique [1]. It uses a model inspired from flying birds in the flock or flock of fish. In the flock, bird or fish (called particle) will search and identify the whole space guided by both its previous best position (pbest) and the best position of the swarm (gbest) or global best position [2]. PSO is applied for multiple fields including scheduling applications, for finding best routes or planning routes and the optimization problems [3-4].

Qang et al. [3] have implemented PSO for job scheduling application. They encoded each particle with a natural number vector and have developed an own approach to move particles in the solution space. They also compared the genetic algorithm (GA) with the PSO for job scheduling application and they found that PSO is very competitive with the GA.

The PSO algorithm with simulated annealing is implemented for optimization of the TSP problem is done by fang et al. [4]. This implementation uses simulated annealing (SA) method for slow down the degeneration of the swarm and increase the swarm diversity. They compared the PSO with SA, basic Genetic Algorithm (GA) and two other algorithms for solving TSP problem in which the PSO with SA gives the superior results than the other methods.

The embedded method of PSO and the SA gives the faster solution than the PSO method in the small and medium-size problem. The SA algorithm is capable to search on a subspace of the whole search space by means of individual particle that result in faster solution and accurate [5].

Hybrid PSO with SA gives the better performance than the adaptive particle swarm optimization and the genetic chaos optimization algorithm [6]. The combination of PSO and SA can narrow down and speed up the field of the search. This strength of the PSO with SA can also optimize the TSP problem with the better search result and speed up.

Hassan et al. [7] have done the comparison of Genetic Algorithm (GA) and the PSO in terms of its effectiveness for finding the global optimal solution and computational efficiency. This research has done the comparison by implementing statistical analysis and formal hypothesis testing.

The goal of this research is to develop and optimize the Travelling Salesman Problem (TSP) on FPGA using High Level Synthesis. Different optimization techniques will be applied such as loop unrolling, loop pipelining, dataflow, loop merging and others. This will help in finding the best route in a high speed.

## II. TRAVELLING SALESMAN PROBLEM

Travelling Salesman Problem (TSP) is a problem of finding the best route for traveling between multiple cities. In the TSP, one salesman wants to visit n cities, the objective of TSP is to identify the shortest Hamilton cycle through which the salesman can visit each city only once and finally return to the starting position or city. The TSP problem is solved using different algorithms as Ant Colony Optimization, Genetic Algorithms, Neural Network, and others [1].

For solving the TSP, the Ant System (AS) [8] and the Particle Swarm Optimization (PSO) [9] is the preferred method due to its optimized solution for the TSP problem. The first implementation of PSO for solving TSP is done by Maurice Clerc in 2000 [9]. At that implementation, results show that PSO was feasible but not very efficient for solving the TSP PSO and AS are implemented for TSP problem for the surveillance mission by Barry R. Secrest [10]. The work is on the planning the best route for the surveillance mission with different types of aircraft. This implementation is targeted for the Mission Route Planning (MRP) for the Unmanned Air Vehicle (UAV) [10].

General description of TSP can be done as: particles have to identify the shortest path that covers all cities along. Let G=(V;E) be a graph where V is a set of vertices and E is a set of edges. Let C=(cij), which is the distance or cost matrix associated with E. The particles need to identify the minimum cost path or Hamilton cycle between the cities [11].

## III. Particle Swarm Optimization

PSO algorithm is inspired from flocks of birds, schools of fish and herds of animals to adapt their environment, find rich source of food and secure themselves from predators by the information sharing approach. Therefore, the PSO algorithm mimics the social behavior of natural organisms, which consists of action of individual member and the effect of other individuals within the group [7]. Each particle in PSO is considered [12],

- to have specific position and a velocity;
- to knows its own position and the value associates with it;
- to knows the best position it has ever achieved, and the value associated with it; and
- knows its neighbors, their best positions and their values.

PSO algorithm gives high performance for different search and pathfinding problems. Therefore, it has been implemented for solving and optimized a wide range of problems. For the optimization on a solution with PSO, the computation cost and the precision are considered as the main variable.

PSO algorithm use Eq. (1) and (2) to calculate the new velocity and position at each iteration:

$$v_i(t+1) = w\, v_i(t) + c_1(p_i(t) - x_i(t)) + c_2\,(g(t) - x_i(t)) \quad (1)$$

where:

w: real value coefficient (inertia).

$c_1$, c2: real value coefficients (the personal influence and the global influence factors).

$v_i(t)$: current velocity of particle i.

$v_i(t+1)$: next velocity of particle i.

$x_i(t)$: current position of particle i.

$p_i(t)$: personal best known position of particle i.

g(t): global best known position of the whole swarm.

$$x_i\,(t+1) = x_i\,(t) + v_i(t+1) \quad (2)$$

where:

$x_i(t)$ : current position of particle i.

$x_i(t+1)$ : next position of particle i.

$v_i(t+1)$ : next velocity of particle i.

The pseudocode of PSO is showed as follows [2],

```
Initializing the whole swarm randomly
        $for(i = 0; i < swarm\ size; i++)$
Evaluate $f(x_i)$
  $while(termination\ criteria\ is\ not\ atisfied)\{$
$for(i = 0; i < swarm\ size; i++)\{$
        $if(f(x_i) > f(pbest_i))\ pbest_i = x_i;$
        $if(f(x_i) > f(gbest_i))\ pgest_i = x_i;$
Update $(x_i, v_i)$
Evaluate $f(x_i)\}\}$
```

$where\ f(x_i)$ is the fitness function for estimating the quality of the solution. $pbest$ is the particle local best position and the $gbest$ is the global best position of the entire flock. The update step is performed by Eq. 3.

$$v_{id}^{k+1} = w * v_{id}^{k} + c_1 rand(\,) * \left(pbest_{id}^{k} - x_{id}^{k}\right) + c_2 rand(\,) * (gbest_{d}^{k} - x_{id}^{k}) \quad (3)$$

where $i$ is the number of particle and $d$ is the number of dimensions, $w$ is inertia weight and it decide how much the pre-velocity will affect the new one. $c1$ and $c2$ are constant values, which are also called as learning factors. These constant values decide the degree of affection of $pbest$ and $gbest$. $rand()$ denotes to a random number between 0 and 1 [2].

The particles fly towards a new position based on Eq. (4). PSO algorithm is implanted for a certain number of iteration until the stopping criteria will give the solution that lies in the global best.

$$x_{id}^{k+1} = x_{id}^{k} + v_{id}^{k+1} \quad (4)$$

## IV. High Level Synthesis

High Level Synthesis (HLS) is the methodology of implementing algorithm on high-level language and targeting the algorithm on the hardware or called Field Programmable Gate Array (FPGA). High-level synthesis program allows writing the algorithms on the high-level language as C/C++ and OpenCL. HLS tool converts the algorithm from this high-level language (C/C++/OpenCL) to the Hardware Description Language (HDL) level [13].

Xilinx has a high-level synthesis tool called VIVADO HLS, which synthesize and converts algorithm written in C/C++/OpenCL into VHDL/Verilog and System C [13]. VIVADO HLS has a number of built-in functions and libraries for video processing, math, linear algebra, digital signal processing and IP (Intellectual Property) Design [13].

Fig. 1 shows the HLS design flow, with the VIVADO HLS we can write out the algorithm on C, C++ or System C or OpenCL which will converted by HLS into VHDL/Verilog/System C or IP format. The HLS tool can also export the design into other formats also which include System Generator, P-Core or XPS format [13].

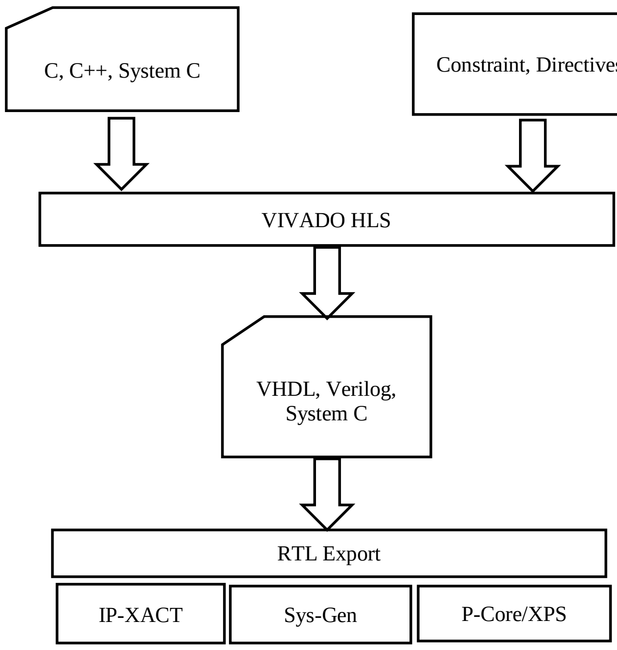

Fig. 1. High Level Synthesis Design Flow.

In HLS, there are many optimization strategies for optimizing the latency and resources. One of the main strategies for optimization of latency and resource consumption is pragma directives. These pragma directives instruct the compiler for performing the specific operation while compilation [14]. In this PSO implementation, different optimization techniques are incorporated such as loop pipelining, loop unrolling, dataflow, loop merging and others.

## V. Hardware Platform

Our PSO implementation is done for targeting the Xilinx FPGA hardware called ZedBoard which is Zynq 7000 family of FPGA. This ZedBoard has xc7z020clg484-1 FPGA device. The Zynq 7000 architecture consists of Processing System (PS), which is programmable dual-core ARM Cortex A9 Processor and Programmable Logic (PL), which is the 7 series Xilinx FPGA Core with resources as LUT, FIFO, BRAM, DSP and IO’s [15]. This Zynq has following resources on the PL section, 53200 logic implementable block called, look up table (LUT), 106400 Flip-Flop (FF), 220 DSP blocks and 280 Block RAM [16].

## VI. PSO Analysis and Optimization

The goal of this research is on solving the simple traveling salesman problem (TSP) with the PSO in High Level Synthesis, HLS allows writing an algorithm on C/C++ or OpenCL language. For the TSP problem, there is predefined number of cities with the predefined distance between those cities. The PSO algorithm is used to solve the TSP problem to find the shortest path between cities so that each city must visited only once. For the calculation of the shortest path between the cities, we have taken the 5 number of cities, which is represented in Fig. 2.

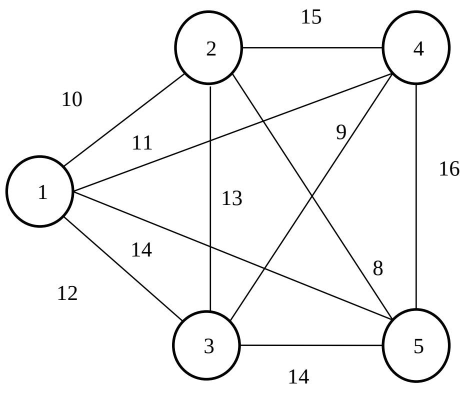

Fig. 2. Traveling Salesman Problem with 5 Cities and Distances.

For the random number generation on HLS Catalin Baetoniu [17] has stated the one of the best methods that can generate the true random number in high speed in Xilinx FPGA. Linear Feedback Shift Register (LFSR) is another method of random number generation that uses the shift register to take input as the function from the previous shift register. LFSR method has least feedback than the counters, so it can also be implemented as the fast counter. Eq. (5) represents the mod-2 polynomial function as the feedback to the input to the LFSR [18]:

$$input\ bit = x^3 + x^2 + 1 \tag{5}$$

LFSR method is used in our research for generating random number; this random number is used by the particles for selection of cities from the given 5 cities.

The PSO algorithm is used for getting the best and shortest path between cities for the Traveling Salesman Problem (TSP). The PSO algorithm can be represented on flowchart, which is depicted in Fig. 3.

We use pipelining technique in the initialization stage, which will help in concurrent operations inside the loop and so increasing the throughput of executing the algorithm and reducing the latency.

At the beginning, the personal best position and fitness of the particle are the current position and fitness of it. Then, the personal best position of this particle is determined by comparing the previous position with the current position. The global best position depends on the function of fitness we choose. If this function aims to find the minimum value and choosing it to be the best choice, then the minimum value of personal best array will be the global best one. The pseudo code of this calculation is shown as follows:

```
for i = 1 to number_of_particles do
    if fitness [i] ≤ personal_best [i] then
    update personal_best_position [i]
    if personal_best [i] ≤ global_best [i]
    then
    update global_best_position [i]
    repeat
```

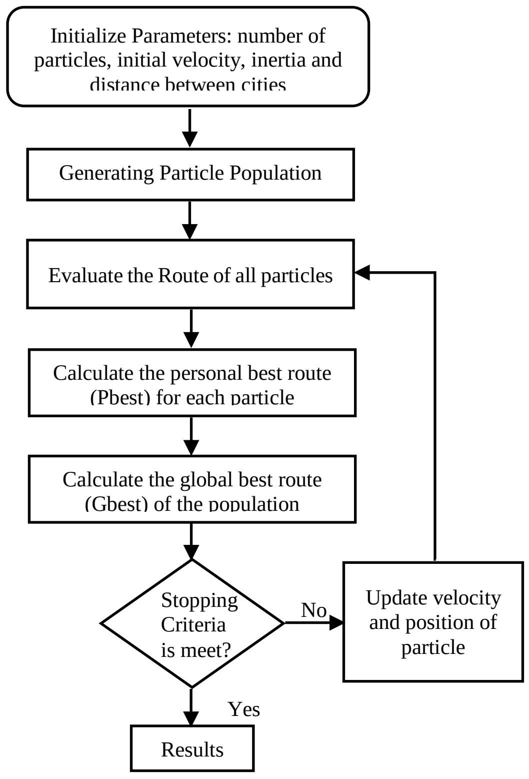
Initialize Parameters: number of particles, initial velocity, inertia and distance between cities
Generating Particle Population
Evaluate the Route of all particles
Calculate the personal best route (Pbest) for each particle
Calculate the global best route (Gbest) of the population
Stopping Criteria is meet?
No
Update velocity and position of particle
Yes
Results

Fig. 3. Algorithm of Particle Swarm Optimization.

We use pipelining and loop unrolling which will result in creating multiple independent operations. This technique will reduce the latency and improve the throughput of the algorithm. In addition, we use pipelining technique in this stage.

Position and velocity of the particle will be updated every iteration. The new values will become the present position and present velocity in the next iteration. The pseudo code for this updating is shown as follows:

```
**for** i = 1 to number_of_particles **do**
**update** velocity and position of particle i
**until** stopping criterion
```

In the global best update, we use pipelining technique in inner loop, which led to increase throughput and decrease execution time. We use pipelining and loop unrolling techniques in this loop, which led to enhance the final results.

The mathematical model of particles motion described in Eq. (1) and (2) forms the velocity equation by sum of three parts. The first part is parallel to the previous velocity and equals to *WVi(t)*; called Inertia component. The second part is parallel to the vector connecting Xi to Pi and equal to *C1(Pi(t)-Xi(t))*; called Cognitive Component. The third part is parallel to the vector connecting Xi to G(t) and equal to *C2(G(t)-Xi(t))*; called Social Component. The three parts of Eq. (1) combined to update velocity vector to new one and this update will cause the particle position to be updated to the new position in the problem search space as in Eq. (2). The main stages in PSO algorithm are problem definition, initialization, and core work of PSO.

Problem Definition: in this stage the optimization problem will be define to solve it by PSO and we define a function of X which return the solved values of the problem, which called the cost value. We also define the numbers of variables that exit in the problem, and the ranges of these variables must be determined. The size of matrix that that will be used of these variables should also be defined, and the parameters of PSO like number of iteration, swarm size and the values of *W*, *C1* and *C2* determine at this stage.

Initialization: In this stage, a group of steps must be taken place to start PSO like create the initialization positions of particles. Then, evaluate them and initialize personal based and global based values of particles, and other initialization tasks needed to run PSO in right ways as shown in Fig. 4.

In this stage the information and data for all particles stored in array of structures. Every swarm particle represented by one structure, all needed field for these particles must be stored in these structures, like positions, velocities, cost values, and personal best position. The particles positions with random values and with zero's velocity values should be initialized. Then, we evaluated the cost values of the particles with its positions and saved it in cost fields. After that, we update the personal best position and save it, the global best initialize firstly to the some value that far from the request solution then it replaced by the best particle personal value.

PSO core work: this stage represents the main loop or work of the PSO algorithm as shown in Fig. 5. For every iteration, the particles new velocities using Eq. (1) is calculated and its new positions using Eq. (2) and update its cost values. A comparison of the current particles position and cost values to its personal best ones that stored in memory is also performed. if the new values better than the personal best values we update the personal best values with the new current values, after that we compare the particle best values with the global best values and replace the global best value with the personal best value if it better than it.

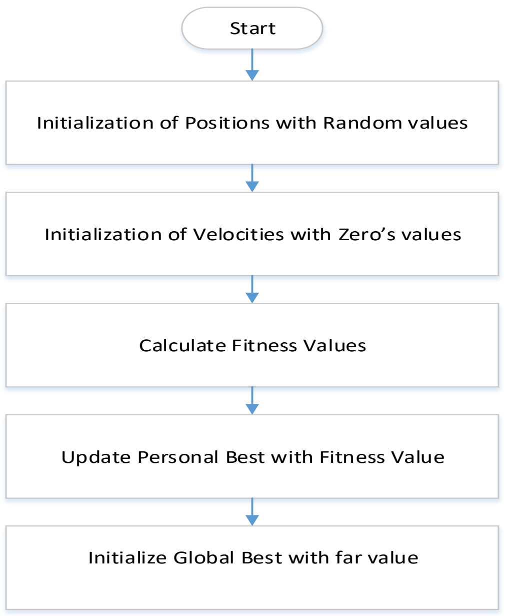
Start
Initialization of Positions with Random values
Initialization of Velocities with Zero's values
Calculate Fitness Values
Update Personal Best with Fitness Value
Initialize Global Best with far value

Fig. 4. Initialization of the PSO Algorithm.

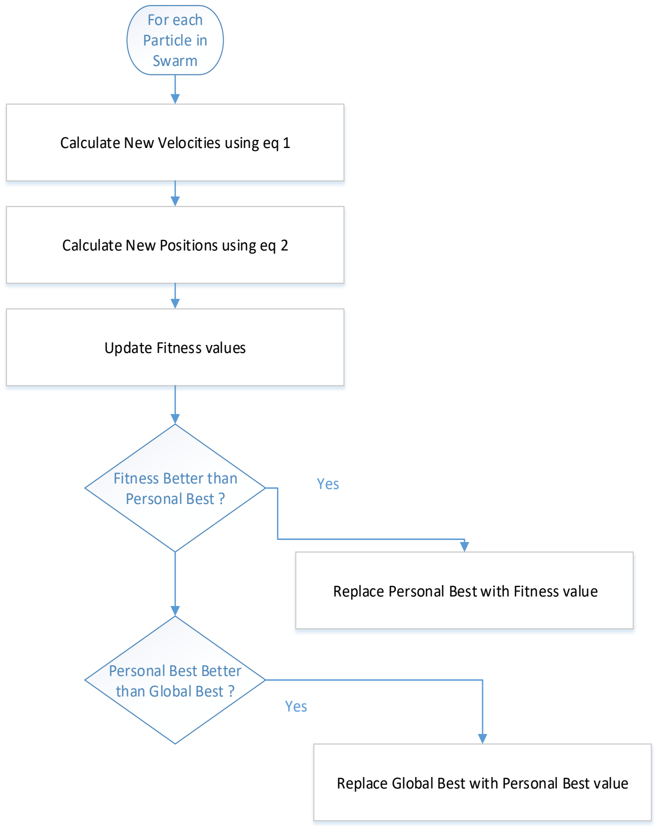
For each Particle in Swarm
Calculate New Velocities using eq 1
Calculate New Positions using eq 2
Update Fitness values
Fitness Better than Personal Best ?
Yes
Replace Personal Best with Fitness value
Personal Best Better than Global Best ?
Yes
Replace Global Best with Personal Best value

Fig. 5. Core Work of the PSO Algorithm.

## VII. Results

We have implemented the PSO algorithm for TSP problem in C language in the VIVADO HLS tool. We also have implemented the same algorithm on the Code:Block Compiler. The design is simulated and tested on the HLS as well as on Code Block environment. On the testing of the PSO for TSP with 5 cities with defined distances is explained above. We have run the test for iteration =10 with the number of particles=7. The best path identified for the TSP is 1-2-5-3-4-1 with the total distance of 52. This best path in terms of distance is shown in following Fig. 6.

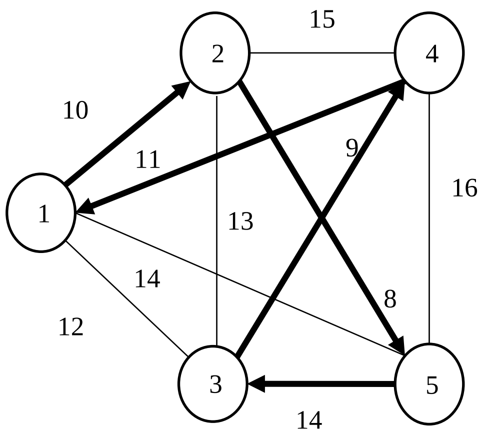

Fig. 6. Minimum Cost Path Obtained from the PSO Algorithm, Path 1-2-5-3-4-1.

We have compared the latency and resource utilization of the PSO for TSP with different scenarios. This PSO for TSP on VIVADO HLS algorithm is targeted for the Xilinx Zynq FPGA having the FPGA device of xc7z020clg484-1.

For the optimization of latency and resource, we have implemented the pragma directives on the HLS. We have tested the PSO for TSP algorithm by changing the number of iterations and placing the number of particles fixed as 7.

Table I is the resource utilization table while implementing the PSO for TSP on VIVADO HLS. The resource utilization of the implementation is shown in number and percentage. The implementation consumes 6% of BRAM_18K, 12% of DSP48E, 9% of FF and 35% of LUT of the targeted Zynq device.

Table II shows the comparison of latency with respect to the number of iteration and particles while implementing the PSO for TSP on HLS. This comparison is without the optimization with a pragma. In this table, the number of particles is placed constant and the number of iterations is varied from 10 to 100.

The latency of implementation with the number of iteration -10 and particles=7 is smallest then the other implementation. While increasing the iteration as the usual the latency has increased.

Table III shows the latency of the implementation with the utilization of pragma directives for the optimization. This optimization shows the latency of implementation with a different number of iteration and the constant number of particles. While comparing Table II and III, the optimization methodology reduces the overall latency than the without optimization. While the number of iterations =100 and the number of particles=7, the latency with optimization is 1.8 times less than the without optimization on HLS implementation.

TABLE I. Resource Utilization of PSO for TSP with Optimization and Float Data Type at Iteration=10 and Number of Particles=7

| Resources | Available Resources | Utilization by float data type | Utilization by float datatype (%) |
|---|---|---|---|
| BRAM_18K | 280 | 19 | 6 |
| DSP48E | 220 | 28 | 12 |
| FF | 106400 | 10026 | 9 |
| LUT | 53200 | 18937 | 35 |

TABLE II. Comparison of Maximum Latency with Respect to the Different Number of Particles and Iteration with Targeted Clock=10ns and without Optimization

| Iteration No.(i) | Number of Particles (a) | Latency (max.) | Time (in Second) |
|---|---|---|---|
| 10 | 7 | 844124 | 0.00844124 |
| 20 | 7 | 960584 | 0.00960584 |
| 30 | 7 | 1077044 | 0.01077044 |
| 40 | 7 | 1193504 | 0.01193504 |
| 50 | 7 | 1309964 | 0.01309964 |
| 100 | 7 | 1892264 | 0.01892264 |

TABLE III. COMPARISON OF MAXIMUM LATENCY WITH RESPECT TO THE DIFFERENT NUMBER OF PARTICLES AND ITERATION WITH TARGETED CLOCK=10NS AND WITH OPTIMIZATION

| Iteration No.(i) | Number of Particles (a) | Latency (max.) | Time (in Second) |
|---|---|---|---|
| 10 | 7 | 792924 | 0.00782924 |
| 20 | 7 | 808584 | 0.00808584 |
| 30 | 7 | 834244 | 0.00834244 |
| 40 | 7 | 859904 | 0.00859904 |
| 50 | 7 | 885564 | 0.00885564 |
| 100 | 7 | 1013864 | 0.01013864 |

Table IV is the table for resource utilization of resources with the pragma directives for the optimization of resources and latency. The BRAM and DSP is constant while varying the number of iteration and making fixed the number of particles. While the number of FF and LUT has increased respectively when increasing the number of iterations from 10 to 100. The resource utilization is increased because of while the number of iteration is increased, the number of FF and LUT needed for processing arraySubtraction_float(), arrayAddition_float() and multiplyArrayWithScalar().

Table V shows the latency and the best path identified by the PSO for TSP on the Intel 6700HQ Processor and Code::Block Compiler. This Intel x86 processor has 3.50 GHz of frequency, 4 cores and 16GB RAM with Windows 10 Operating System.

TABLE IV. RESOURCE UTILIZATION REPORT WITH RESPECT TO THE NUMBER OF ITERATION AND PARTICLES AND WITH OPTIMIZATION

| Iteration No.(i) | Number of Particles (a) | BRAM_18K | DSP48E | FF | LUT |
|---|---|---|---|---|---|
| 10 | 7 | 19 | 28 | 10026 | 18937 |
| 20 | 7 | 19 | 28 | 10496 | 21190 |
| 30 | 7 | 19 | 28 | 10966 | 23300 |
| 40 | 7 | 19 | 28 | 11436 | 24826 |
| 50 | 7 | 19 | 28 | 11906 | 26296 |
| 100 | 7 | 19 | 28 | 14256 | 33790 |

TABLE V. RESULT WHILE RUNNING PSO WITH DIFFERENT NUMBER OF ITERATION AND PARTICLES ON X86 PC [ INTEL I7 6700HQ PROCESSOR

| Iteration No.(i) | Number of Particles (a) | Identified best route | Minimum cost-based distance | Total time spent (sec) |
|---|---|---|---|---|
| 10 | 7 | 0-1-4-2-3-0 | 52 | 0.001 |
| 20 | 7 | 0-1-4-2-3-0 | 52 | 0.001 |
| 30 | 7 | 0-1-4-2-3-0 | 52 | 0.002 |
| 40 | 7 | 0-1-4-2-3-0 | 52 | 0.002 |
| 50 | 7 | 0-1-4-2-3-0 | 52 | 0.002 |
| 100 | 7 | 0-1-4-2-3-0 | 52 | 0.002 |

We have exported the HLS implementation of PSO as the IP format to the VIVADO program. Then we have integrated our PSO IP with other necessary blocks for implementing on the Zynq FPGA board. The interconnection of PSO IP with other blocks is done for instructing the PSO IP with the number of iteration and number of particles from the Zynq Processing System. From this design, Zynq PS can accept the instruction of number of iteration and particles from the UART terminal and then the PS configures the information to the PSO IP. The PSO IP run the information of iteration and number of particles and then reply back the result to the PS.

## VIII. CONCLUSION

In this paper, the Particle Swarm Optimization algorithm has been implemented on the HLS methodology. This work has implemented PSO for traveling salesman problem (TSP) in the C programming language. The number of cities is 5 and the number of iteration and particles are varied. The HLS algorithm also been optimized with the pragma directives. While optimizing the algorithm the number of resources needed has been increased because of the latency optimization, we used the LOOP_FLATTEN, LOOP_UNROLL and PIPELINE pragma directives. Moreover, the VIVADO block design has been implemented and the processor configuration is implemented.

## REFERENCES

[1] L. Diosan and M. Oltean, "Evolving the Structure of the Particle Swarm Optimization Algorithms," in Evolutionary Computation in Combinatorial Optimization-EvoCOP, Lecture Notes in Computer Science-LNCS, Heidelberg, Springer-Verlag, 2006, pp. 25-36.

[2] W.-h. Zhong, J. Zhang and W.-n. Chen, "A novel discrete particle swarm optimization to solve traveling salesman problem," presented at IEEE Congress on Evolutionary Computation, Singapore, September 25-28, 2007.

[3] Q. Kang, H. He, H. Wang and C. Jiang, "A Novel Discrete Particle Swarm Optimization Algorithm for Job Scheduling in Grids," at Fourth International Conference on Natural Computation, Jinan, China, October 18-20, 2008.

[4] L. Fang, P. Chen and S. Liu, "Particle swarm optimization with simulated annealing for TSP," proceeding of International Conference on Artificial Intelligence, Knowledge Engineering and Data Bases, Corfu Island, pp 206-210, vol.6, 2007.

[5] G.H. Shakuri , K. Shojaee and H. Zahedi, "An Effective Particle Swarm Optimization Algorithm Embedded in SA to solve the Traveling Salesman Problem," proceeding of Chinese Control and Decision Conference (CCDC 2009), pp 5581-5586, 2009.

[6] X.-H. Wanf and J.-J. Li, "Hybrid Particle Swarm Optimization with Simulated Annealing," Proceedings of the Third International Conference on Machine Learning and Cybernetics, Shanghai, pp. 2402-2405, vol. 3, 2004

[7] R. Hassan, B. Cohanim, O. d. Weck and G. Venter, "A Comparision of Particle Swarm Optimization and the Genetic Algorithm," American Institute of Aeronautics and Astronautics, Austin, TX, April 18-21, 2004.

[8] M. Dorigo and T. Stützle, Ant Colony Optimization, 1st ed. London, U.K: MIT Press, 2004, ch.3, pp.65-81.

[9] M. Clerc, Particle Swarm Optimization, 1st ed. London, U.K: ISTE, 2006, pp.172.

[10] B. R. Secrest, "Travelling Salesman Problem for Surveillance Mission using Particle Swarm Optimization", M.S Thesis, Dept. of the air force, Air Force Institute of Technology, Air University, Ohio, 2001.

[11] X. Yan, C. Zhang, W. Luo, W. Li, W. Chen and H. Liu, "Solve Traveling Salesman Problem Using Particle Swarm Optimization Algorithm," International Journal of Computer Science, vol. 9, no. 6, pp. 264-271, 2012.

[12] E. F. G. Goldbarg, G. R. de Souza and M. C. Goldbarg, "Particle Swarm for the Traveling Salesman Problem," EvoCOP 2006: Evolutionary Computation in Combinatorial Optimization, vol. 3906, pp. 99-110, April, 2006.

[13] Vivado Design Suite (High Level Synthesis)-UG902, Xilinx, Inc., 2017, pp. 5-14.

[14] Vivado HLS Optimization Methodology Guide-UG1270, Xilinx, Inc., 2017, pp. 9-32.

[15] Zynq-7000 All Programmable SoC, Technical Reference Manual (UG585), Xilinx, Inc., 2017, pp.26-40.

[16] ZedBoard hardware user guide, Avnet, Inc.,2012, pp. 3-29.

[17] C. Baetoniu, "High Speed True Random Number Generators in Xilinx FPGAs," Xilinx, Inc., San Jose, CA, 2004.

[18] Mentor Graphics Corporation, High-Level Synthesis-Blue Book, Mentor Graphics Corporation, 2010, pp. 142-143.