Received August 25, 2019, accepted September 9, 2019, date of publication September 18, 2019, date of current version September 30, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2942205

FPGA Implementation of Floating Point Based Cuckoo Search Algorithm

HANADY HUSSEIN ISSA, (Member, IEEE), AND SALEH MOHAMED EISA AHMED, (Member, IEEE)
Department of Electronics and Communications, Arab Academy for Science Technology and Maritime Transport (AASTMT), Cairo 11799, Egypt
Corresponding author: Hanady Hussein Issa (hanady.issa@aast.edu)

ABSTRACT Cuckoo search algorithm (CSA) has been a candidate for numerous recent applications and showed great compatibility in solving optimization problems. It is a metaheuristic algorithm which is based on the odd breeding strategy of the Cuckoo bird spices. It is used to find an optimum or near optimum solution for a certain problem. In this research, we propose an FPGA hardware implementation for the CSA based on single precision IEEE floating point (FP). The FP format provides a wider range and higher precision when compared to fixed point format. To the best of our knowledge, this is the first study to consider implementing FP format-based CSA on FPGA. The proposed design is implemented using pipelined and parallel techniques to get a high throughput and speed. The design is controlled and coordinated using finite state machines (FSMs) modules and is configured on Cyclone IV E FPGA chip from Intel. Three common benchmark functions are used to evaluate the performance of the proposed design. The design has a maximum operating frequency of 99 MHz. It was found out the maximum power consumption for the most complex function is 610.28 mW, mainly due to the use of FP format. In addition, the proposed design is implementation and evaluated for multidimensional operation. Accordingly, the proposed design is suitable for path planning for unmanned aerial vehicles (UAVs), sensor deployments for wireless sensor networks (WSNs) in addition to medical diagnostic and DSP applications.

INDEX TERMS Cuckoo search algorithm (CSA), field programmable gate array (FPGA), finite state machine (FSM), IEEE floating point format.

I. INTRODUCTION

Optimization algorithm (OA) is a procedure or set of instructions that is used to find an optimum solution for a given problem. According to [1], OAs can be divided into two categories, heuristic and metaheuristic algorithms. The heuristic algorithms are problem specific and cannot be applied to any other problems while metaheuristic algorithms are more general and can be applied to a wide range of problems. The nature inspired or bio metaheuristic optimization algorithms imitate the techniques found in nature to find the best solution. They are categorized as follows: evolutionary algorithms (EAs), swarm-based algorithms, and trajectory-based algorithms.

One of the most common metaheuristic optimization algorithms is the Cuckoo search algorithm (CSA) which is the main focus of this research. It belongs to the swarm-based algorithms and was first introduced by Yang and Deb [2]. The CSA imitates the strange breeding behavior of the Cuckoo bird species. The Cuckoo birds search for a random host nest with recently laid eggs, then they lay their eggs in the nest of the host bird. The Cuckoos have evolved to carefully mimic the color and patterns of the host's bird eggs. If the host bird discovers the cuckoo's eggs, it either gets rid of the imposter's eggs or simply just leaves the nest. If the eggs are not discovered, the Cuckoos eggs hatch earlier than that of the host and the hatchlings get rid of the host eggs immediately. This kind of action increases their chance for survival and hence the re- productivity of the Cuckoos bird species [3]–[5].

According to the statistical analysis performed in [6], [7], CSA outperforms other swarm-based algorithms such as partial swarm (PSA) and artificial bee colony algorithms (ABC). CSA has the advantage of convergence to the true global optimum. For example, in PSA all possible solutions are crowded around the current solution and thus PSA converges prematurely and the global minimum cannot be found. On the other hand, CSA has the feature of local and global search

The associate editor coordinating the review of this manuscript and approving it for publication was Alex James.

which ensures that all the solution space is explored. The local search improves the current best solution while the global search ensures the diversity of the population which is achieved through the use of random walk. Another study performed in [8] showed that the CSA outperforms other swarm-based algorithms in terms of problem solving. In addition, the CSA is computationally more efficient than PSA and GA as shown in [9], [10].

The applications of the CSA are diverse and can be applied in problem solving and design optimization [1], [11]. For example, the maximum power tracking of photovoltaic systems using CSA is investigated in [12], [13]. The results have shown that the CSA surpasses other optimization techniques such as perturb and observe (P&O) and particle swarm optimization (PSO). Likewise, the design of reliable embedded systems using the CSA as a multi-objective optimization can also be found in [14]. Engineering Structural design optimization and structural damage identification using CSA are investigated in [15], [16]. The CSA is also used in signal processing to design a stable higher order infinite impulse response (IIR) filters such as low pass filter (LPF) and high pass filter (HPF) as indicated in [17]. The results show that the design is computationally more efficient than other optimization algorithms. In image processing, the CSA with levy flight is used to increase the computational efficiency and implementation of multilevel thresholding techniques used for color image segmentation [18]. CS is successfully utilized in multilevel image thresholding in order to maximize the entropy criterion in [19], [20]. The design showed comparable results to that of the PSA, GA and BAT algorithms. Furthermore, the CSA was used to generate an optimal mask in order to suppress the noise found in speech signals [21].

Combinatorial optimization such as scheduling and resource allocation problems can effectively be solved using CSA. For instance, in [22] the CSA combined with random-key encoding scheme was successfully used to solve the travelling salesman problem (TSP). Virtual machine placement problem for resource optimization of data centers using CSA is investigated in [23]. Moreover, CSA is also found suitable for medical applications. Machine learning methods along with CSA and PSA are used to forecasting and diagnoses of heart diseases, breast cancer and diabetes [24], [25]. Another popular application that increasingly employ the CSA is clustering and mining applications. The CSA can be used to cluster medical data, web document and gene data clustering as specified in [26]–[28]. In data mining application, combining CSA with association rule mining (ARM) produces rules that are simple, easy to follow, and provide good coverage of the dataset as specified in [29]. In addition, this combination consumed less time than other algorithms which is very critical when number of items or transactions becomes large [30]. In [31], the CSA was examined in order to act as a cryptanalysis tool for cryptosystems specifically for Vigenere cipher. The simulation results show that the CSA successfully recover the cipher key with a performance better than genetic algorithm (GA) and PSA. The solution of the localization problem in wireless sensor network (WSN) is solved using CSA as indicated in [32]. It offers high localization accuracy in addition to fast convergence rate. In [33], Chaotic CS algorithm was successfully used to solve the path planning problems for UAV. The simulation results showed that the CSA give comparable results in terms of convergence rate, mean, standard deviation and the generated best solution when compared to PSA and ABC. It is worth mentioning that most of the research paper test the CSA or the hybrid CSA using a group of functions known as Benchmark functions before implementing it on the targeted application such as in [34]. These functions can be constrained and unconstrained, continuous and discrete variables, and unimodal and multimodal problems as specified in [1].

A growing trend in electronics circuits and components is reconfigurability. The best candidate for reconfigurability is field programable gate array (FPGA). The increasing cost of application specific integrated circuit (ASIC) and long time to market make the FPGA more appealing to researchers [35]. The FPGA consists mainly of configurable logic blocks (CLBs), hard-core intellectual property (IP) blocks and configurable wires. Modern FPGAs are suitable for real time applications since they include digital signal processing (DSP) blocks, digital clock management (DCM) blocks, memory controller, error correcting code (ECC) blocks and one or more dedicated microprocessors. Moreover, they include protocol engines supporting common peripheral interfaces and a variety of high-speed I/O standard peripherals [36]. Hardware description languages (HDL) are commonly used in programing FPGAs. Recently programing the FPGAs became easier as they can be programed using C-based languages, MATLAB and LabVIEW. FPGAs are suitable for the following applications: signal and image processing, financial applications, security, pattern matching, networking, numerical and scientific computing, molecular dynamics and optimization problems.

As the CSA requires high resources and arithmetic operations with high speed, it is considered the best candidate for FPGA hardware implementation. The use of FPGA reduces the computational time through utilizing the pipelining and parallel computation techniques [37]. In addition, FPGAs support arithmetic operations based on fixed- and floating-point formats. Floating point arithmetic is very crucial for DSP application and specific systems that require high data range, higher accuracy and high complexity [38], [39].

This paper is concerned with FPGA hardware implementation for CSA-based on IEEE single precision floating point format. To the best of our knowledge, the proposed design has not been published in literature yet.

This paper is organized as follows: Section 2 discusses the use of Mantegna's algorithm to generate random Levy flight walks followed by an overview on the CSA in section 3. Section 4 presents the proposed FPGA implementation for the CSA and the proposed control units that are implemented using FSM. Section 5 shows the simulation results and section 6 presents the system synthesis, performance evaluation results and the FPGA resources used. Finally, the conclusion of our work and the summary of the results.

II. LEVY FLIGHT BASED CUCKOO SEARCH

The main objective of the CS optimization algorithm, is to find a new and better solution for a certain problem. The nest that contains eggs is considered as a solution for the problem and the cuckoo's egg is considered as a new and better solution that replaces the old one. The percentage of the cuckoo's eggs that are discovered by the host must be replaced by a new solution. The location of the nest in the CS is found using a random walk which is a random process. It is a nature inspired technique that imitate the forging pattern path of animals and the flight behavior of birds and insects [3]–[5]. The random walk is composed of successive random steps and can be expressed as:

$$S_n = \sum_{i=1}^{n-1} X_i + X_n = S_{n-1} + X_n \tag{1}$$

where Sn the random walk with n random steps, Xi is the ith random step that has a predefined length and Xn is the motion or transition from the current to the next state. The above equation indicates that the next state depends on the current state in addition to the transition Xn. When the step size or length follows Levy's distribution, the random walk is called a Levy flight or Levy walk. Mantegna's algorithm is used to generate the Levy flight step length S in a fast yet accurate manner and can be evaluated using the following equation [40], [41]:

$$S = \frac{u}{|v|^{1/\beta}} \tag{2}$$

where β is a parameter between 1 and 2 usually taken as 1.5, u and v are random numbers derived from normal distributions as:

$$u \sim N\left(0, \sigma_u^2\right), \quad v \sim N\left(0, \sigma_v^2\right) \tag{3}$$

where

$$\sigma_u = \left\{ \frac{\Gamma\,(1+\beta)\sin\,(\pi\beta/2)}{\Gamma[\,(1+\beta)\,/2]\beta 2^{(\beta+1)/2}} \right\}^{1/\beta}, \quad \sigma_v = 1 \tag{4}$$

III. CUCKOO SEARCH ALGORITHM

Optimization problem by imitating the breeding behavior of the cuckoo bird as described earlier. The pseudocode in Figure 1 is used to exemplify how the CSA can be implemented. Each egg in a host nest (xi) represents a solution for the specified problem. The cuckoo algorithm is used to generate a new egg (xj) that represents a new solution for the problem. A fitness function also known as objective function (f(x)) is used to indicate if the Cuckoo's egg is similar to the host egg. If the egg is somehow similar to the host egg (Fi > Fj), it replaces the host egg aiming for a better or optimum solution for the problem. A portion of the unfit eggs (Pa) that represent the ones discovered by the host are replaced by new eggs. In order to describe the CSA there are three rules that must be considered while using this algorithm. The first rule is that each cuckoo lay only one egg at a time in a random nest. The second rule is eggs in the best nest are passed to the succeeding generation. The third rule dictates that the number of nests is constant and a portion of the nests are replaced by new ones to represent the eggs discovered by the host. The flowchart in Figure 2 is used to understand the operation of the CSA. In the beginning, the CSA starts by generating an initial population consisting of n host nests.

Cuckoo Search Algorithm

```
1:  Objective function $f(x)$, $x = (x_1, x_2, \ldots, x_d)^T$
2:  Generate an initial population of n randomly
    host nests $x_i$ ($i$= 1, 2, … , n)
3:  **While** t< Max_iteration **do**
4:      (t< Max_Generation) or (stop criterion)
5:      use Levy flight to generate a new nest
6:      evaluate the fitness of the generated nest
7:      choose a nest among n (say, $x_j$) randomly
8:      **if** $F_j > F_i$ **then**
9:          replace nest $x_j$ with $x_i$
10:     **end if**
11:     A fraction $Pa$ of the worse nests are
    replaced randomly generated new ones
12:     Keep best solution (or nest with quality
    solutions)
13:     Rank the solution and find the current best
14: **end while**
15: Postprocess results and visualization
```

FIGURE 1. Pseudocode of the CSA.

The cuckoo will use the Levy flight and start to lay eggs in these nests. The new nest quality is then evaluated using the fitness function. Then the calculated fitness of the new nest (Fj) is compared to the initial/old nest (Fi). If the new nest is better than the initial nest, the new nest will replace the initial nest otherwise the initial nest is unchanged. A portion of the worst nests, represented by the probability Pa, are replaced by new random ones to represent the eggs discovered by the bird. In this case the bird throws the eggs out of the nest or simply abandon the nest and build a new one [42]. Generating a new solution/nest follows the below equations:

$$x_i^{(t+1)} = x_i^{(t)} + \alpha \cdot S \cdot r \tag{5}$$

where

$$\alpha = \alpha_0 \cdot (x_i^{(t)} - x_{best}^{(t)}) \tag{6}$$

where $x_i^{(t+1)}$ is the generated new solution for the iteration t using the cuckoo's ith egg, $x_i^{(t)}$ is the current solution, $x_{best}^{(t)}$ is the best solution in the current iteration, $\alpha 0$ is a constant and it is usually greater than 0, α is the biased step size, and r is a random number from a Gaussian distribution. It is worth pointing out that in the real world if a cuckoo's egg is very similar to a host's eggs, then this cuckoo's egg is less likely to be discovered, thus the fitness should be related to the

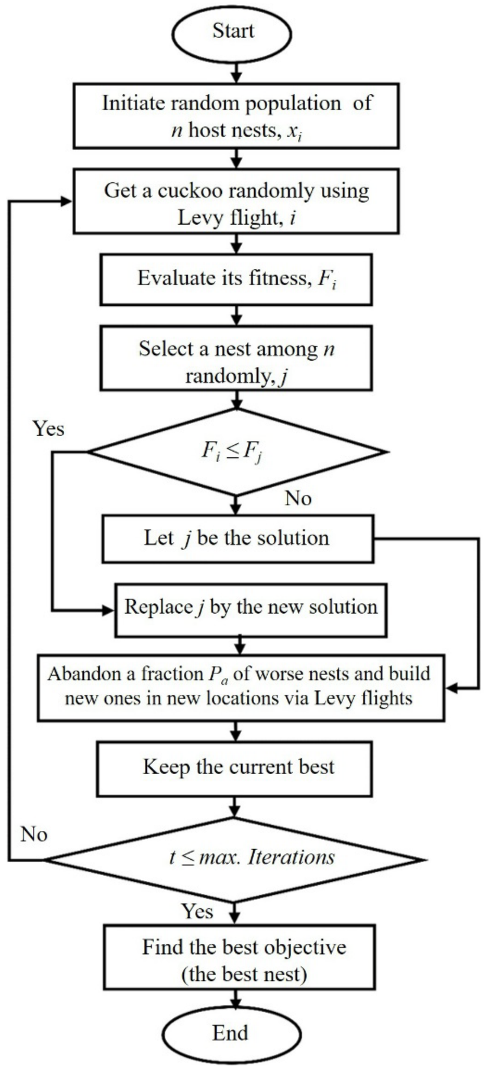

Start
Initiate random population of n host nests, x_i
Get a cuckoo randomly using Levy flight, i
Evaluate its fitness, F_i
Select a nest among n randomly, j
$F_i \le F_j$
Yes
No
Let j be the solution
Replace j by the new solution
Abandon a fraction P_a of worse nests and build new ones in new locations via Levy flights
Keep the current best
$t \le$ max. Iterations
No
Yes
Find the best objective (the best nest)
End

FIGURE 2. CSA operation flowchart [33].

difference in solutions. Therefore, it is a good idea to do a random walk in a biased way with some random step sizes as follows [43].

$$stepsize = rand \cdot \left(permute1\left(x_i^t\right) - permute2\left(x_i^t\right)\right) \tag{7}$$

where permute1 and permute2 are different random permutation functions applied to nests, and thus the new solution can be calculated using the following equation

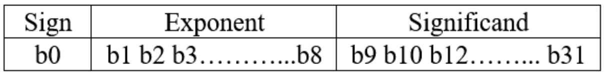
$$x_i^{t+1} = x_i^t + stepsize * P \tag{8}$$

The P is using the below equation where Pa is the fraction probability which is usually equals 0.25 [1]:

$$P = \begin{cases} 1 & if\ rand < Pa \\ 0 & if\ rand \ge Pa \end{cases} \tag{9}$$

IV. PROPOSED FPGA BASED HARDWARE IMPLEMENTATION

This section describes a detailed FPGA implementation for the proposed CSA design. The proposed design adopts the 32 bits single precision IEEE 754 standard floating-point formats. This format is suitable for a wide range of applications when compared to the fixed-point format. Floating point can represent very small or very large numbers as indicated in [35]. In this format, the number is composed of three parts: sign, exponent and significand. The sign is either ‘0’ or ‘1’ for positive or negative numbers respectively, while the exponent is an integer value represented in 8 bits. The significand or mantissa is represented in 23 bits as shown in Figure. 3 [35], [36].



Sign	Exponent	Significand
b0	b1 b2 b3………...b8	b9 b10 b12…….... b31

FIGURE 3. Representation of single precision 32-bit IEEE 754 floating point number.

The proposed CSA is composed of four main units: habitat memory unit (HM), get best nest unit (GBN), get cuckoo unit (GC), and empty nest unit (EN). A master control unit is designed to organize the operation of each unit in the proposed design.

The structure and operation of each unit will be explained in the following subsections. Standard benchmark functions can be used in order to validate the performance of any optimization algorithm as specified in [37], [38]. For our proposed design we adopted the two dimensional (2-D) ‘Sphere’ benchmark F2 function with search boundary range (−5, 5). The function can be expressed as follows:

$$\mathrm{F2}(x, y) = x^2 + y^2, \quad -\infty \le \mathrm{x}, \mathrm{y} \ge \infty \tag{10}$$

A. GET CUCKOO UNIT (GC)

The role of GC unit is to apply Levy flights which are random walks used to generate new nests (solutions) as described in equation (5). Two steps are required to generate this random walk with Levy flights. The first step is to choose a random direction, and the second step is to generate step length which obeys Levy distribution [41], [42]. In our proposed design, we apply Levy flights based on Mantegna’s approach which was expressed by equation (2). The β value is 3/2 and the random numbers u and v follow equation (3, 4).

To generate new solutions (new_nestx/ new_nesty), the GC unit takes lower bound (Lb), upper bound (Ub), nestx(i)/nesty(i) and best_nestx/ best_nestx as inputs. The GC generates a new solution and updates the existing nest in five pipelined phases as illustrated in Figure 4. The first phase after initializing the Levy flight step (S) based on Mantegna’s algorithm is to get the difference between the current nest and the best_nestx/ best_nesty to keep the best solution unchanged as expressed in equation (6). The second phase computes the step size of the walks by multiplying the Levy walk by the output of phase one. The third and fourth phases are to compute the actual random walks or flights as expressed in equation (5). In the fifth phase the bound checker module is enabled to check if the calculated solution is within the specified boundaries. If the new

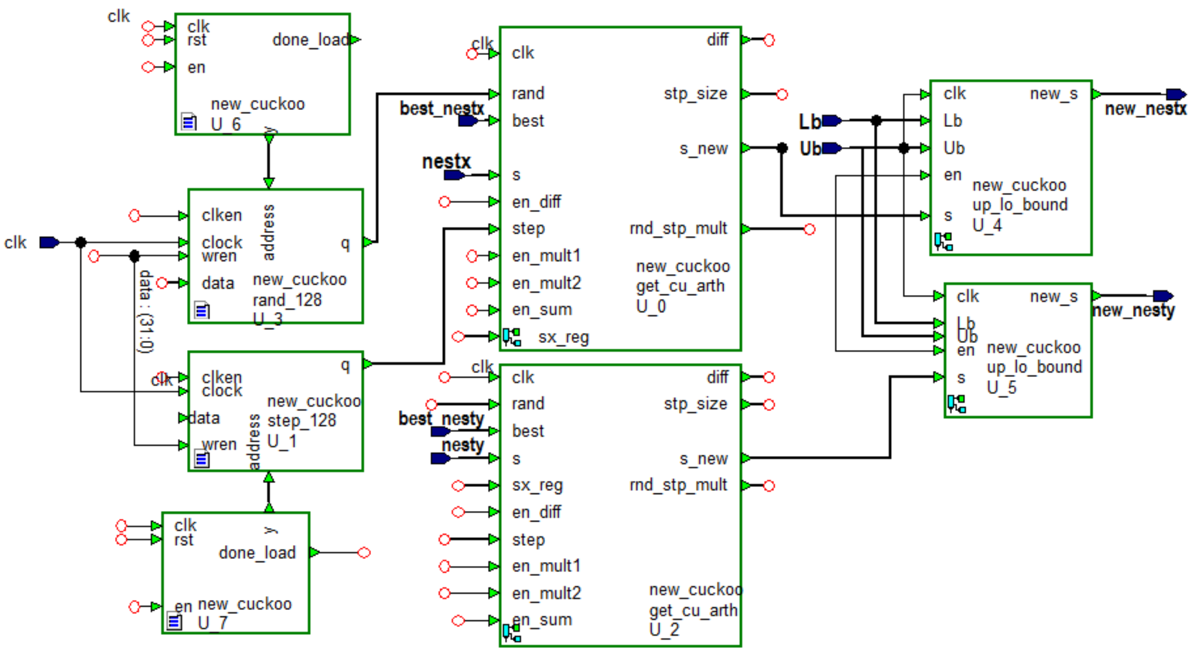


FIGURE 4. Block diagram of GC unit.

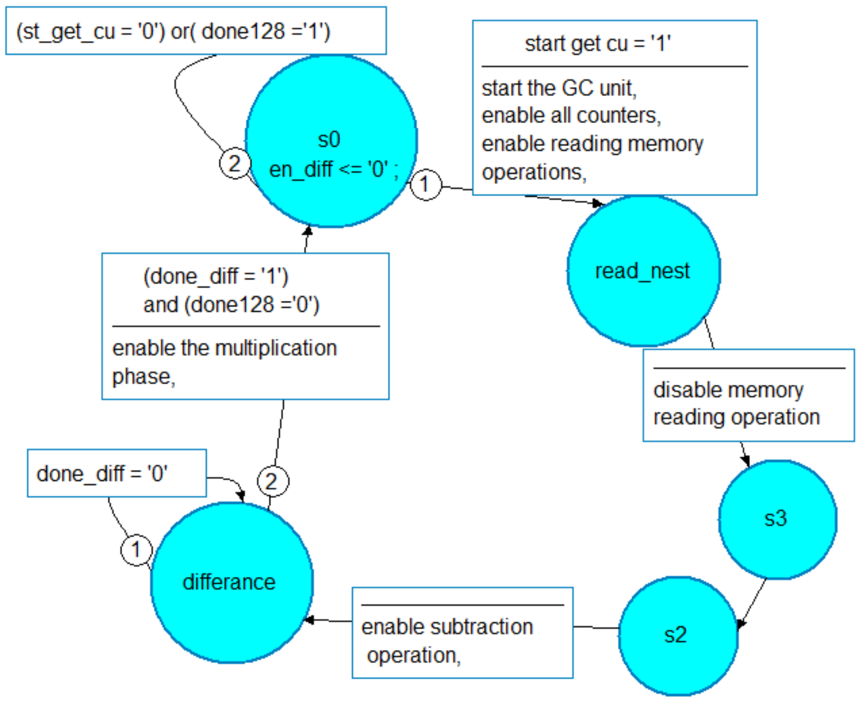


FIGURE 5. Part of the GC control unit (FSM).

solution lies within the specified boundaries, the GC unit will update the current nest. The GC unit is duplicated as this hardware is concerned to implement the F2 benchmark function.

The GC unit is supervised by an internal control unit which is implemented by the synthesized finite state machine (FSM) shown in Figure 5. This unit comprises five sub FSMs that control each of the six phases described above. Accordingly, this controller guarantees that the five pipelined phases are working in a parallel manner to get benefit from the FPGA hardware parallelism which is not provided if a central processing unit (CPU) is used instead.

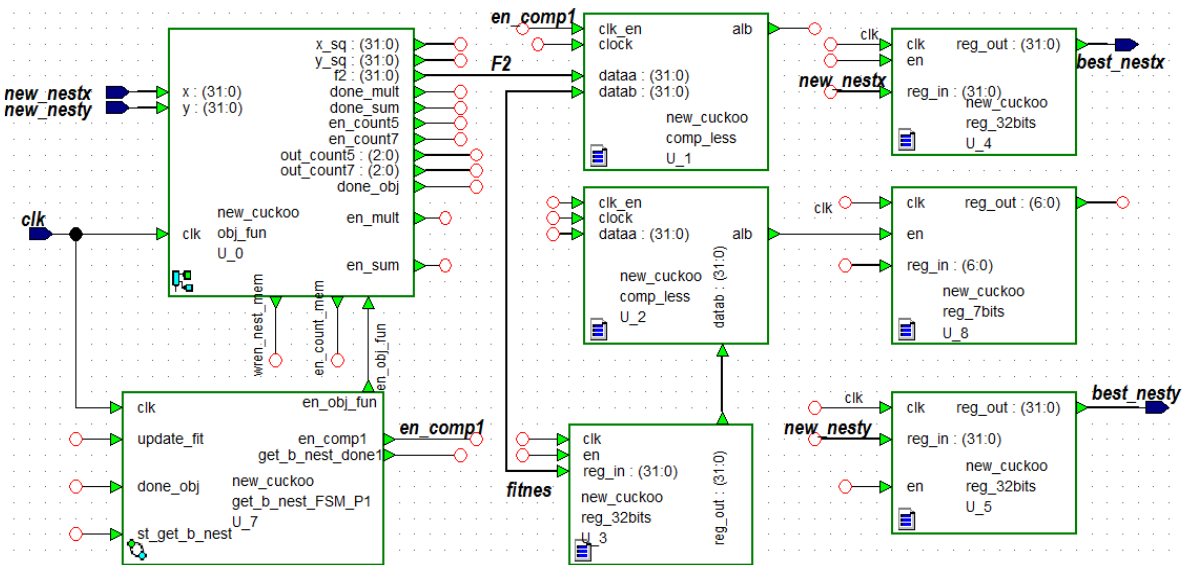


FIGURE 6. Block diagram of *GBN unit*.

B. GET BEST NEST UNIT (GBN)

In our proposed design, the GBN is responsible for updating both the nests (solutions) and the fitness memories (habitats). At the beginning, it evaluates the fitness of the new generated solution for both x and y (as we consider a 2-D function) and hence keep the solution that has the best fitness.

The inputs for this unit are the generated new solutions (new_nestx/ new_nesty) and the fitness. The outputs provided from this unit are the best_nestx/best_nesty and their corresponding minimum fitness, the updated nest and the fitness. The GBN accomplishes its calculation for each solution in three main pipelined phases as shown in Figure 6. In phase one, the fitness function is calculated for each new_nestx/new_nesty solution (line 6 in the algorithm as in Figure 1).

Then phase two starts by comparing the evaluated fitness with the fitness of the current solution to preserve the best solution. Hence update the nest and fitness habitat memories (line 8-10 in the algorithm as in Figure 1). Finally, the third phase keeps the best solution with the minimum fitness (line 12 in the algorithm as in Figure 1).

This unit is controlled by its dedicated control unit which is implemented by a synthesizable FSM. Similar to the control unit of the GC module, this unit is designed to ensure that the processing time is optimized by allowing all phases to operate in parallel. The detailed structure of the BNG control unit is shown in Figure 7.

C. EMPTY NEST (EN) UNIT

The role of the EN unit is to replace a fraction of the worst solutions Pa with random solutions which are generated by random Levy flight. The nest, Ub, Lb, and Pa are the input ports for EN unit while new_nest is the output port.

This unit accomplishes its function in five phases as shown in Figure 8. The first phase starts by getting the difference between two different nests which are chosen randomly. The second phase multiplies the output of the previous stage with a random number of RAND to adjust the step size of the random walks as in equation (7).

Each alien solution (egg) in the nest is discovered and replaced in phase three and four. The operation of these phases depends on the P values generated according to equation (9) that are stored in a separate SRAM memory. According to equation (8) the new solution is generated by adding the biased step size to the original solution if the value of P equals '1'

On the other hand, the original solution is unchanged if the value of P is '0'. In the final phase, the bounds are applied to the new solutions to guarantee that they lie in the search domain. All phases in the EN unit are coordinated using a FSM-based control unit as shown in Figure 9.

V. SIMULATION RESULTS AND DISCUSSION

The proposed CSA is designed using Advantage Pro 8 tool from Mentor Graphics. The simulations are performed using ModelSim SE Plus 6.3. The required parameters for simulation are saved in storage elements (SRAM or registers) and are loaded before running the simulation. The random initial population and fitness values are generated using a Matlab program and stored in a separate SRAM with a size of 128 nest. The value of pa is set to 0.25, $\sigma_v = 1$ and $\beta = 1.5$. All these parameters along with Ub and Lb are stored in separate registers. The following subsections explain the main design modules and their simulation results.

A. GET CUCKOO UNIT

The outputs of the five phases of the GC unit are diffx, stp_sizex, rnd_stp_multx, s_newx and new_nestx as shown in Figure 10. Each phase lasts 11 clock cycles in order to calculate its output. Figure 11 shows the control

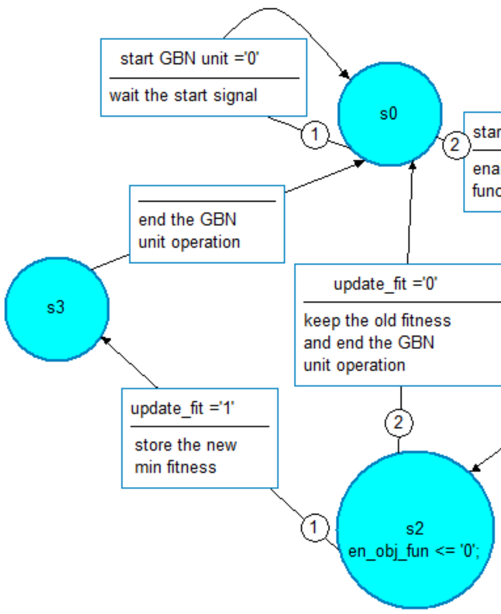

start GBN unit ='0'
wait the start signal
s0
start GBN unit='1'
enable objective function unit
end the GBN unit operation
done_obj ='0'
s1
s3
update_fit ='0'
keep the old fitness and end the GBN unit operation
update_fit ='1'
store the new min fitness
done_obj ='1'
enable comparison between the fitness values.
s2
en_obj_fun <= '0';

FIGURE 7. Part of the GBN FSM based control unit.

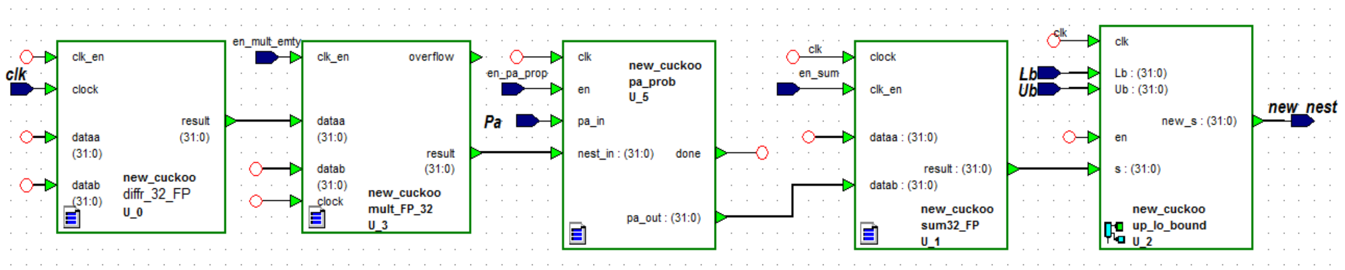


FIGURE 8. Block diagram of *EN unit*.

signals generated by the FSM. These signals are en_part2, en_part3 and en_part4 which enable the operation of phase 2, phase 3 and phase 4 respectively. The FSM generates these signals every 11 clock cycles (1100 ns) to control the operation of each phase. The operation of the GC unit will generate the final output new_nestx after 30 clock cycles i.e. latency equals 30 clocks. However, since the proposed design is pipelined and the FSM-based control unit is used to grantee that the outputs are overlapped, only the first output takes 14 clock cycles and further on each output takes 11 clock cycles. The pipeline operations due to the use of the FSM-based control unit increases the throughput of the unit.

B. GET BEST NEST UNIT (GBN)

The GBN unit simulation results are presented in Figure 12. This unit includes the objective function ($F2 = x2 + y2$) which calculates the fitness value f2 for both new solutions new_nestx and new_nesty as shown in Figure 12. From the simulation results, the objective function takes 15 clock cycles to evaluate f2. Then the new fitness f2 is compared with the stored one, which is called fitness in Figure 12.

The comparison operation starts when the internal FSM based control unit generates the control signal en_compl immediately when f2 is available. If the comparator output update_fit is high, the FSM generates control signals for the nestx, nesty and fitness habitat memories and updates their values. The duration of the update operation is 4 clock cycles and hence the total latency of the GBN unit is 19 clock cycles. On the other hand, if the comparator output update_fit is low, the values of nestx, nesty and fitness habitat memories are kept unchanged. Meanwhile during the update of the memories, the objective function starts to evaluate the fitness for another new nest.

Finally, when the unit finish evaluating the fitness for all nests it starts to preserve the best nests (bestx and besty) that have the minimum fitness (best_fit) as illustrated in Figure 13.

C. EMPTY NEST UNIT (EN)

The EN unit replaces the worst solutions that are calculated by the GBN unit based on Pa values with other random ones. As mentioned earlier, the EN unit acquires the

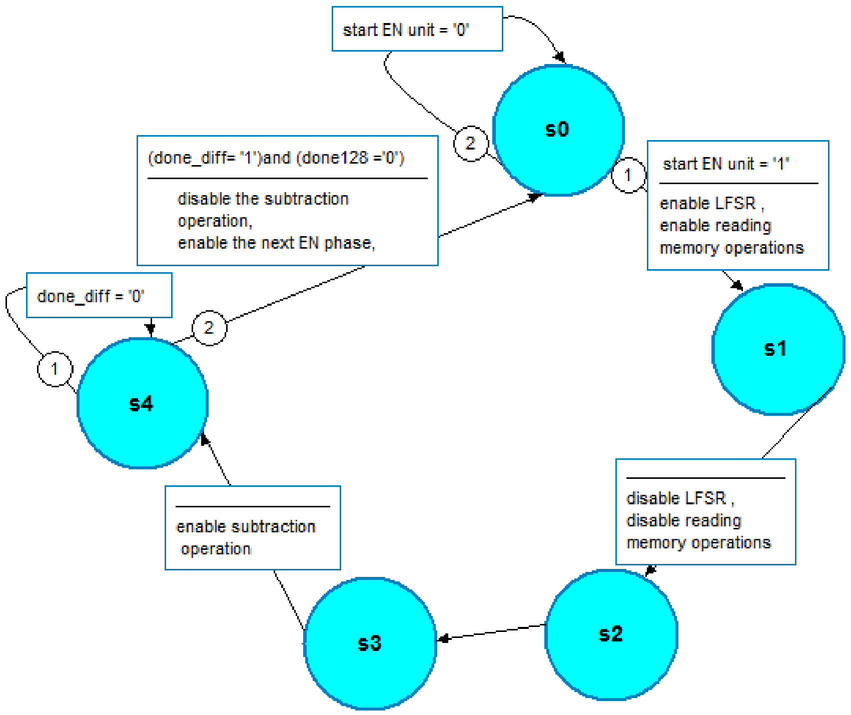
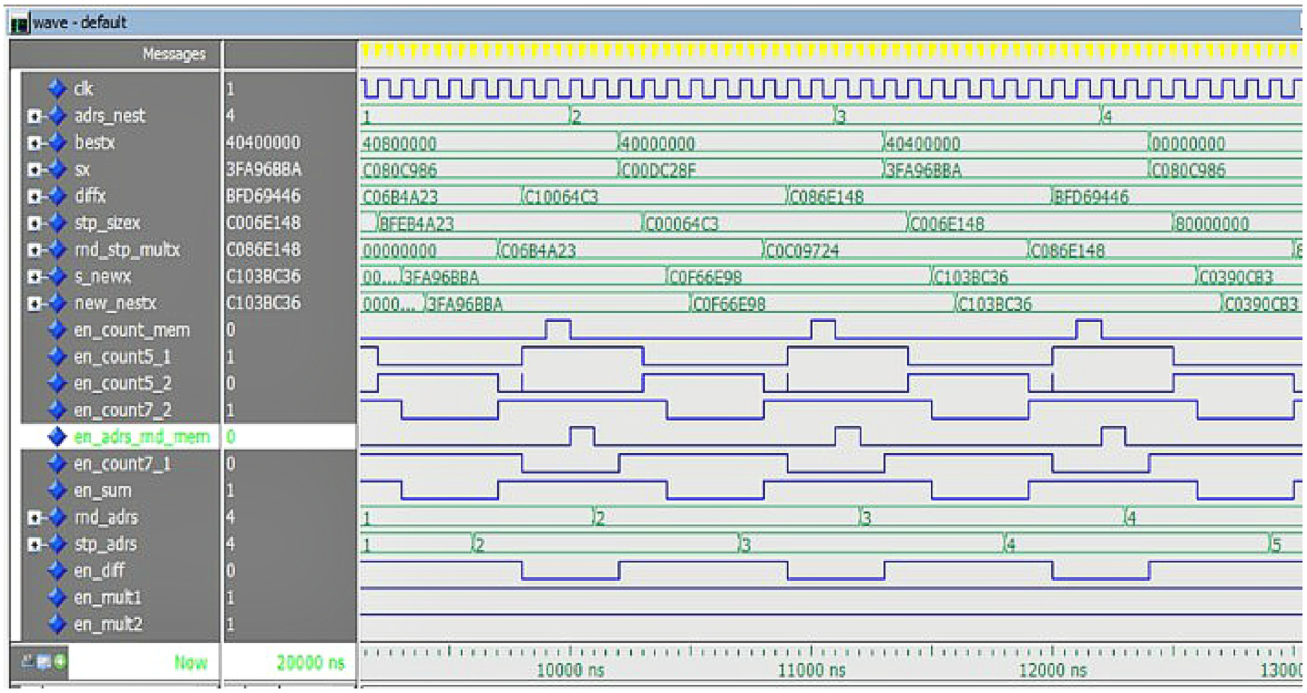


FIGURE 9. Part of the EN FSM-based control unit.


wave - default

Messages	
clk	1
adrs_nest	4
bestx	40400000
sx	3FA96BBA
diffx	BFD69446
stp_sizex	C006E148
rnd_stp_multx	C086E148
s_newx	C103BC36
new_nestx	C103BC36
en_count_mem	0
en_count5_1	1
en_count5_2	0
en_count7_2	1
en_adrs_rnd_mem	0
en_count7_1	0
en_sum	1
rnd_adrs	4
stp_adrs	4
en_diff	0
en_mult1	1
en_mult2	1
Now	20000 ns

FIGURE 10. Simulation results of GC unit.

difference between two stored solutions from the habitat. These solutions are addressed randomly using two linear shift registers (LFSRs). The random addresses adrs_x1 and adrs_x2 in Figure 14 represents the LFSRs outputs and nx_r1 and nx_r2 are the corresponding outputs for the nestx habitats. After seven clock cycles the difference nest_diff is evaluated and ready for the multiplication phase to get the step size.

The FSM sends control signals for the multiplication phase to start and another signal for the current phase to handle the

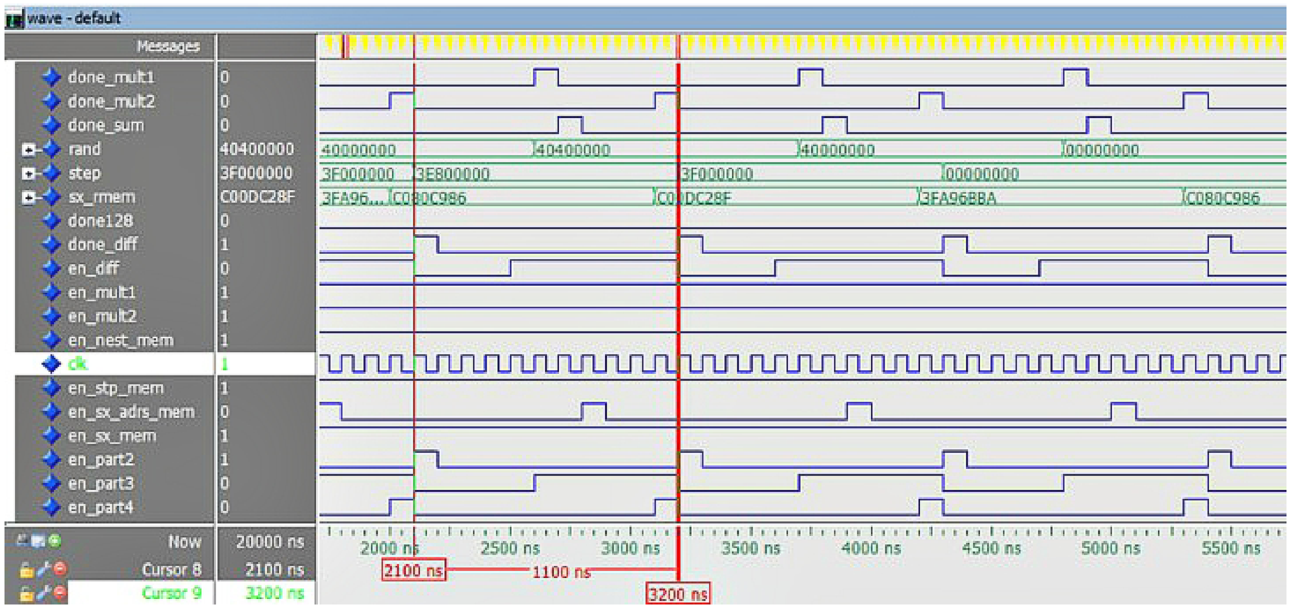


FIGURE 11. Simulation results of FSM control unit.

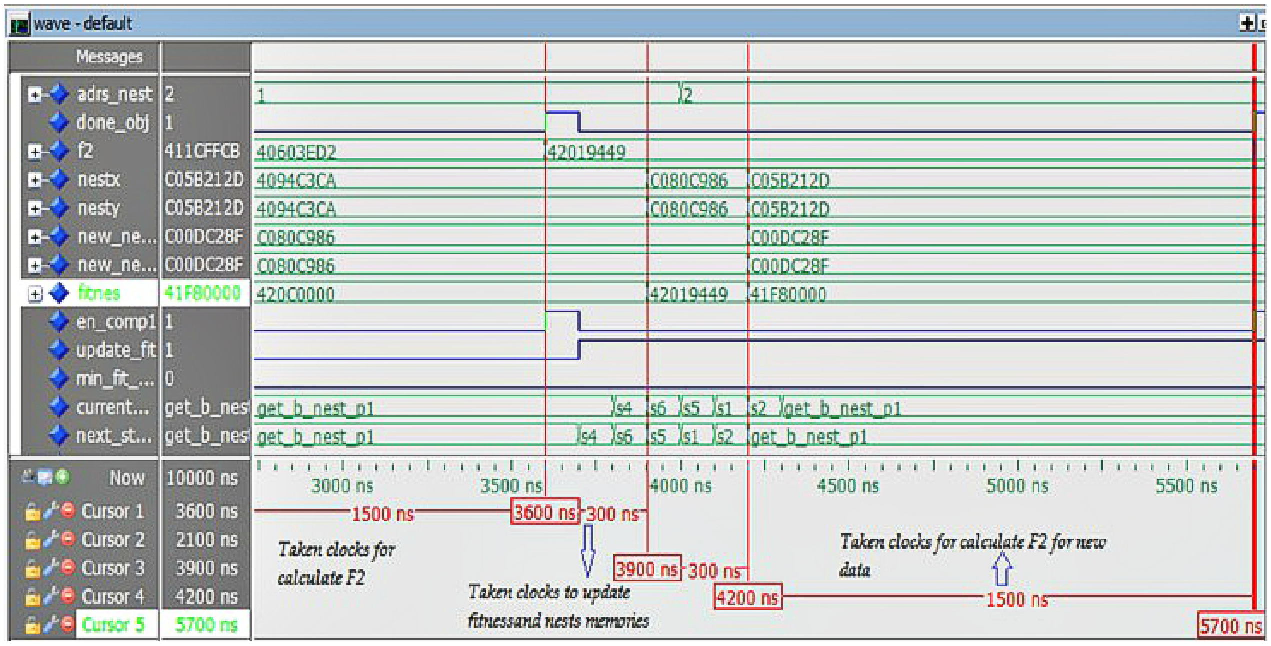


FIGURE 12. Fitness function *f2*. calculation and simulation results.

new nests. The step size (step_size) is calculated by multiplying the random value rnd_mem_empy which is stored on an SRAM with nest_diff.

This phase lasts five clock cycles. The calculated step_size is transferred to the addition block based on the pa_nest output which is generated in the next clock cycle based on the pa value. If pa value is ‘1’ the pa_nest equals step_size otherwise pa_nest equals zero as shown in Figure 14.

In summation phase, both pa_nest and the current solution nestx are added to get the new solution sum_op within 7 clock cycles. The final phase compares the new solution sum_op with the upper and lower bounds. The survive_nest is the result of this phase, which is confined in the solution range, and then will be stored in the habitat nest with a total latency of 32 clock cycles. The designed FSM controls this unit, ensures that the operation is pipelined and guarantees that the

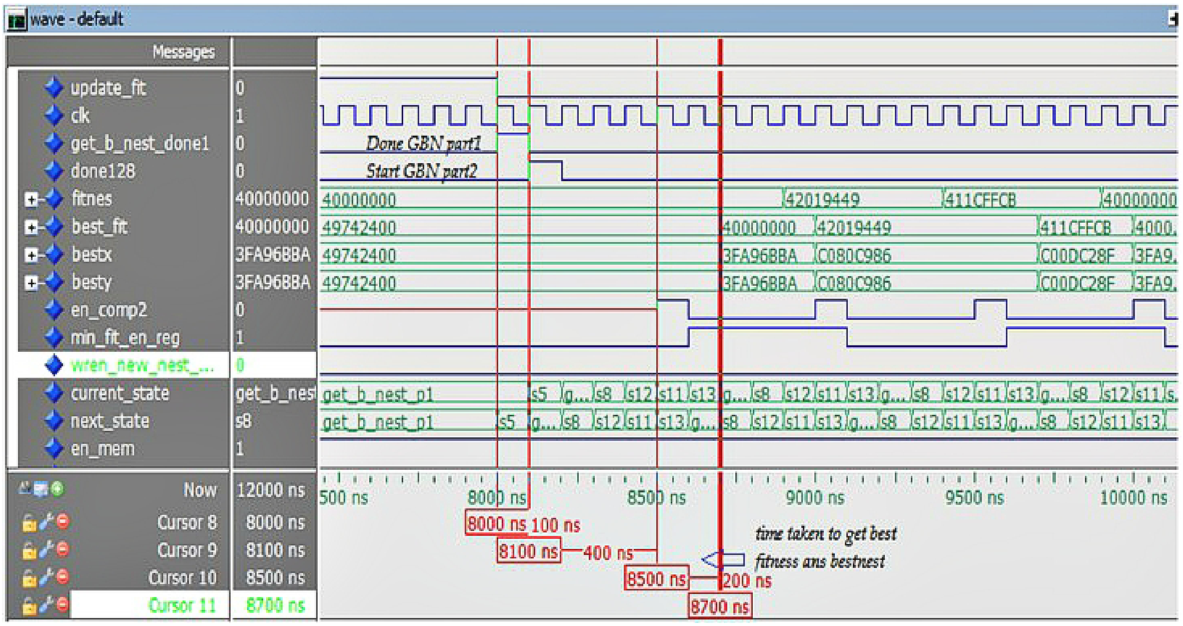


FIGURE 13. Simulation results of saving the best solution with minimum fitness.

unit generates survive_nest every 11 clocks cycles (1100 ns) as in Figure 14.

VI. SYSTEM SYNTHESIS AND PERFORMANCE EVALUATION

The targeted FPGA for hardware implementation is Cyclone® IV E FPGA from Intel (Altera). The FPGA chip consists of 114 k programmable logic elements (LEs), 388 embedded memory (Kbits), four PLLs, and 532 multipliers (9-bit). In addition, it contains 20 global clock networks, 8 user I/O banks and 528 maximum user I/O ports. Cyclone ® IV E offers low cost, low power and high functionality as indicated in [48].

A. SYNTHESIS RESULTS

The proposed hardware implementation of the CSA is synthesized using Quartus 15.1 tool from Intel. The resources utilized in designing F2 based CSA after place and route (P&R) are summarized in Table 1.

The post P&R results show that the proposed design occupies 7282 logic elements, 3754 register, around 58 k memory bits and 49 embedded multipliers. The total power dissipated is 424.83 mW and the maximum operating frequency for the proposed CSA is 99 MHz.

B. SYSTEM PERFORMANCE EVALUATION

In order to evaluate the performance of the proposed CSA system, two of the most common benchmark functions were implemented along with the spherical function as in [49], [50].

The functions are Rosenbrock (F1) and Rastrigin (F9) functions. The F1 function, also known as the Banana function, is a non-convex, unimodal and non-separable function. It is defined as follows

TABLE 1. Place and route results for 2-D F2.

	Available	GC unit	GBN unit	EN unit	Total
Total logic elements	114,480	3,540	1392	2350	7,282
Total registers		1860	754	1140	3754
Total memory bits	3,981312	20621	20541	16,456	57,618
Embedded Multiplier 9-bit elements	532	28	14	7	49
Dynamic. power consumption (mW)		74.41	2.47	50.75	127.63
Static power consumption (mW)		99.08	98.98	99.14	297.2
Total power consumption (mW)		173.49	101.45	149.89	424.83
Max. Freq. MHz		101.33	97.22	99.18	97

$$F_1 = \sum_{i=1}^{d-1} (1-x_i)^3 + 100\left(y_i - x_i^2\right)^2 \tag{11}$$

The function is in the range $-10 \leq$ xi, yi ≤ 10 where d is the domain dimension.

The Rastrigin function (F9) is a non-convex, multimodal and separable function. The function falls in the range $-5.12 \leq$ xi ≤ 5.12 and is defined in d-dimensions as in

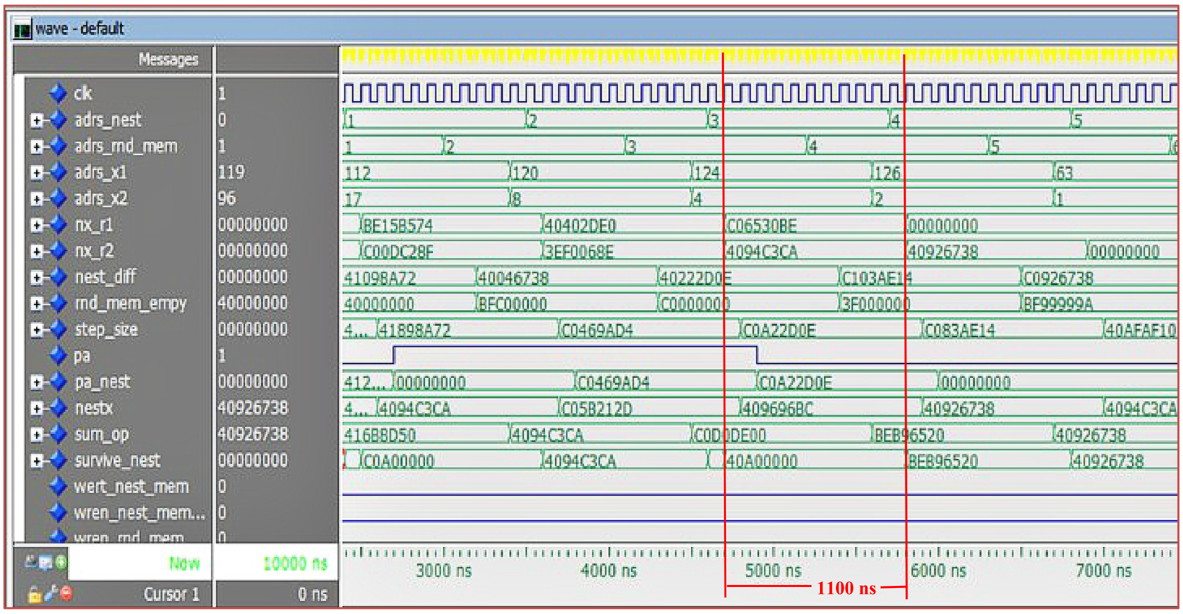


FIGURE 14. Simulation results of FSM control unit.

TABLE 2. Place and route results for 2-D F1.

	Available	GC unit	GBN unit	EN unit	Total
Total logic elements	114,480	3,540	2523	2350	8,413
Total registers		1860	1285	1140	4285
Total memory bits	3,981312	20621	12505	16,456	49,582
Embedded Multiplier 9-bit elements	532	28	21	7	56
Dynamic. power consumption (mW)		74.41	57.42	50.75	182.58
Static power consumption (mW)		99.08	99.13	99.14	297.35
Total power consumption (mW)		173.49	156.55	149.89	479.93
Max. Freq. (MHz)		101.33	106.09	99.18	99

TABLE 3. Place and route results for 2-D F9.

	AVAILABLE	GC UNIT	GBN UNIT	EN UNIT	TOTAL
Total logic elements	114,480	3,540	11864	2350	17754
Total registers		1860	6515	1140	9515
Total memory bits	3,981312	20621	17356	16,456	54433
Embedded Multiplier 9-bit elements	532	28	104	7	139
Dynamic. power consumption (mW)		74.41	186.63	50.75	311.79
Static power consumption (mW)		99.08	100.27	99.14	298.49
Total power consumption (mW)		173.49	286.9	149.89	610.28
Max. Freq. (MHz)		101.33	98	99	98

equation 12.

$$F_9 = 10 \cdot d - \sum_{i=1}^{d} \left[x_i^2 + 10 \cos\left(2\pi x_i\right) \right] \quad (12)$$

Tables 2 and 3 show the synthetization results for 2-D Rosenbrock (F1) and 2-D Rastrigin (F9) respectively on the same FPGA. The GC and EN modules are kept the same while the GBN module is modified according to the implemented objective function. The implementation results show that as the complexity of the objective function increases the FPGA utilization increases. When compared to F2, the total power consumption is increased by 13 % for F1 and 44 % for F9. The maximum frequency for F1 is 99 MHz. and the maximum frequency of F9 is 98 MHz.

In addition, four-dimensional (4-D) operation is considered to evaluate the performance of the proposed hardware design. New 4-D GBN modules for both Rosenbrock (F1) and Rastrigin (F9) functions are designed based on parallel architecture. Minor changes are done for the GC unit and EN unit to be compatible with the 4-D operations. Table 4 shows the allocated resources, power consumption and maximum operating frequency of the 4-D F1 and F9 functions. The resources allocated for the 4-D F1 increased in the range

TABLE 4. Place and route results for 4-D operation.

	Available		4-D F1		4-D F9
Total logic elements	114,480	8,413	14723	17754	25151
Total registers		4285	7730	9515	13240
Total memory bits	3,981312	49,582	74509	54433	63182
Embedded Multiplier 9-bit elements	532	56	119	139	191
Dynamic. power consumption (mW)		182.58	298.12	311.79	404.37
Static power consumption (mW)		297.35	298.49	298.49	299.37
Total power consumption (mW)		479.93	596.61	610.28	703.74
Max. Freq. (MHz)		99	99	98	99

TABLE 5. Performance Comparison.

	This work 32 bits FP format			[51] 32 bits fixed point format		
	F1	F2	F9	F1	F2	F9
Logic Elements	8,413	7,282	13,121	42,146	8,449	100,30 2
Total Registers	4,285	3,754	6,888	9,575	1,583	2,920
Total memory bits	49,582	57,618	49,862	32,801	6,945	97,869
Multipliers	56	49	94	NA	NA	NA
Dynamic. power consumption (mW)	182.58	127.63	242.11	113.53	81.18	365.36
Static power consumption (mW)	297.35	297.2	290.27	78.65	34.79	70.3
Total power (mW)	479.93	424.83	532.38	191.18	115.98	435.66
Max. Frequency (MHz)	99	99	99	246.12	299.65	250.47

between 50% and 113% when compared to that of the 2-D design. Compared to the 2-D design, the allocated resources for the 4-D F2 increased in the range from 16% to 42%. The total power consumption for the 4-D F1 and F9 designs are 596.91 mW and 703.74 mW respectively. The results show that the total power consumption for the F1 design increased by 24% and by 15% for the F9 design when compared to the 2-D designs. Since both designs employ parallel architecture the maximum operating frequency is maintained at 99 MHz. The proposed architecture sacrificed the allocated resources and power consumption in order to preserve the operating frequency.

C. PERFORMANCE COMPARISON

The performance of the proposed CSA is compared to the recently published work in [51]. The CSA in [51] is based on fixed point format and implemented on Cyclone IV GX FPGA from Altera.

Table 5 sums up the results of both designs in terms of FPGA utilization, consumed power and maximum operation frequency. The habitat of the proposed CSA is 128 nests while that of [51] is 75 nests with 32 bits word size for both designs. The proposed design utilizes logic elements ranging from 8,413 to 13,121 that are less than the elements used in [51]. On the other hand, more registers are used in the proposed design with a maximum of 6888 registers.

As far as the occupied memory is concerned, the proposed design utilizes around 50 k bits for both F1 and F9 based systems. The F2 based system employs around 58 k memory bits which is higher than that in [51].

The proposed CSA made use of the provided 9-bit embedded multiplier elements in all the designs while in [51] no embedded multipliers were used. The total power consumption for the proposed design, with different objective functions, is relatively high due to two main reasons. The first reason is the use of FP arithmetic which is more complex than fixed point arithmetic. The second reason is the number of nests used which is 70% higher in the proposed design than that in [51].

The spherical function F2 consumes less power as it has less arithmetic operations. The consumed power is 480 mW for the proposed design and 116 mW for the design in [51]. For more complex objective functions such as F9, the power consumption increased around 25 % in the proposed design and 275 % in [51] when compared to F2 based systems. The maximum frequency range for the design in [51] is 250 - 300 MHz while in the proposed design the maximum frequency is settled around 99 MHz. Although the proposed design consumes more power and has a lower operating frequency, it provides a higher precision and a wider range which makes it suitable for DSP and biomedical applications.

VII. CONCLUSION

This paper presented an FPGA hardware implementation for CSA based on IEEE single precision floating point data. The design adopted Mantegna's algorithm to generate a random Levy flight walk. The proposed design consisted of four main units: HM unit, GBN unit, GC unit, and EN unit. All these modules were controlled using FSM based control units. The system is designed using parallel and pipeline techniques to maintain a reasonable operating speed. A 2-D benchmark spherical function is used to validate the proposed hardware design.

The design was implemented on Cyclone IV® E from Intel. It occupied 6.4 % of the available logic elements, 3754 register, less the 1.4 % of the available memory and 49 embedded multipliers. The maximum speed achieved was 99 MHz and the consumed power was 424.83 mW which was relatively high due to the use of the FP format and the large number of nests. Rosenbrock and Rastrigin function were used to evaluate the performance of the proposed CSA. The total consumed power increased by 13% for the Rosenbrock based design and by 44% for the Rastrigin based design when compared to the spherical function-based design. 4-D operation of the F1 and F9 was also examined.

The results showed that the allocated resources and power consumption increased while the maximum frequency is maintained around 99 MHz. This is attributed to the parallel architecture used in designing the CSA. In addition, the performance of the proposed CSA was compared to recently published research. Even though the design in [51] was based on fixed point format, the proposed CSA design gave comparable results especially for complex objective functions. From the obtained results, it could be concluded that the proposed design can be used to solve optimization problems related to WSN, UAV, DSP and medical diagnostic applications as they require wide range and high precision.

REFERENCES

[1] M. Shehab, A. T. Khader, and M. A. Al-Betar, "A survey on applications and variants of the cuckoo search algorithm," *Appl. Soft Comput.*, vol. 61, pp. 1041–1059, Dec. 2017.

[2] X. Yang and S. Deb, "Cuckoo search via Lévy flights," in *Proc. World Congr. Nature Biologically Inspired Comput. (NaBIC)*, Coimbatore, India, Dec. 2009, pp. 210–214.

[3] P. Civicioglu and E. Besdok, "A conceptual comparison of the Cuckoo-search, particle swarm optimization, differential evolution and artificial bee colony algorithms," *Artif. Intell. Rev.*, vol. 39, no. 4, pp. 315–346, Apr. 2013.

[4] C. T. Brown, L. S. Liebovitch, and R. Glendon, "Lévy flights in Dobe Ju/'hoansi foraging patterns," *Hum. Ecol.*, vol. 35, no. 1, pp. 129–138, Feb. 2007. doi: 10.1007/s10745-006-9083-4.

[5] A. M. Reynolds and M. A. Frye, "Free-flight odor tracking in drosophila is consistent with an optimal intermittent scale-free search," *PLoS ONE*, vol. 2, no. 4, pp. 1–9, 2007. doi: 10.1371/journal.pone.0000354.

[6] H. R Soneji and R. C Sanghvi, "Towards the Improvement of Cuckoo Search Algorithm," *Int. J. Comp. Inf. Syst. Ind. Manage. App. IJCISIM*, vol. 6, pp. 77–88, 2014. [Online]. Available: http://www.mirlabs.org/ijcisim/volume_6.html

[7] I. Fister, Jr., D. Fister, and I. Fister, "A comprehensive review of cuckoo search: Variants and hybrids," *Int. J. Math. Model. Numer. Optim.*, vol. 4, no. 4, pp. 387–409, 2013.

[8] K. Ilamathi and P. Rangarajan, "Intelligent computation techniques for optimization of the shortest path in an asynchronous network-on-chip," *Cluster Comput.*, vol. 22, pp. 335–346, Jan. 2019. doi: 10.1007/s10586-018-1924-6.

[9] M. A. Adnan and M. A. Razzaque, "A comparative study of Particle Swarm Optimization and Cuckoo Search techniques through problem-specific distance function," in *Proc. Int. Conf. Inf. Commun. Technol. (ICoICT)*, Bandung, Indonesia, Mar. 2013, pp. 88–92. doi: 10.1109/ICoICT.2013.6574619.

[10] A. Kumar and S. Chakarverty, "Design optimization using genetic algorithm and Cuckoo search," in *Proc. IEEE Int. Conf. Electro/Inf. Technol.*, Mankato, MN, USA, May 2011, pp. 1–5. doi: 10.1109/EIT.2011.5978616.

[11] X. Yang and S. Deb, "Cuckoo search: State-of-the-art and opportunities," in *Proc. IEEE 4th Int. Conf. Soft Comput. Mach. Intell. (ISCMI)*, Port Louis, Mauritius, Nov. 2017, pp. 55–59. doi: 10.1109/ISCMI.2017.8279597.

[12] J.-Y. Shi, F. Xue, Z.-J. Qin, W. Zhang, L.-T. Ling, and T. Yang, "Improved global maximum power point tracking for photovoltaic system via cuckoo search under partial shaded conditions," *J. Power Electron.*, vol. 16, no. 1, pp. 287–296, Jan. 2016. doi: 10.6113/JPE.2016.16.1.287.

[13] M. A. Enany, "Cuckoo search-based maximum power point tracking controller for PV water pumping system," *J. Renew. Sustain. Energy*, vol. 9, Nov. 2017, Art. no. 063501. doi: 10.1063/1.5009681.

[14] A. Kumar and S. Chakarverty, "Design optimization for reliable embedded system using Cuckoo search," in *Proc. 3rd Int. Conf. Electron. Comput. Technol.*, Kanyakumari, India, Apr. 2011, pp. 264–268. doi: 10.1109/ICECTECH.2011.5941602.

[15] C. Qu and W. He, "A cuckoo search algorithm with complex local search method for solving engineering structural optimization problem," in *Proc. MATEC Web Conf.*, vol. 40, 2016, p. 09009. doi: 10.1051/matecconf/20164009009.

[16] H. J. Xu, J. K. Liu, and Z. R. Lu, "Structural damage identification based on cuckoo search algorithm," *Math. Models Eng.*, vol. 1, no. 1, pp. 1–11, May 2015. doi: 10.1177/1369433216630128.

[17] N. Agrawal, A. Kumar, V. Bajaj, and G. K. Singh, "High order stable infinite impulse response filter design using cuckoo search algorithm," *Int. J. Automat. Comput.*, vol. 14, no. 5, pp. 589–602, Oct. 2017. doi: 10.1007/s11633-017-1091-x.

[18] S. Pare, A. Kumar, V. Bajaj, and G. K. Singh, "A multilevel color image segmentation technique based on cuckoo search algorithm and energy curve," *Appl. Soft Comput.*, vol. 47, pp. 76–102, Oct. 2016. doi: 10.1016/j.asoc.2016.05.040.

[19] S. J. Mousavirad and H. Ebrahimpour-Komleh, "Entropy based optimal multilevel thresholding using cuckoo optimization algorithm," in *Proc. 11th Int. Conf. Innov. Inf. Technol. (IIT)*, Dubai, United Arab Emirates, Nov. 2015, pp. 302–307. doi: 10.1109/INNOVATIONS.2015.7381558.

[20] S. J. Mousavirad and H. Ebrahimpour-Komleh, "Multilevel image thresholding using entropy of histogram and recently developed population-based metaheuristic algorithms," *Evol. Intell.*, vol. 10, nos. 1–2, pp. 45–75, 2017.

[21] A. Garg and O. P. Sahu, "Cuckoo search based optimal mask generation for noise suppression and enhancement of speech signal," *J. King Saud Univ. Comput. Inf. Sci.*, vol. 27, no. 3, pp. 269–277, 2015. doi: 10.1016/j.jksuci.2014.04.006.

[22] A. Ouaarab, B. Ahiod, and X.-S. Yang, "Random-key cuckoo search for the travelling salesman problem," *Soft Comput.*, vol. 19, no. 4, pp. 1099–1106, 2015. doi: 10.1007/s00500-014-1322-9.

[23] S. M. Sait, A. Bala, and A. H. El-Maleh, "Cuckoo search based resource optimization of datacenters," *Appl. Intell.*, vol. 44, no. 3, pp. 489–506, Apr. 2016. doi: 10.1007/s10489-015-0710-x.

[24] X. Liu and H. Fu, "PSO-based support vector machine with cuckoo search technique for clinical disease diagnoses," *Sci. World J.*, vol. 2014, May 2014, Art. no. 548483. doi: 10.1155/2014/548483. [Online]. Available: https://www.hindawi.com/journals/tswj/2014/548483/cta/

[25] D. Giveki, H. Salimi, R. Bahmanyar, and Y. Khademian, "Automatic detection of diabetes diagnosis using feature weighted support vector machines based on mutual information and modified cuckoo search," Jan. 2012, *arXiv:1201.2173*. [Online]. Available: https://arxiv.org/abs/1201.2173

[26] P. Mohapatra, S. Chakravarty, and P. K. Dash, "An improved cuckoo search based extreme learning machine for medical data classification," *Swarm Evol. Comput.*, vol. 24, pp. 25–49, Oct. 2015.

[27] M. Zaw and E. Mon, "Web document clustering using Cuckoo search clustering algorithm based on levy flight," *Int. J. Innov. Appl. Stud.*, vol. 4, no. 1, pp. 182–188, Sep. 2013.

[28] R. Rajeswari and G. GunaSekaran, "An improved cuckoo search based robust ensemble co-clustering algorithm (Ics–Recca) for enzyme clustering," *Int. J. Comput. Intell. Inform.*, vol. 5, no. 3, pp. 198–206, Dec. 2015.

[29] U. Mlakar, M. Zorman, I. Fister, and I. Fister, "Modified binary cuckoo search for association rule mining," *J. Intell. Fuzzy Syst.*, vol. 32, no. 6, pp. 4319–4330, 2017.

[30] R. A. Mohammed, M. G. Duaimi, "Association rules mining using cuckoo search algorithm," *Int. J. Data Mining Model. Manage.*, vol. 10, no. 1, pp. 73–88, 2018.

[31] A. K. Bhateja, A. Bhateja, S. Chaudhury, and P. K. Saxena, "Cryptanalysis of Vigenere cipher using Cuckoo search," *Appl. Soft Comput.*, vol. 26, pp. 315–324, Jan. 2015.

[32] S. Goyal and M. S. Patterh, "Wireless sensor network localization based on cuckoo search algorithm," *Wireless Pers. Commun.*, vol. 79, no. 1, pp. 223–234, Nov. 2014.

[33] J.-S. Pan, J.-L. Liu, and S.-C. Hsiung, "Chaotic cuckoo search algorithm for solving unmanned combat aerial vehicle path planning problems," in *Proc. 11th Int. Conf. Mach. Learn. Comput.*, Feb. 2019, pp. 224–230.

[34] G. P. Gupta and S. Jha, "Integrated clustering and routing protocol for wireless sensor networks using Cuckoo and Harmony Search based metaheuristic techniques," *Eng. Appl. Artif. Intell.*, vol. 68, pp. 101–109, Feb. 2018.

[35] S. M. Trimberger, "Three ages of FPGAs: A retrospective on the first thirty years of FPGA technology," *Proc. IEEE*, vol. 103, no. 3, pp. 318–331, Mar. 2015.
[36] J. C. Lyke, C. G. Christodoulou, G. A. Vera, and A. H. Edwards, "An introduction to reconfigurable systems," *Proc. IEEE*, vol. 103, no. 3, pp. 291–317, Mar. 2015.
[37] R. Tessier, K. Pocek, and A. DeHon, "Reconfigurable computing architectures," *Proc. IEEE*, vol. 103, no. 3, pp. 332–354, Mar. 2015.
[38] R. N. Mantegna, "Fast, accurate algorithm for numerical simulation of Lévy stable stochastic processes," *Phys. Rev. E, Stat. Phys. Plasmas Fluids Relat. Interdiscip. Top.*, vol. 49, no. 5, pp. 4677–4683, 1994.
[39] S. Rani, "Area and speed efficient floating point unit," in *Proc. Int. Conf. Recent Adv. Innov. Eng. (ICRAIE)*, Jaipur, India, May 2014, pp. 1–4.
[40] M. Langhammer and B. Pasca, "Design and implementation of an embedded FPGA floating point DSP block," in *Proc. IEEE 22nd Symp. Comput. Arithmetic*, Lyon, France, Jun. 2015, pp. 26–33.
[41] X. Yang, *Nature-Inspired Metaheuristic Algorithms*, 2nd ed. London, U.K.: Luniver Press, 2010, pp. 105–115.
[42] K. N. A. Rani, M. F. AbdulMalek, and S. Neoh, "Nature-inspired cuckoo search algorithm for side lobe suppression in a symmetric linear antenna array," *Radio Eng. J.*, vol. 21, no. 3, pp. 865–874, Sep. 2012.
[43] A. Kaveh, *Advances in Metaheuristic Algorithms for Optimal Design of Structures*, 2nd ed. Springer, Cham, Switzerland: 2017, pp. 321–354.
[44] V. Rajaraman, "IEEE standard for floating point numbers," *Resonance*, vol. 21, pp. 11–30, Jan. 2016.
[45] M. Langhammer and B. Pasca, "Floating-point DSP block architecture for FPGAs," in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays*, Monterey, CA, USA, Feb. 2015, pp. 117–125.
[46] M. Jamil and X.-S. Yang, "A literature survey of benchmark functions for global optimisation problems," *Int. J. Math. Model. Numer. Optim.*, vol. 4, no. 2, p. 150, 2013.
[47] J. M. Dieterich and B. Hartke, "Empirical review of standard benchmark functions using evolutionary global optimization," *Appl. Math.*, vol. 3, no. 10, pp. 1552–1564, 2012.
[48] Cyclone. *IV FPGAs Product Table*. Accessed: Jun. 15, 2019. [Online]. Available: https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/pt/cyclone-iv-product-table.pdf
[49] M. S. B. Ameur and A. Sakly, "FPGA based hardware implementation of bat algorithm," *Appl. Soft Comput.*, vol. 58, pp. 378–387, Sep. 2017.
[50] M. Peker, "A fully customizable hardware implementation for general purpose genetic algorithms," *Appl. Soft Comput.*, vol. 62, pp. 1066–1076, Jan. 2018.
[51] M. G. Alfailakawi, M. El-Shafei, I. Ahmad, and A. Salman, "FPGA-based implementation of cuckoo search," *IET Comput. Digit. Techn.*, vol. 13, no. 1, pp. 28–37, 2019.

HANADY HUSSEIN ISSA received the B.Sc. and M.Sc. degrees from the Arab Academy for Science Technology and Maritime Transport (AASTMT), Egypt, in 1998 and 2003, respectively, and the Ph.D. degree in electronics and communication engineering from Ain Shams University, Egypt, in 2009.

She was a Teaching Assistant with AASTMT, where she became the Director of the Education Planning Unit, in 2016. She is currently a Professor with the Electronics and Communication Department, AASTMT, Egypt. Concurrently, she has also been with the Director of the Center of Excellence (COE) in Nanotechnology, AASTMT, since 2017. Since 2009, she has supervised more than 20 master's/Ph.D. theses in the areas of digital design for communication systems based on FPGA and low power design. She has coauthored more than 30 articles. Her research interests include analog/digital circuits design, VHDL-based FPGA design simulation and synthesis, and low power design.

SALEH MOHAMED EISA AHMED received the B.Sc. and M.Sc. degrees from the Arab Academy for Science, Technology and Maritime Transport (AASTMT), in 2000 and 2006, respectively, and the Ph.D. degree from Ain Shams University, Cairo, Egypt, in 2014.

From 2015 to 2016, he was an Assistant Professor with the Electronics and Communication Department, Faculty of Engineering, AASTMT at Cairo. He is currently the Acting Head of the Electronics and Communication Department, Faculty of Engineering, AASTMT at Smart Village. His research interests include analog and digital VLSI design, low power design, FPGA-based system design, and printed electronics.