Neural Computing and Applications (2021) 33:8893–8922
https://doi.org/10.1007/s00521-020-05637-5

ORIGINAL ARTICLE

Application of ameliorated Harris Hawks optimizer for designing of low-power signed floating-point MAC architecture

Rajkumar Sarma[1] · Cherry Bhargava[1] · Shruti Jain[2] · Vikram Kumar Kamboj[3,4]

Received: 31 December 2019 / Accepted: 15 December 2020 / Published online: 16 January 2021
© The Author(s), under exclusive licence to Springer-Verlag London Ltd. part of Springer Nature 2021

Abstract

Recently established Harris Hawks optimization (HHO) has natural behaviour for finding an optimum solution in global search space without getting trapped in previous convergence. However, the exploitation phase of the current Harris Hawks optimizer algorithm is poor. In the present research, an improved version of the Harris Hawks optimization algorithm, which combined HHO with Particle Swarm Optimization and named as ameliorated Harris Hawks optimizer algorithm, has been proposed to find the solution of various optimization problems such as nonlinear, non-convex and highly constrained engineering design problem. In the proposed research, the exploitation phase of the existing HHO algorithm is improved using a particle swarm optimization algorithm and its performance tested for CEC2005, CECE2017 and CEC2018 benchmark problems. Also, discrete algorithms such as FFT algorithms, convolution and image processing algorithm use multiply and accumulate (MAC) unit as a critical component. The efficiency of a MAC is mainly dependent upon the speed of operation, power dissipation and chip area along with the complexity level of the circuit. In this research paper, a power-efficient signed floating-point MAC (SFMAC) is proposed using universal compressor-based multiplier (UCM). Instead of having a complex design architecture, a simple multiplexer-based circuit is used to achieve signed floating output. The 8×8 SFMAC can take 8-bit mantissa and 3-bit exponent. And therefore, the input to the SFMAC can be in the range of $-(7.96875)_{10}$ to $+(7.96875)_{10}$. The design and implementation of the proposed architecture is done on the Cadence Spectre tool in GPDK 90 nm and TSMC 130 nm technologies. The analysis has proved that the proposed SFMAC architecture has consumed the least power than the recent MAC architectures available in the literature.

Keywords Floating-point MAC · UCM · Cadence · TSMC 130 nm · GPDK 90 nm

1 Introduction

Artificial intelligence and machine learning algorithms are getting popularity to solve many real-world optimization problems, which have continuous or discrete behaviour and constrained or unconstrained in nature [1, 2]. Difficulties arise while tackling the issues based on these types of characteristics using conventional approaches with mathematical or numerical programming, including sequential quadratic programming, quasi-Newton method, fast steepest, and conjugate gradient [3, 4]. There are several pieces of evidence, which shows that these all methods are not enough effective or efficient to deal many types of non-differentiable, non-continuous problem and also not applicable for large-scale real-world multi-modal problems [5]. Thus, a meta-heuristic algorithm has been considered. It is used to tackle so many issues which are mostly simple

✉ Cherry Bhargava
cherry.bhargava@lpu.co.in

[1] VLSI Design, School of Electronics and Electrical Engineering, Lovely Professional University, Phagwara, Punjab, India

[2] Department of Electronics and Communication Engineering, Jaypee University of Information Technology, Waknaghat, India

[3] Power System, School of Electronics and Electrical Engineering, Lovely Professional University, Phagwara, Punjab, India

[4] Schulich School of Engineering, University of Calgary, Calgary, AB, Canada

and easily implemented. In optimization, inhabitants-based techniques used to find some solution based on suboptimal and optimal that can be the same with the exact optimal value situated in its nearby point or neighbourhood. By generating a population set of individuals, the optimization process of the algorithm starts. In the population, each individual represents a candidate solution to the problem based on the optimization technique. By replacing the population of the best position of the current location, the population will be changed iteratively and generate a new population using some operators that are stochastic [6, 7]. This process of optimization continued until it can satisfy the maximum iteration. In recent years, there is a growing awareness and interest in an efficient, inexpensive meta-heuristic search algorithm for successfully solving various continuous and discrete real-world problems.

Further, no free lunch (NFL) theorem [8] logically suggested that none of the developed algorithms is universally fit for all kinds of optimization problems. According to the NFL theorem, it cannot consider an algorithm theoretically as the globally best type of optimizer for general purposes. Hence, the NFL theorem motivates penetrating and rising more effective algorithms based on optimization techniques.

Today's portable devices are capable of image filtering to face recognition, an audio signal enhancement to voice recognition, and gesture-based control to biometric authentication. All those functionalities are the applications of digital signal processing (DSP). A large number of mathematical operations are performed repeatedly and quickly on a series of data samples by DSP algorithms. Most operating systems and general-purpose microprocessors can successfully execute DSP algorithms. Because of power efficiency constraints, they are not suitable for use in portable devices such as PDAs and mobile phones. However, the rapid growth of portable electronics has introduced the significant challenges of low power and high throughput for VLSI design engineers. Among the other digital blocks, multiply and accumulate (MAC) unit plays a vital role while evaluating the performance of a DSP block. While performing convolution, filtering, or any DSP operations, it is always desired to use an efficient MAC unit. The efficiency of a MAC unit measured in terms of two factors:

- speed of operation
- overall power consumption [9, 10]

The underlying MAC architecture contains the main functional blocks as a multiplier, adder and register/accumulator. The multiplier performs the multiplication operation over the two input operands; the adder performs the addition of the result of the multiplier with the result of the previous cycle, and the register or accumulator stores the sum for the next cycle addition. Different approaches for multiplication and addition for MAC operations have been described in detail in the literature [11, 12]. The essential activity of MAC is to generate the product of two operands X_i and Y_i and add the result with the previously stored result from the last multiplication, as shown in Eq. (1).

$$F = \sum_{i=0}^{n-1} X_i Y_i \tag{1}$$

where i denotes the range of the values. The generalized block diagram of 8 × 8 bit MAC is shown in Fig. 1.

In recent years, researchers have developed different MAC architectures [9–17]. For example, a high-speed MAC architecture that promises with an optimized area is proposed in 2007 by Abdelgawad et al. [9]. It uses 4:2 compressor circuits to improve speed. A low-power high-throughput architecture is proposed in [10], where fixed-point MAC architecture is designed for a signed number. In [11], an improved version of tree-based Wallace tree multiplier architecture using the Booth recorder proposed. This proposed architecture reduces the latency and area of Wallace tree multiplier with the help of the Booth algorithm and compressor adders. In 2014, Luu et al. proposed an unsigned 32-bit multiplier for best timing performance with the optimized area [12]. In 2012, Deepak et al. [14] proposed a novel architecture for the multiplier. In 2013, Jagadees et al. proposed a novel architecture using a modified Wallace tree multiplier [15]. The implementation is done for 64 bits. In 2013, Francis et al. used the modified

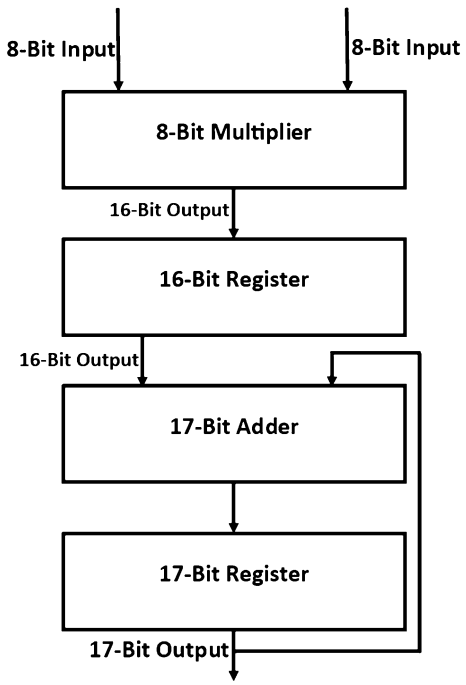


Fig. 1 Generalized block diagram of 8 × 8 bit MAC

Braun multiplier to implement a basic MAC unit [16]. The implementation was done on NCSim and RTL compiler. Warrier et al. proposed the low-power Baugh–Wooley multiplier-based unit in 2014 [17]. A pipeline-based architecture was introduced in this work. Split MAC architecture is explained by Xia et al. [13]. A technique to compress the partial product using 'interleaved adders' and a 'modified hybrid partial product reduction tree (PPRT)' schemes was proposed to enhance the speed of operation further. There are several architectures explained in the past by various designers. However, the architectures described in [9–13, 15–17] (90% of them in the literature) are designed either in Verilog or VHDL.

The main disadvantage of using HDL is that the basic blocks that are used while designing any architecture use predefined building blocks (standard PMOS–NMOS implementation). Because of this, even after using smart and efficient structural designs, the architecture lags in certain aspects. The main reason for such a shortcoming is the non-optimization of basic building blocks. In this research work, a full custom IC approach is adopted to design the internal blocks of the MAC architecture, yielding an efficient MAC in terms of power and delay. On the other hand, the MAC architecture also uses the universal compressor-based multiplier (UCM) for designing the proposed SFMAC architecture.

In recent years, various hybrid meta-heuristics search algorithm has been developed such as orthogonally designed adapted grasshopper optimization [18], hybrid crossover oriented PSO and GWO [19], imperialist competitive learner-based optimization [20], Barnacles mating optimizer [21], equilibrium optimizer [22], improved fitness-dependent optimizer algorithm [23], improved whale optimization algorithm [24], multi-strategy enhanced sine cosine algorithm [25], refined selfish herd optimizer [26], artificial ecosystem-based optimization [27], incremental Grey wolf optimizer, expanded Grey wolf optimizer [28], life choice-based optimizer [29], multi-objective heat transfer search algorithm [30], simplified salp swarm algorithm [31], self-adaptive differential artificial bee colony [32]. Along with this, few recent variants of HHO algorithm have been developed and have been applied to different science and engineering optimization problems including numerical and global optimization problems; for example, intensify Harris Hawks optimizer [IHHO] [33] has been used to engineering optimization problems, HHO-IGWO has been tested for CEC2005 benchmark problems [34], Swarm Intelligence based Harris Hawks optimization has been applied for spatial assessment of landslide susceptibility [35], quasi-reflected Harris Hawks optimizer has been applied for global optimization problems [36], dynamic Harris Hawks optimization with mutation mechanism has been applied for satellite image segmentation problem [37], and quadratic binary Harris Hawk optimization has been used for feature selection [38]. In the proposed research, a new nature-inspired hybrid optimization technique, hHHO-PSO, has been implemented and tested for various well-known benchmark functions, i.e. CEC2005, CEC2017, and CEC2018. The primary impression behind this kind of proposed optimization technique is encouraged from the cooperative-natural behaviour of the most intelligent bird, named Harris Hawks of its natural hunting behaviour and the escaping or avoiding nature of the prey (rabbit) [39]. Thus, a novel mathematical model, which is stochastic and meta-heuristic, is implemented in this research paper and applied to optimize low-power signed floating-point MAC architecture to tackle different types of optimization problems.

The remaining part of this paper is divided into the following sections: Sect. 3 describes the UCM architecture and its advantages over the Wallace tree multiplier. Section 4 describes the proposed SFMAC architecture in detail with the initial considerations and various essential building blocks. Section 5 describes the results and discussion along with the comparison with the existing architectures. Finally, the conclusion and future work are explained in Sect. 6.

2 Literature review

Optimization is a vast area of research, and research is going on very fast. The researchers are doing continuous work on different problems to implement various types of new techniques on diverse issues and are capable of successfully finding the results. The work is profitable to see the latest algorithms and also the algorithms with its hybrid form to mitigate any types of drawback in the present in exiting methods. In the proposed research, contemporary research papers are selected to investigate the shortfalls of existing algorithms. A brief review of literature pertaining to recent meta-heuristics search algorithm is depicted in Table 1.

In the recent year, several variants of Harris Hawks optimizer have been proposed such as IHHO [40], MOHHO [41] and quasi-reflected HHO [36]. Further, conventional HHO and its recently proposed variants have been applied to various engineering optimization problems; namely, Ahmad Abbasi et al. proposed the applications of the algorithm Harris Hawks optimizer to design the micro-channel heat sinks to decrease the generation to entropy. A variety of materials and liquids have likewise been assessed to decide the ideal structure of the micro-channel [42]. Mohamed Abd Elaziz et al. designed competitive chain-based HHO optimization algorithm to solve the global optimization problem with multilevel image threshold

Table 1 Recent meta-heuristics optimization algorithms

Year of development	Algorithm name	Main findings or conclusion relevant to the proposed research work
2020	Orthogonally designed adapted grasshopper optimization [OAGO] [18]	Orthogonally designed dapted grasshopper optimization was designed to solve the optimization problem. It was tested on 30 IEEE CEC2017 benchmarks to find the effectiveness of the meta-heuristic algorithm
2020	Hybrid crossover oriented PSO and GWO [HC-PSO-GWO] [19]	Hybrid PSO and GWO algorithms were designed to solve a global optimization problem
2020	Imperialist competitive learner-based optimization [ICLBO] [20]	Imperialist competitive learner-based optimization was implemented to solve the engineering design problem
2020	Barnacles mating optimizer [BMO] [21]	Barnacles mating optimizer was designed to solve the problem related to engineering optimization
2020	Equilibrium optimizer [EO] [22]	Equilibrium optimizer was created to solve optimization problems, and it was tested on 58 unimodal, multi-modal, and composition functions and three engineering problems
2020	Improved fitness-dependent optimizer algorithm [IFDOA] [23]	Improved fitness-dependent optimizer algorithm was designed and tested on CEC2019 to validate its feasibility to a real-world problem
2020	Improved whale optimization algorithm [IWOA] [24]	Improved whale optimization algorithm was designed using the mechanism of a joint search to solve the global optimization problems
2020	Multi-strategy enhanced sine cosine algorithm [MSESCA] [25]	Multi-strategy enhanced sine cosine algorithm was designed to an engineering design problem in the real-world and improve global optimization
2020	Refined selfish herd optimizer [RSHO] [26]	Refined selfish herd optimizer designed to solve the global optimization problem
2020	Intensify Harris Hawks optimizer [IHHO] [40]	Hybrid Harris Hawks optimizer combined with SCA implemented to get solutions to numerical and engineering optimization problems
2019	Artificial ecosystem-based optimization [AEBO] [27]	A novel meta-heuristic optimizer, artificial ecosystem-based optimization, was implemented to resolve the problem related to unidentified search space
2019	Incremental Grey wolf optimizer and expanded Grey wolf optimizer [I-GWO and Ex-GWO] [28]	Incremental GWO and expanded GWO were the improved versions of GWO, which are used to solve the global optimization problem
2019	Life choice-based optimizer [LCBO] [29]	Life choice-based optimizer considered to resolve optimization problems, and it tested on six CEC-2005 functions
2019	Multi-objective heat transfer search algorithm [MHTSA] [30]	The multi-objective technique invented to get solutions to the problem related to the truss method
2019	Simplified salp swarm algorithm [SSSA] [31]	Simplified salp swarm algorithm created to resolve the optimization problem, and it was verified on 23 standard benchmarks to check the feasibility of this technique
2019	Self-adaptive differential artificial bee colony [SA-DABC] [32]	The new method was designed and tested on 28 Nos. of standard benchmark problem to solved global optimization problems

problems, and it was verified in 36 IEEE CEC 2005 standard benchmarks and 11 natural grey-scale images [43]. Huiling Chen et al. designed a photovoltaic cell which was used to identify the parameter using Harris Hawks optimizer algorithm with chaotic drifts [44]. Dalia Yousri et al. designed an artificial ecosystem-based optimization technique which is used to recognize different types of parameter configuration of PV model [45]. Saravanakumar and Dr. D. Chandra Mohan have solved crossover line balanced problem using Harris Hawks optimization technique to select good quality chromosomes to identify the presentation of the sequential construction system [46]. Krishan Arora et al. designed an improved Harris Hawks optimizer which is used to analysis the sensitivity of load frequency control problem including wind power penetration [47]. Kashif Hussain et al. applied Harris Hawks optimization problem to solve the problem including optimum and high-dimensional power flow [48]. Pei Du et al. designed hybrid multi-objective Harris Hawks optimizer (MOHHO) for daily check PM 2.5 and PM 10 forecasting with high stability and accuracy [41]. Mohammad Zohrul Islam et al. designed single- and multi-objective optimum power flow including emission of environment [49]. Qian Fan et al. proposed a new quasi-

reflected Harris Hawks optimizer to solve the global optimization problems [36]. In recent research studies pertaining to optimization algorithm, it has been reported that swarm intelligence optimization have some drawbacks which need to be solved [50]. Another important concern in swarm intelligent algorithm is regarding exploration, which is inopportune measure without theoretical guidance. In practical application, it creates serious problem. This motivated our attempts to propose yet another memetic algorithm for the designing of low-power signed floating-point MAC architecture.

3 Mathematical formulation of ameliorated Harris Hawks optimizer

In this paper, the HHO algorithm includes exploitative and exploratory phases inspired by surprise pounce, the nature of exploration of prey, and different strategies based on the attacking phenomenon of Harris Hawks. It is one of the gradient-free and inhabitants-based algorithms for optimization techniques, which can be used to formulate an optimization problem. HHO has some major phases, such as the phase of exploration, conversion from exploration to exploitation and stage of exploitation.

3.1 Phase of exploration

In the proposed technique, Harris Hawks settle randomly on some positions and wait to detect and locate the prey based on two types of strategies. They are considering the equal chance w for each balancing approach based on the locations of another member of the family. Equation (2) shows that when $w \geq 0.5$, it determines the average location of Harris Hawks and when $w < 0.5$, it balance on random positions.

$$H(\text{iter}+1) = \begin{cases} H_{\text{rand}}(\text{iter}) | H_{\text{rand}}(\text{iter}) - 2e_{s_2} H(\text{iter})|; & w \geq 0.5 \\ [H_{\text{rabbit}}(\text{iter}) - H_m(\text{iter})] - e_{s_3}[L_{\text{Bound}} + e_{s_4}(U_{\text{Bound}} - L_{\text{Bound}})]; & w < 0.5 \end{cases} \tag{2}$$

Equation (2) is used to find out the best value of different locations and the component of random scale, which is based on the range of the upper and the lower bound of the variables, where e_{s3} is the scale coefficient, which is used to increase the rule of nature randomly, while the value of e_{s4} is mostly near about one, and this pattern is distributed similarly, including the average position. The ordinary location of the Harris Hawks can be achieved by using Eq. (3).

$$H_m(\text{iter}) = \frac{1}{N}\sum_{i=1}^{N} H_i(\text{iter}) \tag{3}$$

where e_{s1}, e_{s2}, e_{s3}, e_{s4} and w are arbitrary numbers in between (0, 1) and these are improved in each iteration, $H_{\text{rabbit}}(\text{iter})$ = rabbit's position, $H_{\text{rand}}(\text{iter})$ = number of Harris Hawks are selected from recent population, N = total number of Harris Hawks.

The initial position of the search agents has been generated randomly within the lower and upper bounds of the search space using $L_{\text{Bound}} + \text{rand} \times (U_{\text{Bound}} - L_{\text{Bound}})$, and then modified positions can be generated using Eq. (2) to determine the best value of different locations. According to this rule, the HHO algorithm adds the movement of scale length up to the lower bound L_{Bound}. Randomization of the scaling coefficient has been taken into consideration to provide more exploration in different sections of the search space. There is also a possibility to create other types of updated rules, but here we have developed the simple rule, which can copycat the nature of the hawks.

3.2 Conversion from the phase of exploration to the phase of exploitation

In this algorithm, based on the HHO optimization technique, there can be transference from exploration condition to exploitation condition. After that, alteration between various types of nature is based on exploitative behaviour, based on the prey's avoidance energy. Due to this avoidance behaviour, it decreases the energy of the prey. The equation based on the behaviour of the energy of the prey is given below:

$$\text{EG} = 2\text{EG}_0 \times \left(1 - \frac{\text{iter}}{\text{iter}_{\max}}\right) \tag{4}$$

where EG = Avoidance energy of the prey, EG_0 = Initial condition of the energy, $\text{iter}_{\max}$ = Maximum iteration.

Here, for this proposed algorithm, EG_0 changes randomly in between the interval (− 1, 1) for each number of iteration. If the number of iteration reduces from 0 to − 1, that means the nature of the physically flagging behaviour of the rabbit is represented, and if the number of iteration rises from 0 to 1, that means the establishment nature of the rabbit is represented. The avoidance energy EG is dynamic, and it also tends to go down in a decreasing manner during the running condition of the iterations. If the avoidance energy $|\text{EG}| \geq 1$, that means to explore the location of the rabbit, the Harris Hawks search different kinds of regions. Hence, this algorithm can perform the phase of exploration and if $|\text{EG}| < 1$, that means during the stage of exploitation, the nature of the algorithm is to exploit the nearby solutions. That means we can say exploration occurs when $|\text{EG}| \geq 1$, while the exploitation occurs when $|\text{EG}| < 1$.

- *Phase of exploitation*

In this section, the performance of Harris Hawks is to behave like a surprise attack by attacking nature upon projected prey identified in the previous phase. Though prey often tries to escape or avoid dangerous conditions. Hence, various types of hunting styles happen in real circumstances. According to avoiding and escaping the nature of the victim and the policies of hunting of Harris Hawks, there are four types of probable strategies that are presented in the optimization technique based on HHO to implement the stage of attacking strategy. The main behaviour of such kinds of preys is to try to avoid or escape from this kind of situation, based on threatening conditions. Let e_s be the probability that the prey can escape successfully when $e_s < 0.5$ or cannot escape successfully when $e_s \geq 0.5$ before surprise the pounce. The Harris Hawks will give their performance soft or hard encircle to catch the prey at any conditions whatever the prey does. That means hawks will surround the prey from the different types of various directions softly or hardly, which is depending upon the prey's energy itself. In real-time operation or for the real situation by the performance of the surprise attack, the Harris Hawks get nearer to the projected prey to raise their probabilities in attacking and after that killing the rabbit. After some time, avoiding or escaping prey must lose their energy; therefore, the Harris Hawks increase the encircle process to catch the shattered prey smoothly. The constraint EG is applied to implement this tactic and allows HHO to change over between hard and soft encircle processes. The soft encircle occurs for $|\mathrm{EG}| \geq 0.5$, and hard encircle occurs for $|\mathrm{EG}| < 0.5$.

- *Soft encircle*

If $e_s \geq 0.5$ and $|\mathrm{EG}| \geq 0.5$, that means the rabbit still has its enough energy through which it can try to avoid or escape by jumps randomly but is unable to go forward into its safe condition. This soft encircle occurred by Harris Hawks, for which the victim rabbit tends to more and more exhaust and, after that, execute the surprise attack. This kind of natural behaviour is demonstrated by some rules and given below:

$$H(\mathrm{iter}+1) = \Delta H(\mathrm{iter}) - \mathrm{EG}|\mathrm{KH}_{\mathrm{rabbit}}(\mathrm{iter}) - H(\mathrm{iter})| \tag{5}$$

$$\Delta H(\mathrm{iter}) = H_{\mathrm{rabbit}}(\mathrm{iter}) - H(\mathrm{iter}) \tag{6}$$

where $\Delta H(\mathrm{iter})$ = difference between the iteration based on the present location and the vector-based on the position of that victim rabbit.

e_s = random number inside (0, 1), $K = 2(1 - e_s)$, this signifies the strength of jump randomly throughout the procedure of avoiding and escaping. The value of K randomly changes for each iteration to simulate the behaviour of the motion of the rabbit.

- *Hard encircle*

If $e_s \geq 0.5$ and $|EG| < 0.5$, that means the rabbit is so shattered, and at this condition, it has small avoidance or avoidance energy. In this situation, the Harris Hawks perform scarcely to enclose the projected victim to do finally shock attack. At this condition, the current locations are changed and updated by using Eq. (7)

$$H(\mathrm{iter}+1) = H_{\mathrm{rabbit}}(\mathrm{iter}) - \mathrm{EG}|\Delta H(\mathrm{iter})| \tag{7}$$

3.2.1 Soft encircle with advanced fast dives

If still $|\mathrm{EG}| \geq 0.5$, but at this time $e_s < 0.5$, that means the rabbit has sufficient energy to avoid or escape successfully, and before the surprise attack, a soft encircle created. This process is much more intelligent than the previous process.

The levy flight (LFT) conception is applied in the HHO algorithm by which we can understand the mathematical model of the leapfrog movements and the patterns of avoiding or escaping prey. It helps to detect the LFT based on such patterns, which can be identified as the activity of chasing such kinds of animals like sharks and monkeys. Hence, the LFT-based patterns can be utilized in this technique of the phase of the HHO algorithm. It also developed to mimic the actual zigzag movements of victims (rabbits) during the phase of escaping and asymmetrical abrupt and advanced fast dives of Harris Hawks around avoiding the prey. Harris Hawks perform numerous teams rapidly dives around the prey (rabbit) and then try to increasingly correct their position and location as well as the directions concerning the rabbit's pretended motion. Observations also maintain this type of mechanism in the real world in other reasonable situations in the behaviour of nature. With the help of this nature, the hawks can increasingly select the best probable dive towards the rabbit when they desire to catch the rabbit in economic situations. So, for better performance of a soft encircle, the Harris Hawks can decide their next movement (MN) based on a rule given in Eq. (8):

$$M_{\mathrm{N}} = H_{\mathrm{rabbit}}(\mathrm{iter}) - \mathrm{EG}|\mathrm{KH}_{\mathrm{rabbit}}(\mathrm{iter}) - H(\mathrm{iter})| \tag{8}$$

After that, they can relate the probable result of such kind of movement to the previous dives, which detect that it will be better dive or not. If they see that the performance (motion) of the prey (rabbit) is more deceptive, which means they also start to accomplish abrupt, rapid, and irregular dives when oncoming the rabbit. The dive is based upon the LFT patterns, which follow the given rule in Eq. (10) [51].

$$P = M_{\mathrm{N}} + R_{\mathrm{S}} \times \mathrm{LF}_{\mathrm{T}}(D_{\mathrm{IM}}) \tag{9}$$

where D_{IM} = problem's dimension and R_{S} = size of the random vector by size $1 \times D_{\mathrm{IM}}$

$$\mathrm{LF}_T(x) = 0.01 \times \frac{\alpha \times \delta}{|\mu|^{\frac{1}{\gamma}}} \tag{10}$$

$$\delta = \left(\frac{\Gamma(1+\gamma) \times \sin(\frac{\pi\gamma}{2})}{\Gamma(\frac{1+\gamma}{2}) \times \gamma \times 2^{(\frac{\gamma-1}{2})}} \right)^{\frac{1}{\gamma}} \tag{11}$$

where α, μ are denoted as such kind of value randomly in between (0, 1) and γ set to 1.5, which represent as default constant.

So, including all of these, the actual and final strategy for updating the actual location of Harris Hawks in the phase of soft encircle can be achieved by Eqs. (12) and (13), which are given below.

$$H(\mathrm{iter}+1) = \{M_N; \text{ if } F(M_{\mathrm{N}}) < F(H(\mathrm{iter})) \tag{12}$$

$$H(\mathrm{iter}+1) = \{P; \text{ if } F(P) < F(H(\mathrm{iter})) \tag{13}$$

3.2.2 Hard encircle with advanced fast dives

If $|\mathrm{EG}| < 0.5$ and $e_s < 0.5$, at this moment, the prey has insufficient energy to escape or avoid, and a hard encircle is created before the surprise attack to catch and kill the rabbit. In this situation, the Harris Hawks try to reduce the distance with the escaping rabbit. So, this rule can be performed based on a hard encircle condition followed by Eqs. (12) and (13), where the value of $\mathrm{M_N}$ and P can be obtained using the new rule in Eqs. (14) and (15) which contain the next position for the next iteration.

$$M_{\mathrm{N}} = H_{\mathrm{rabbit}}(\mathrm{iter}) - \mathrm{EG}|\mathrm{KH}_{\mathrm{rabbit}}(\mathrm{iter}) - H_{\mathrm{m}}(\mathrm{iter})| \tag{14}$$

$$P = M_{\mathrm{N}} + R_{\mathrm{S}} \times \mathrm{LF}_{\mathrm{T}}(D_{\mathrm{IM}}) \tag{15}$$

where $H_{\mathrm{m}}(\mathrm{iter})$ can be obtained from Eq. (2).

3.3 Particle Swarm Optimizer

An airborne particle in a search space can mimic with the help of the PSO algorithm and moving towards to catch optimal solutions in the global region. The particle of PSO [52] is represented by

$$P_r \in [x, y], \quad r = 1, 2, 3 \ldots D_{ia} \text{ and } x, y \in R_N. \tag{16}$$

In starting condition, there are the position and velocity of each particle, which are initialized randomly in nature. Each of the particles should maintain its best position in the local region as well as the global region.

Equation (17) is used to appraise the velocity and the location among all of the existing elements.

$$\begin{aligned} \mathrm{vel}_i(\mathrm{iter}+1) = {} & \omega \times \mathrm{vel}_i(\mathrm{iter}) + c_1 \\ & \times r_1(PL_{\mathrm{BEST}} - X_i(\mathrm{iter})) + c_2 \\ & \times r_2(\mathrm{GL}_{\mathrm{BEST}} - X_i(\mathrm{iter})) \end{aligned} \tag{17}$$

$$X_i(\mathrm{iter}+1) = X_i(\mathrm{iter}) + \mathrm{vel}_i(\mathrm{iter}+1) \tag{18}$$

where vel_i = velocity of the particle, X_i = position of the particle, $\mathrm{PL_{BEST}}$ = best location of the particle in the local region, $\mathrm{GL_{BEST}}$ = best location of the particle in the global region, r_1 and r_2 are the random number between the limits [0, 1], c_1 and c_2 are used as a leaning factor,

where ω the inertia weight which regulates how much a particle can hold its current velocity in the next iteration. Suitable selection of inertia weight can deliver the particles with a balance between exploitation and exploration capability.

- *Complexity of computation*

The complexity of computation of this proposed algorithm is mainly subject to three major processes, such as at first initialization, after that evaluation of fitness and at last updating of Harris Hawks. It must be noted that I denotes number of Harris Hawks and the complexity of computation of the process of initialization is $U(I)$ For the updating mechanism, the complexity of computation is $U(\mathrm{iter}_{\max} \times I) + U(\mathrm{iter}_{\max} \times I \times D_{\mathrm{IM}})$, which is collected for penetrating for the best position and again update the position vector of all the Harris Hawks, where $\mathrm{iter}_{\max}$ represents the maximum iteration and the dimension of the problem represents D_{IM}. Thus, the complexity of computation of this proposed algorithm is $U(I(\mathrm{iter}_{\max}) + \mathrm{iter}_{\max} D_{\mathrm{IM}} + 1))$.

In the proposed hybrid HHO-PSO algorithm, the steps of PSO are applied sequentially after the HHO algorithm to improve the exploitation and exploration to further extent. The pseudocode of the proposed hybrid optimizer has been shown below.

4 Test data for validation

To validate the performance of the proposed hybrid HHO-PSO optimizer, the CEC2005, CEC2017 and CEC2018 benchmark functions have been taken into consideration [72, 73]. The test data for unimodal, multi-modal and fixed dimensions benchmark functions are shown in Appendices 1, 2 and 3, respectively. The pseudocode of the proposed hybrid optimizer is shown in Fig. 2. The overall pseudocode of the proposed optimizer has been divided into two sections. The foremost section represents the steps of Harris Hawks optimizer using Eqs. (2)–(15), and the later section represents the steps of Particle swarm optimizer, which are represented in Eqs. (16)–(18). The steps of the

PSEUDO code of the proposed hybrid algorithm

INPUTS: The population size is taken as N and maximum iteration number is taken as $iter_{\max}$

OUTPUTS: The position of prey (rabbit) and its value of fitness

Initialization of random population $X_i(i=1,2,3.....,N)$

While (the stopping condition not met) **DO**

Calculation of the fitness value of Harris hawks

Set the parameter H_{rabbit} as the best position of the prey (rabbit)

for (each Harris hawks (H_i)) DO

$EG_O = 2rand() - 1, K = 2(1 - rand())$ → Update energy at initial condition EG_O and K

Update EG using equation number (4)

if $|EG| \geq 1$ **then** → **Phase of Exploration**

Update the position vector using equation (1) and (2)

if $|EG| < 1$ **then** → **Phase of Exploitation**

if $(e_s \geq 0.5)$ and $|EG| \geq 0.5$ **then** → Soft encircle

Location vector updated using equation number (5)

else if $(e_s \geq 0.5)$ and $|EG| < 0.5$ **then** → **Hard encircle**

Location vector updated using equation number (7)

else if $(e_s < 0.5)$ and $|EG| \geq 0.5$ **then** → **Soft encircle with advanced fast dives**

Location vector updated using equation number (12) and (13)

else if $(e_s < 0.5)$ and $|EG| < 0.5$ **then** → **Hard encircle with advanced fast dives**

Location vector updated using equation number (14) and (15)

Initialize the X_i ← Location Vector

for i=1 to $iter_{\max}$

for each particle x in X

DO $fx = f(x)$

if $fx < PL_{BEST}$

$xbest = x$

end

end

GL_{BEST}=best x in X

for each particle $x \in X$

DO $vel_i(iter+1) = \omega \times vel_i(iter) + c_1 \times r_1(PL_{BEST} - X_i(iter)) + c_2 \times r_2(GL_{BEST} - X_i(iter))$

$X_i(iter+1) = X_i(iter) + vel_i(iter+1)$

end

DO Location Vector ← X_i

Return H_{rabbit}

Fig. 2 Pseudocode for proposed hybrid optimizer

proposed hybrid algorithms are described in the form of a flow chart in Fig. 3. The upper half of the flow chart represents the major steps of Harris Hawks optimizer, which included phase of exploration, phase of exploitation, hard encircle, soft encircle with advanced fast dived and hard encircle with advanced fast dived. After implementation of hard encircle with advanced fast dives using Eqs. (14) and (15), the updated final positions are further supplied to the swarm position vector, which is further updated using steps of positions and velocity updation using Eqs. (17) and (18) of particle swarm optimization. The finally obtained position has been supplied to the fitness function to calculate the final fitness value.

5 The UCM architecture

Although the Wallace tree multiplier is much faster than the array multiplier [53], it requires a large number of adders. Moreover, the Wallace tree multiplier is highly

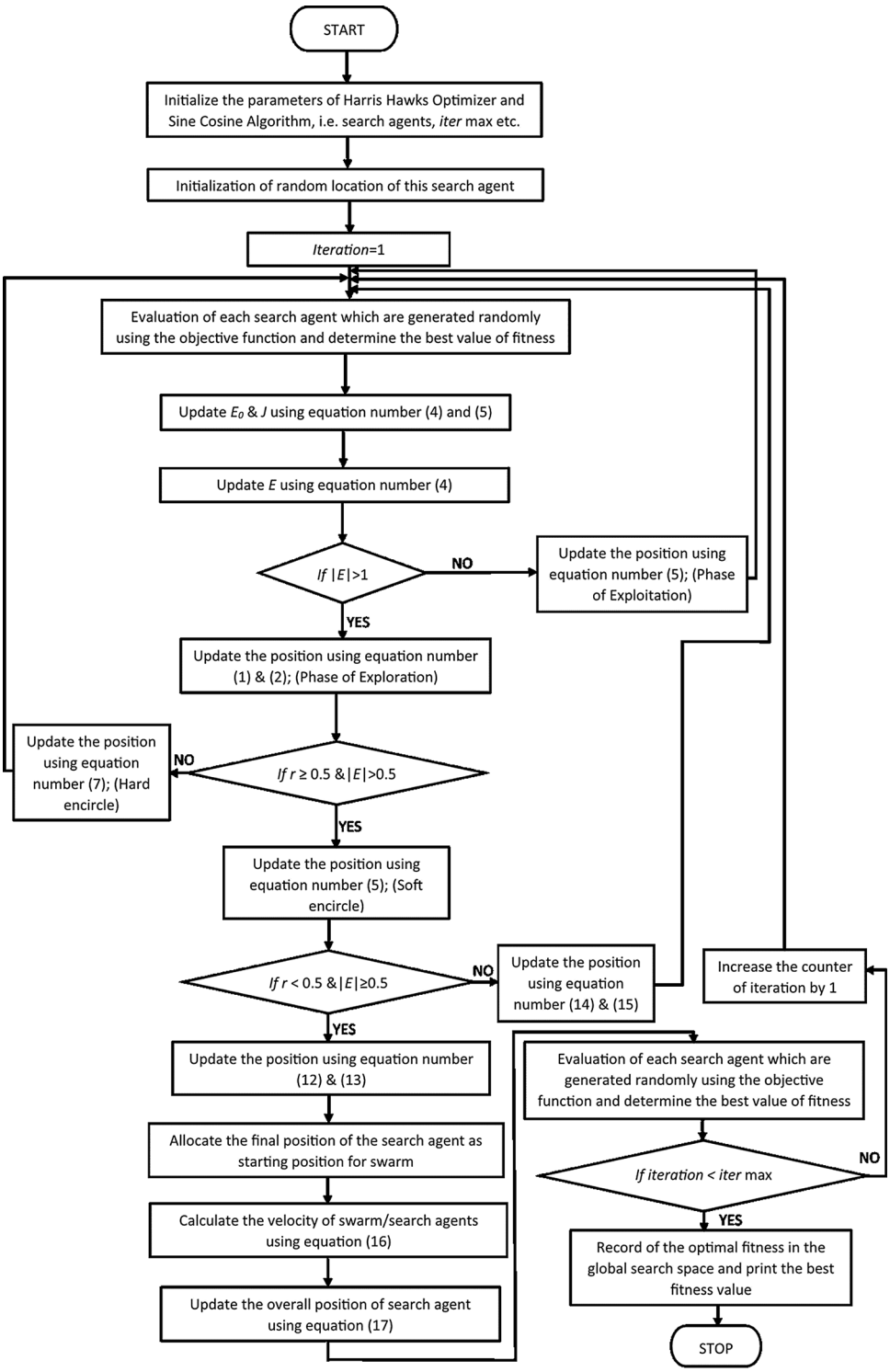


Fig. 3 Flow chart of the hHHO-PSO algorithm

irregular and complicated. So, in order to overcome the irregular structure, several modified Wallace tree multipliers are proposed in the literature [11, 12, 53–60]. All these multiplier architectures are based upon the Wallace tree algorithm. Hence, replacing the Wallace tree algorithm may further improve the result of the multiplier. Another important point here is, instead of using traditional Wallace tree adder, compressor circuits such as 3:2 compressors or 4:2 compressors can be used for partial product addition. But as there is a possibility of using the same compressor again and again for doing addition (same as Wallace tree addition), the same would not be much effective. The

universal compressor-based multiplier (UCM) architecture is used for the design of the proposed SFMAC architecture considering the disadvantages of the Wallace tree multiplier, as mentioned earlier, where an N:1 bit size universal compressor algorithm is used for the multiplication operation. It is claimed that the UCM produces the result with better efficiency than any other multiplier proposed in the literature [61].

6 Proposed signed floating-point MAC (SFMAC) architecture

The proposed MAC architecture mainly focuses on the signed architecture based on the synchronized block enabling with effective pipelining technique. The block enabling is a power saver technique that activates a circuit momentarily, and for the rest of the time, the circuit is inactive. Because of this simple phenomenon, most of the energy/power can be saved. Moreover, the main reason for introducing synchronization is to avoid unnecessary loss of data. Due to the non-availability of proper synchronization, the data under process in the preceding block may be lost while transferring the same to the next block. Thirdly and most importantly, the pipeline processing is the utmost requirement of the digital system as it increases the system's efficiency tremendously. As multiplier and the adder are the two core blocks, a detailed analysis done while choosing the appropriate circuits. In comparison, it is selecting the adder, the key importance given to the delay and power-efficient design. The adder circuit proposed by Bhattacharyya et al. in 2014 is used [62] in the proposed SFMAC design to address the issue

of power dissipation. As discussed earlier, the array multiplier is the simplest multiplier algorithm, but it produces very high delay in comparison with the conventional Wallace tree multiplier. On the other hand, the rectangular styled Wallace tree multiplier is faster than conventional Wallace tree multiplier as it divides the partial products into two groups [57]. But the irregular structure is the most significant disadvantage of rectangular styled Wallace tree multiplier. The novel UCM architecture, on the other hand, has a regularity in structure and performs better in terms of delay in comparison with the Wallace tree multiplier. Therefore, the novel UCM architecture is chosen as the multiplier for the proposed design.

A multiplexer-based MAC architecture is proposed in this paper, capable of performing MAC operation on signed floating-point inputs. For this, a novel input-data format is recommended, which takes 9-bit binary data with the MSB as the sign bit and 4-bit exponential input with the MSB as the exponential sign bit. Therefore, the size of the novel input-data format is 13 bits. Moreover, the SFMAC architecture, with its various building blocks, uses multiplexer circuits rigorously for selecting among a positive or negative number.

6.1 Initial considerations

The architecture uses sign-magnitude as well as 2's complement representations to represent positive as well as negative numbers (including exponent terms). The overall inputs and output of SFMAC are represented in sign-magnitude form, whereas for internal calculations, the same inputs are converted to 2's complement form. The final output of the proposed MAC architecture (MAC output) is 16 bits along with one sign bit. The inputs to the SFMAC are two 8-bit binary numbers arranged in a format, as shown in Fig. 4. The overall size of each input of the SFMAC representation is 13 bits, in which two bits reserved for the sign bits of the number, and it is the exponent. The sign bit can be '0' or '1' based on positive or negative number representation. Remaining eleven bits are used for an 8-bit binary representation and 3-bit exponent representation in binary. One crucial point here to note is that the 3rd bit of the exponent in binary representation is by default made as '0' because to represent a 2-bit number in 2's complement form it requires 3 bits. The range of 2's complement representation is shown in Eq. (19):

$$-(2^{n-1})to + (2^{n-1} - 1) \tag{19}$$

where 'n' is the number of bits.

Therefore, in this architecture, the exponent term can range from '− 4' to '+ 3'. The input numbers can have a range from $-(0.11111111)_2 \times 2^{+3}$ to $+(0.1111\ 1111)_2 \times 2^{+3}$. Hence, the range of the inputs of the proposed SFMAC architecture in the decimal number system is from $-(7.96875)_{10}$ to $+(7.96875)_{10}$. The inputs to the SFMAC architecture should be entered in decimal point only. For example, instead of providing the inputs to the SFMAC as $(001)_2$ and $(010)_2$, the numbers should be

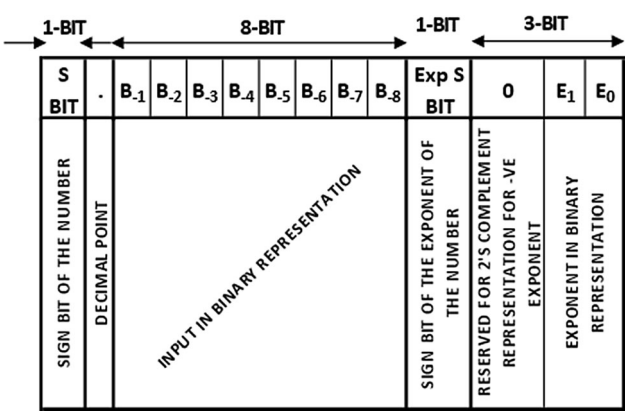


Fig. 4 Input format representation of SFMAC

entered as $(0.00100000)_2 \times 2^{+3}$ and $(0.0100000)_2 \times 2^{+3}$. Similarly, $(101)_2$ and $(10)_2$ should be represented as $(0.10100000)_2 \times 2^{+3}$ and $(0.10000000)_2 \times 2^{+2}$, respectively, to process it through the SFMAC.

The primary contents of the SFMAC architecture are exponential adder (EA), 8-bit multiplier, 16-bit register, exponent comparator circuit (ECC), exponent shifter circuit (ESC), 16-bit adder and 2:1/4:1 multiplexer of

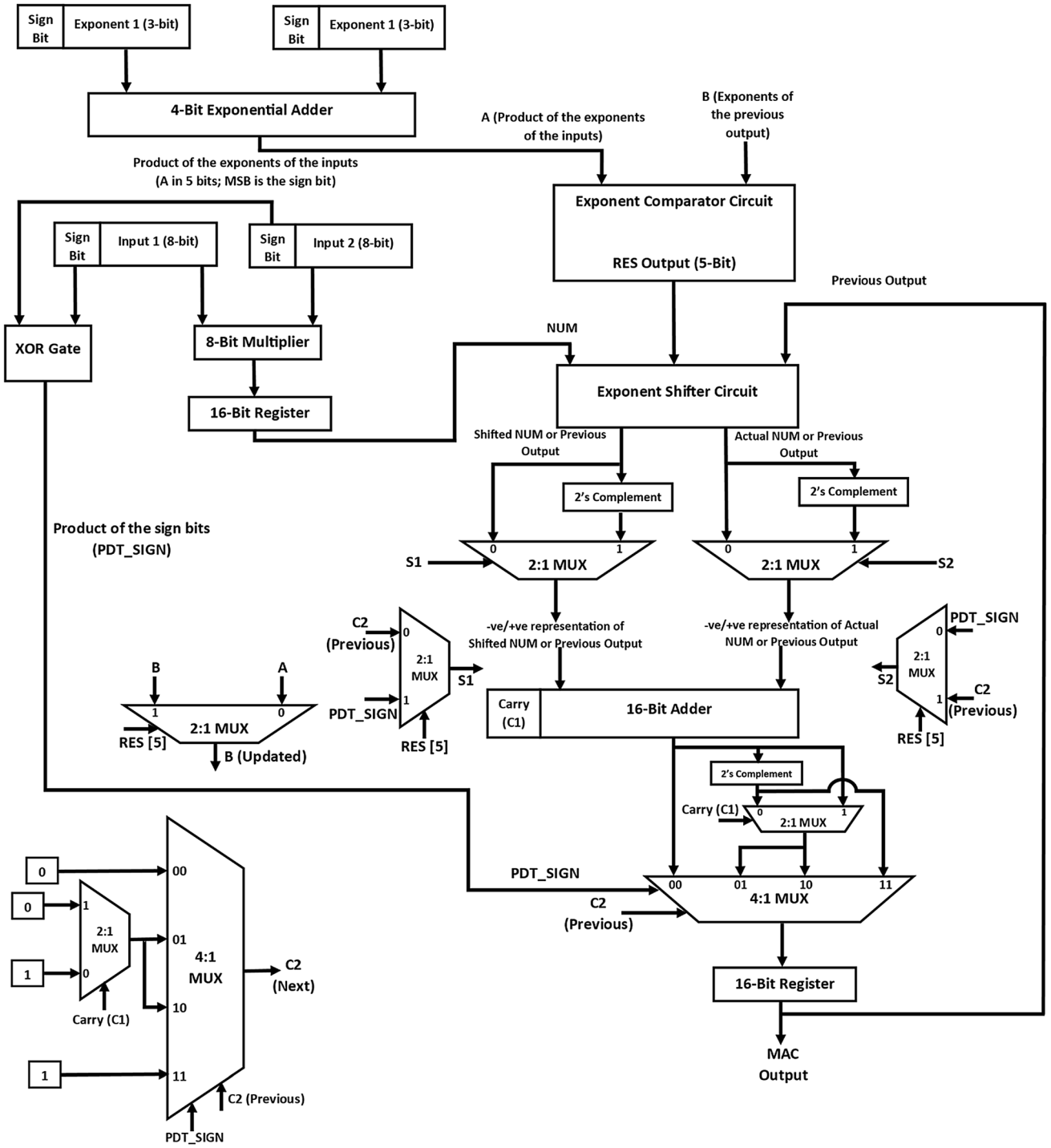


Fig. 5 The novel SFMAC architecture

different sizes. The overall architecture of SFMAC is shown in Fig. 5.

6.2 Exponential adder (EA)

The EA block performs multiplication of the exponents of the inputs. Therefore, it basically adds the exponents (as $2^n \times 2^m = 2^{(n+m)}$). Though the inputs to the EA block are

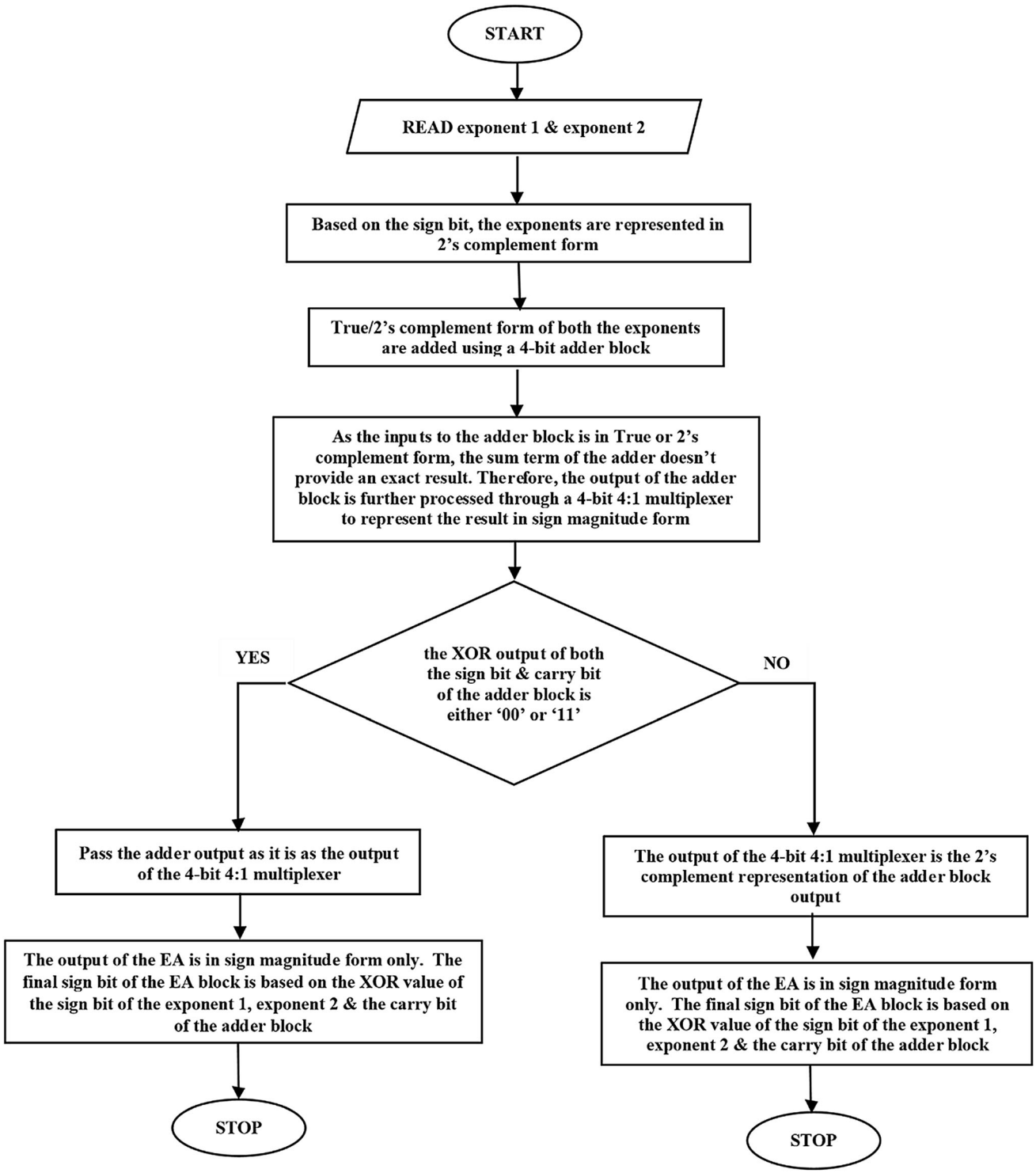


Fig. 6 Operation of EA block

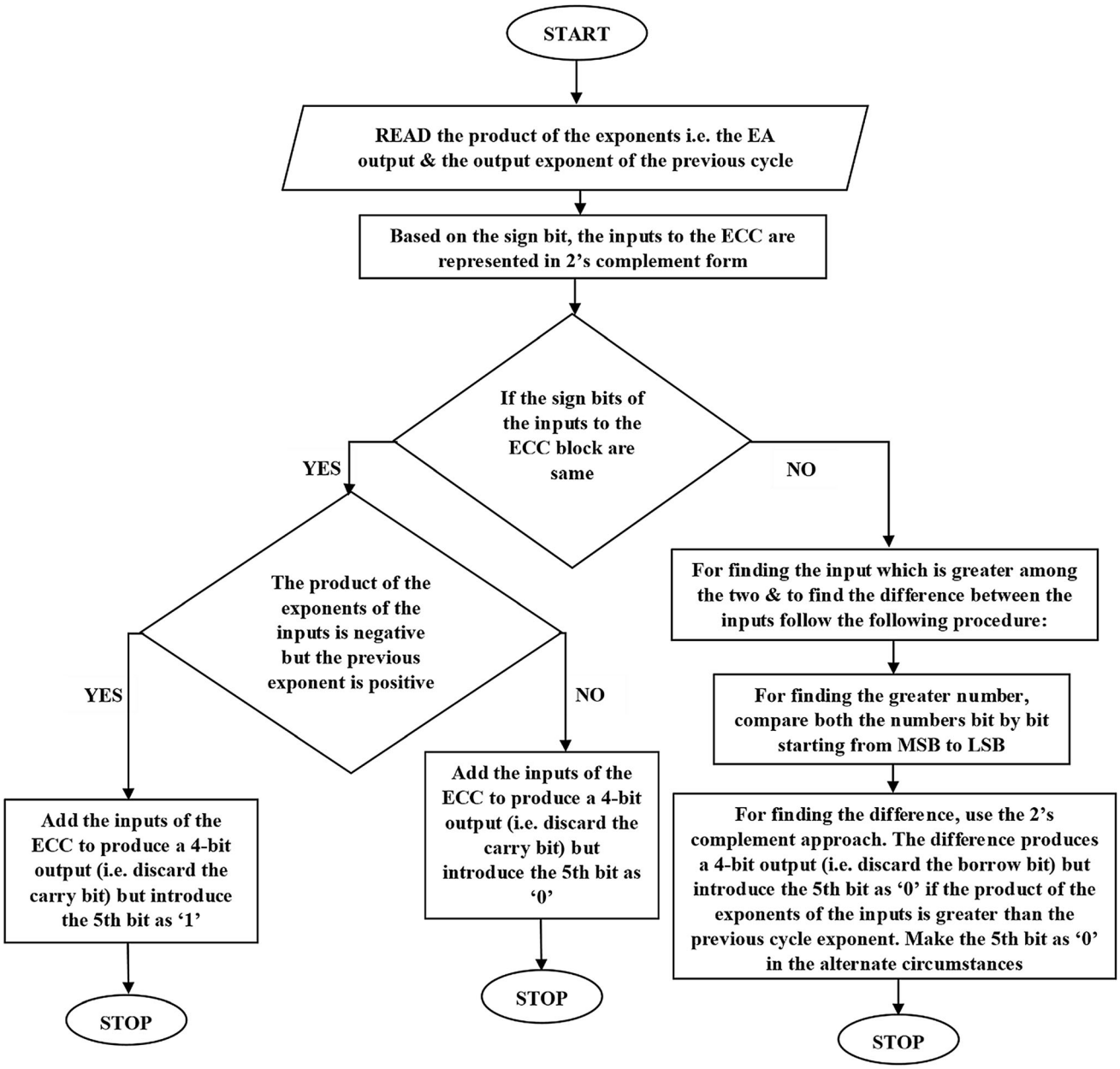


Fig. 7 Operation of ECC block

Table 2 Test results for unimodal benchmark functions using the hHHO-PSO algorithm

Functions	Mean	SD	Best	Worst	Median	*p* value
F1	7.62E−98	3.11E−97	9.40E−118	1.67E−96	2.30E−103	1.73E−06
F2	2.49E−51	1.20E−50	9.00E−58	6.56E−50	9.05E−55	1.73E−06
F3	7.38E−75	4.02E−74	3.39E−98	2.20E−73	8.85E−88	1.73E−06
F4	1.23E−47	6.73E−47	4.52E−57	3.69E−46	2.82E−53	1.73E−06
F5	0.007324	0.008526	0.000442	0.035424	0.003318	1.73E−06
F6	0.000144	0.00025	5.53E−07	0.001333	6.96E−05	1.73E−06
F7	0.000177	0.000174	1.02E−05	0.00077	0.000127	1.73E−06

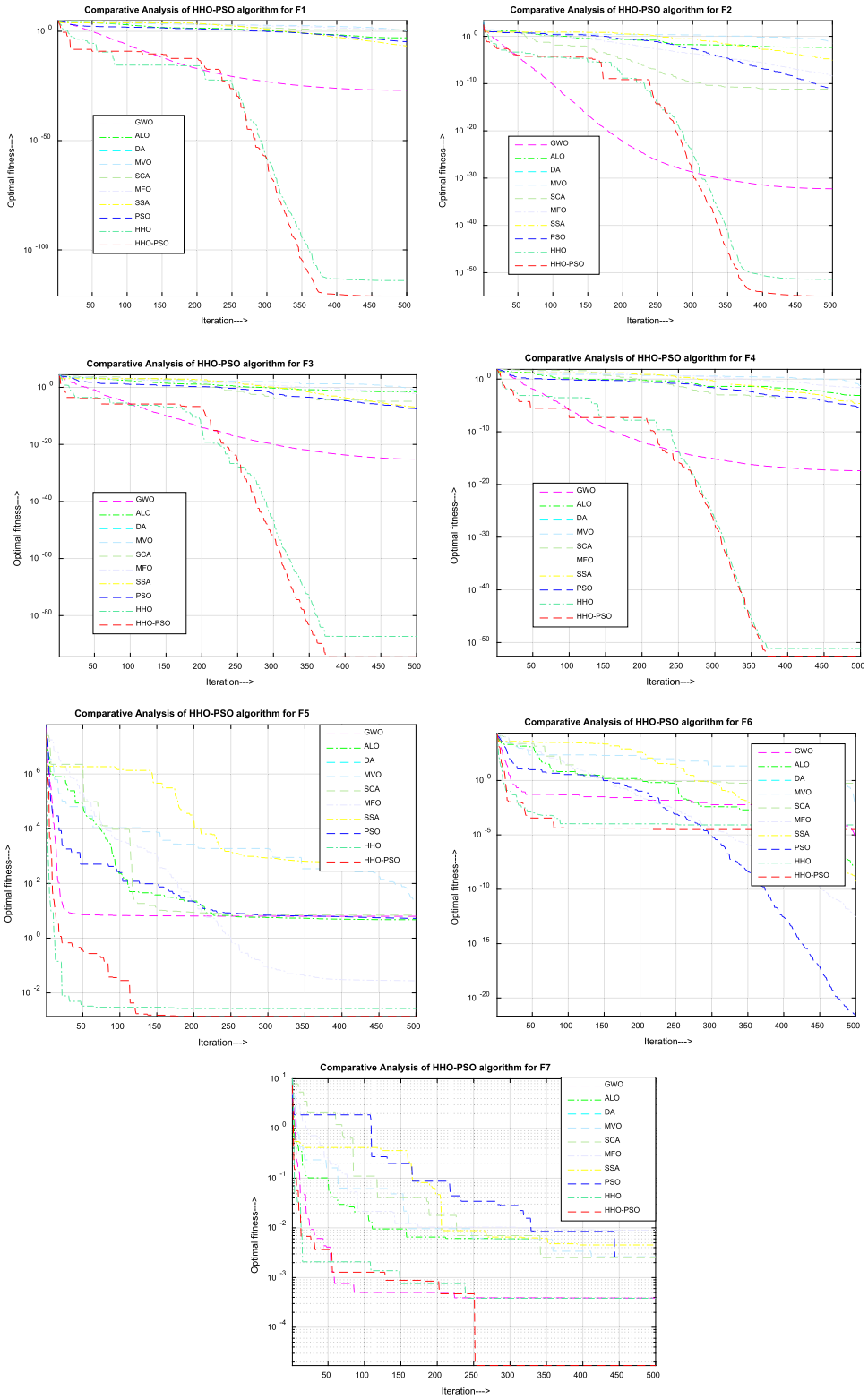


Fig. 8 Convergence curve of hHHO-PSO with GWO, ALO, DA, MVO, SCA, MFO, SSA, PSO and HHO for unimodal benchmark functions

of 4 bits each (including one sign bit), it produces the result in 5 bits as the addition of two 2-bit numbers can produce a maximum of 3-bit result and for representing a 3-bit binary number in 2's complement form, it requires 4 bits. On the other hand, the MSB bit (i.e. 5th bit) is the result's sign bit. A PED-latch block pair is used to make the EA block synchronized at the output of every output bits. The flow chart explains the operation of the EA block in Fig. 6.

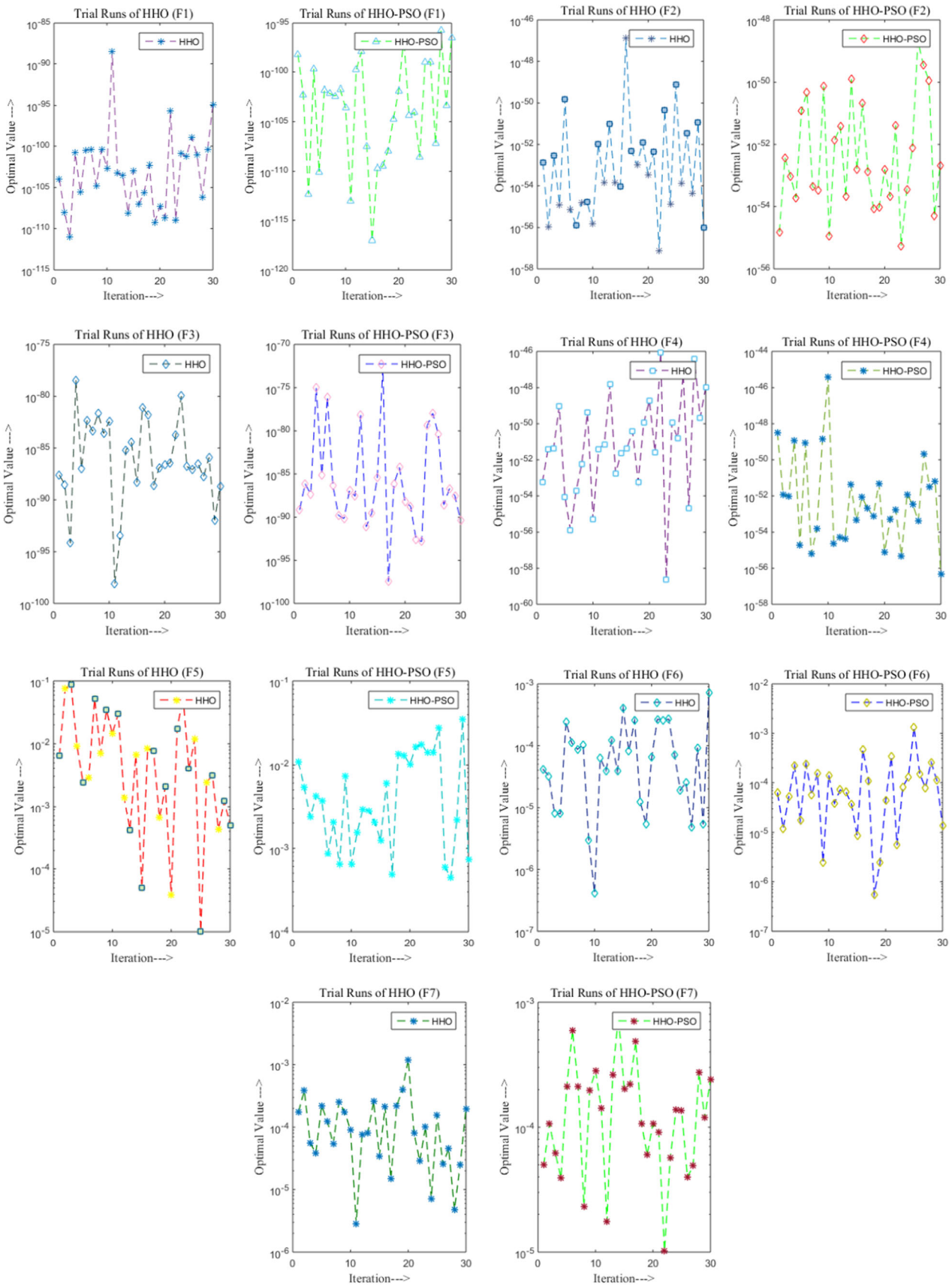


Fig. 9 Trial solutions for unimodal benchmark functions

Table 3 Test results for fixed dimension multi-modal benchmark functions using the hHHO-PSO algorithm

Functions	Mean	SD	Best	Worst	Median	p value
F14	1.359861	1.255145	0.998004	5.928845	0.998004	1.73E−06
F15	0.000394	0.000218	0.000311	0.001534	0.00035	1.73E−06
F16	− 1.03163	4.80E−09	− 1.03163	− 1.03163	− 1.03163	1.73E−06
F17	0.397904	3.49E−05	0.397887	0.398014	0.397887	1.72E−06
F18	3	5.28E−07	3	3.000002	3	1.73E−06
F19	− 3.86123	0.003411	− 3.86278	− 3.84668	− 3.8626	1.73E−06
F20	− 3.09508	0.103221	− 3.23625	− 2.83723	− 3.13262	1.73E−06
F21	− 5.0494	0.006604	− 5.05463	− 5.02164	− 5.0518	1.73E−06
F22	− 5.08332	0.004117	− 5.08739	− 5.07417	− 5.08524	1.73E−06
F23	− 5.12272	0.005766	− 5.12837	− 5.10713	− 5.12455	1.73E−06

6.3 8-bit multiplier

The multiplier block used in this case is the UCM architecture, explained in [61]. The additional circuitry added to the multiplier is synchronization. As used in the EA block, a PED-latch block pair is used at the output of every output bits of the multiplier.

6.4 16-bit register

Generally, due to fluctuation in the inputs, the output changes, and it is almost impossible to track the output. The primary use of the register is to hold the data until the next cycle is processed. Here, 16-bit registers are used at the final output and immediately after the multiplier. The main content of the register is a D flip-flop and a data selection circuit consisting of basic gates.

6.5 Exponent comparator circuit (ECC)

The ECC inputs are the product of the exponents (EA output, i.e. 5-bit) and the output exponent of the previous cycle (5 bits in size). The major point to consider here is that if both the input terms to the ECC block carry the same sign, then the actual difference among the two is the arithmetic difference between the numbers, whereas if both the inputs carry different signs, then the actual difference among the two is the arithmetic sum of the two numbers. For example, the actual difference between '+ a' and '+ b' is 'a − b' or 'b − a', whereas, for '− a' and '− b', the actual difference is '(− a) − (− b)', which is 'b − a' or '(− b) − (− a)' which is 'a − b'. But if the inputs are ' + a' and '− b' or '− a' and ' + b', then the actual difference is going to be 'a − (− b)', which is 'a + b' or 'b − (− a) which is 'b + a'. The flowchart explains the operation of the ECC block in Fig. 7. ECC architecture uses multiplexers to compare the inputs. This operation produces a 5-bit output, which is to be used for performing the binary shifts.

6.6 Exponent shifter circuit (ESC)

The ESC block is responsible for shifting the smaller number (either the product of the 8-bit inputs or the previous cycle MAC output) by the amount of difference between the exponents of these two. The inputs to the ESC block are the 5-bit output of the ECC block, a 16-bit product of the inputs, and 16-bit value of the previous cycle output. The step-by-step procedure is as follows:

1. The identification of the smaller number is made based on the ECC output (5-bits). If the MSB of the ECC block output is '1', then the product of the inputs is shifted towards the right by the equivalent decimal value of the remaining 4-bit binary of the ECC block output. On the other hand, if the MSB of the ECC block output is '0', then the previous output is shifted towards the right by the equivalent decimal value of the remaining 4-bit binary of the ECC block output.
2. The input to the ESC block, which need not be shifted, is identified by the MSB of the ECC block output.

6.7 16-bit adder

The adder block is again a synchronized block (i.e. it is clocked). The ESC block outputs processed through a 2's complement block and a 2:1 multiplexer for representing a positive or negative value. For example, if the shifted output of the ESC block is negative, then its 2's complement value is considered. Similarly, the non-shifted output of the ESC block is negative; then, it's 2s complement value is considered. The shifted or non-shifted number can be the product of the inputs or the previous output. Therefore, to distinguish the same, the 5th bit of the ECC

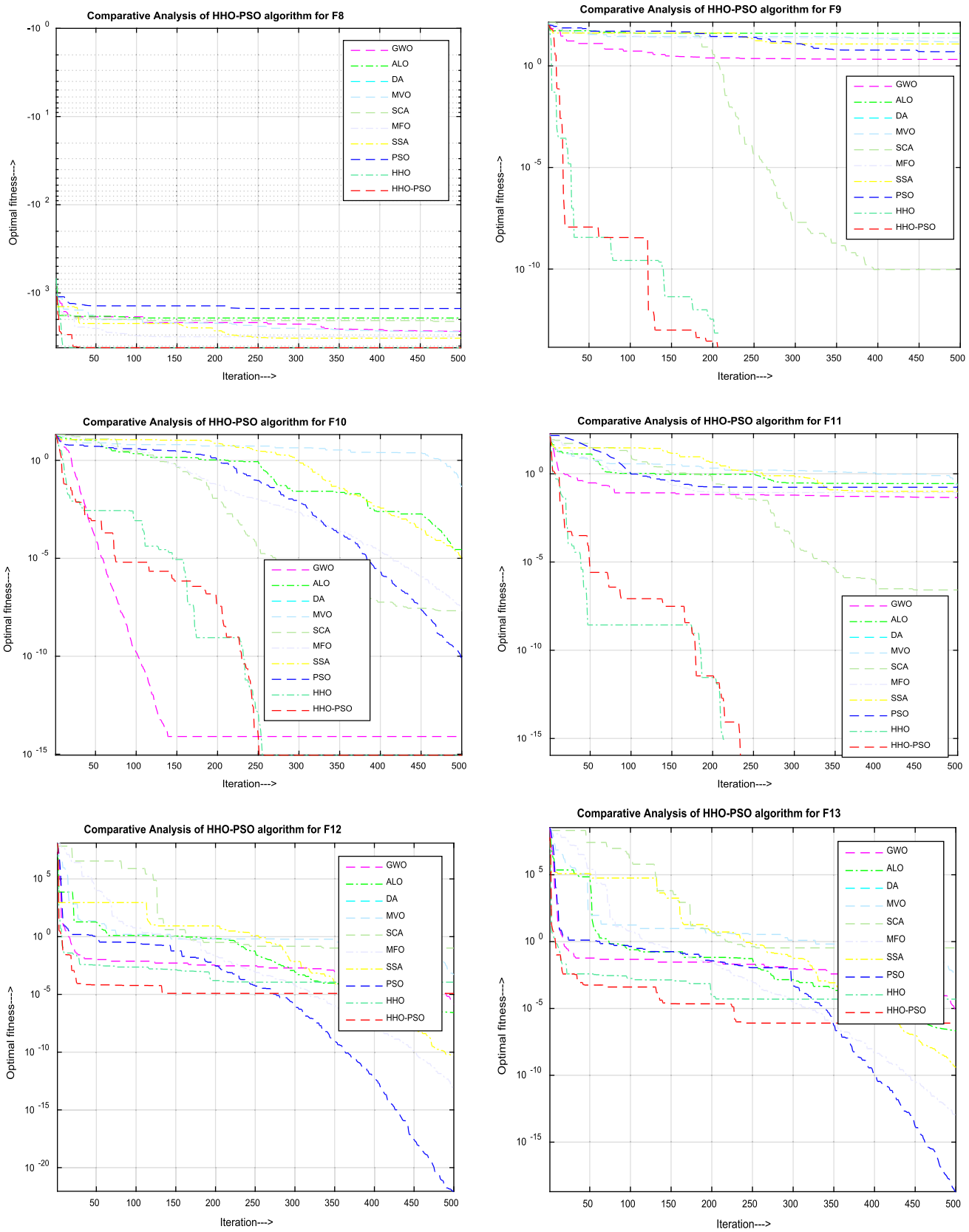


Fig. 10 Convergence curve of hHHO-PSO with GWO, ALO, DA, MVO, SCA, MFO, SSA, PSO and HHO for multi-modal benchmark functions

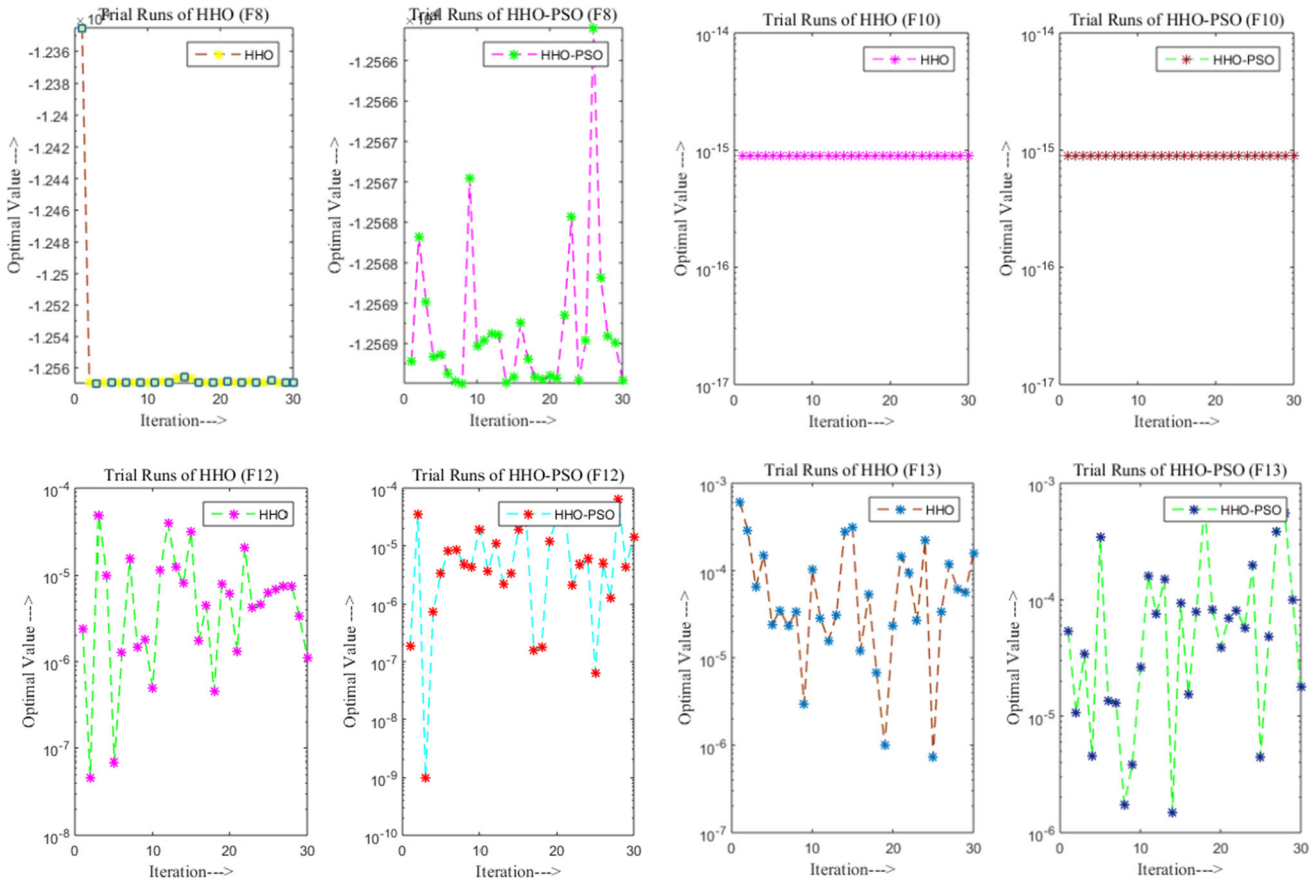


Fig. 11 Trial solutions for multi-modal benchmark functions

block output is considered. The shifted or non-shifted value of the current inputs' product and previous output in positive or negative number representation is the adder block's inputs. Additionally, the PED and latch pair is used in the adder block for synchronization.

6.8 1/4:1 multiplexer of different sizes

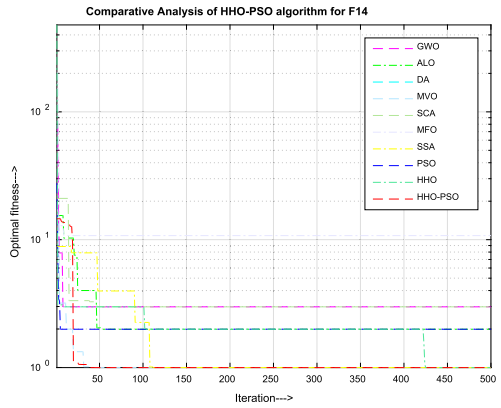
As the algorithm does not use any programming approach, multiplexers of different sizes with multiple or single bit are considered for solving the conditions.

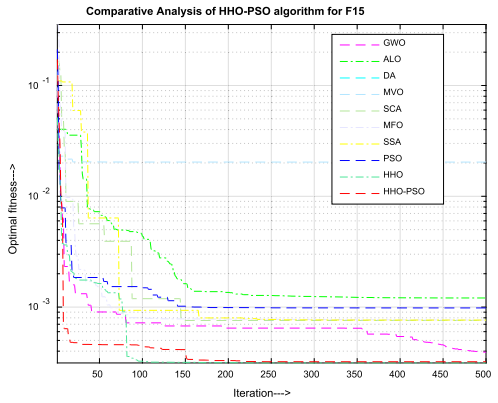
7 Results and discussion

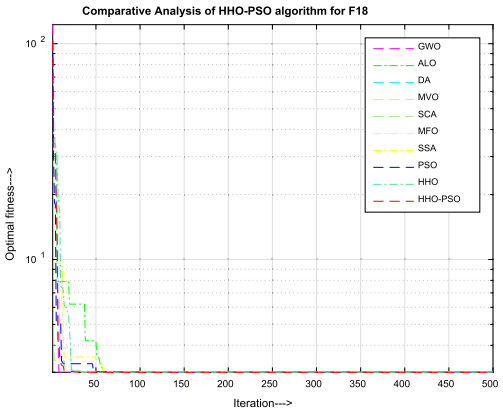
To validate the results, 30 trial runs are taken into consideration to overcome the stochastic nature of the projected hHHO-PSO algorithm, and each objective function has been estimated for average value, best values, standard deviation and worst value. To approve the phase of exploitation recommended algorithm, unimodal benchmark functions F1, F2, F3, F4, F5, F6, and F7 are taken into consideration. Table 2 shows the solution of the unimodal benchmark function using the hHHO-PSO algorithm. The convergence curve of hHHO-PSO and its comparative analysis with GWO, ALO, DA, MVO, SCA, MFO, SSA, PSO and HHO are shown in Fig. 8, and the trial solutions for unimodal benchmark functions are shown in Fig. 9. To validate the phase of exploration of the submitted algorithm, the multi-modal benchmark functions F7, F8, F9, F10 and F11 are taken into consideration, as these functions have many local optimal searches with the number increasing exponentially with dimension.

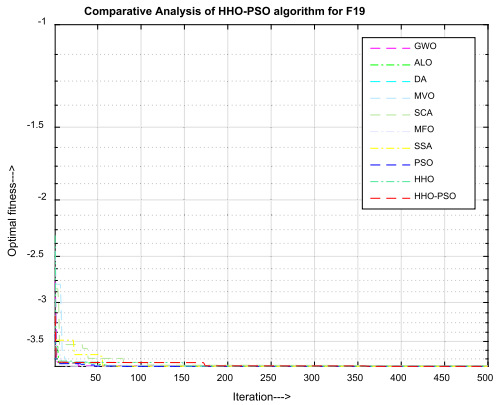
For fixed dimension benchmark problems, the test results are shown in Table 3. The trial solutions for fixed dimension benchmark functions and their convergence curve are shown in Fig. 14 and Figs. 7, 8, 9, 10, 11, 12, 13, respectively. For statistical analysis of the proposed algorithm, the nonparametric test, i.e. Wilcoxon rank-sum test, has been taken into consideration at 0.05 significance level.

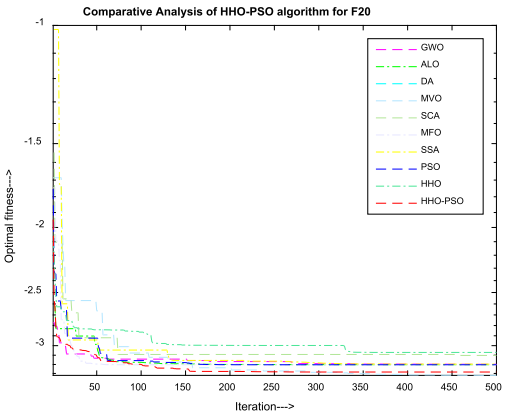
For statistical analysis of the proposed algorithm, the nonparametric test, i.e. Wilcoxon rank-sum test, has been taken into consideration at 0.05 significance level. The p value from Wilcoxon rank-sum has been recorded for CEC2017 Hybrid Benchmark functions and is shown in Tables 4, 5, 6, 7. It has been experimentally found that the

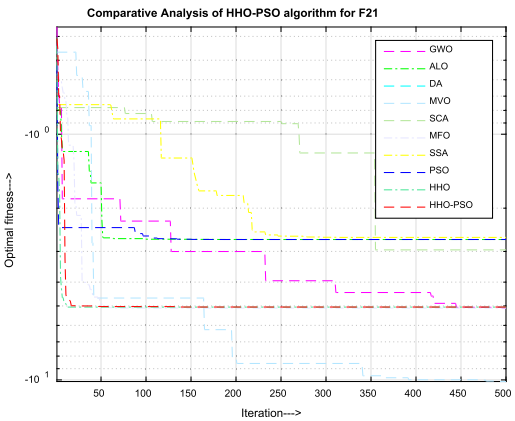

Comparative Analysis of HHO-PSO algorithm for F14
GWO
ALO
DA
MVO
SCA
MFO
SSA
PSO
HHO
HHO-PSO
Optimal fitness--->
Iteration--->
10 2
10 1
10 0
50 100 150 200 250 300 350 400 450 500

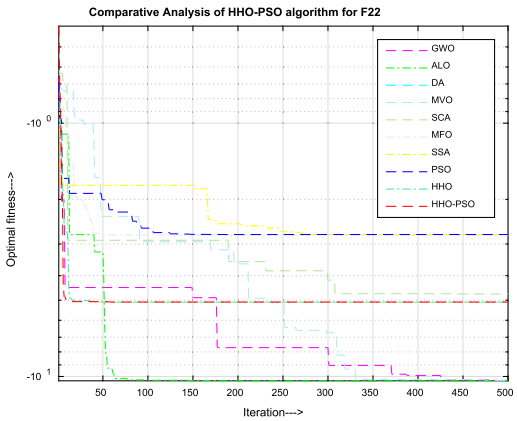

Comparative Analysis of HHO-PSO algorithm for F15
GWO
ALO
DA
MVO
SCA
MFO
SSA
PSO
HHO
HHO-PSO
Optimal fitness--->
Iteration--->
10 -1
10 -2
10 -3
50 100 150 200 250 300 350 400 450 500

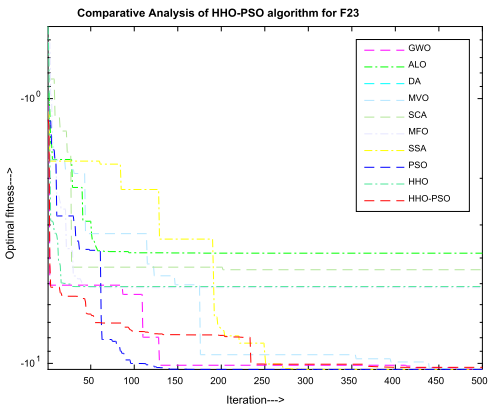

Comparative Analysis of HHO-PSO algorithm for F18
GWO
ALO
DA
MVO
SCA
MFO
SSA
PSO
HHO
HHO-PSO
Optimal fitness--->
Iteration--->
10 2
10 1
50 100 150 200 250 300 350 400 450 500


Comparative Analysis of HHO-PSO algorithm for F19
GWO
ALO
DA
MVO
SCA
MFO
SSA
PSO
HHO
HHO-PSO
Optimal fitness--->
Iteration--->
-1
-1.5
-2
-2.5
-3
-3.5
50 100 150 200 250 300 350 400 450 500


Comparative Analysis of HHO-PSO algorithm for F20
GWO
ALO
DA
MVO
SCA
MFO
SSA
PSO
HHO
HHO-PSO
Optimal fitness--->
Iteration--->
-1
-1.5
-2
-2.5
-3
50 100 150 200 250 300 350 400 450 500


Comparative Analysis of HHO-PSO algorithm for F21
GWO
ALO
DA
MVO
SCA
MFO
SSA
PSO
HHO
HHO-PSO
Optimal fitness--->
Iteration--->
-10 0
-10 1
50 100 150 200 250 300 350 400 450 500


Comparative Analysis of HHO-PSO algorithm for F22
GWO
ALO
DA
MVO
SCA
MFO
SSA
PSO
HHO
HHO-PSO
Optimal fitness--->
Iteration--->
-10 0
-10 1
50 100 150 200 250 300 350 400 450 500


Comparative Analysis of HHO-PSO algorithm for F23
GWO
ALO
DA
MVO
SCA
MFO
SSA
PSO
HHO
HHO-PSO
Optimal fitness--->
Iteration--->
-10 0
-10 1
50 100 150 200 250 300 350 400 450 500

◀**Fig. 12** Convergence curve of hHHO-PSO with GWO, ALO, DA, MVO, SCA, MFO, SSA, PSO and HHO for fixed dimension benchmark functions

low p-value of Wilcoxon rank-sum validates the research study at 0.05 significant level.

To verify the parametric variations of PSO, two different values of c1 and c2 are taken into consideration for CEC2017 benchmark functions, and comparative results are recorded for $c1 = 2$, $c2 = 2$, and $c1 = 0.95$, $c2 = 0.89$. It has been experimentally observed that the best value of $c1$ and $c2$ is 2 for proper tuning of the simulated annealing algorithm. The computational time is recorded for CEC2017 benchmark functions and compared with TLBO-FL and PSO-GWO algorithm, as shown in Table 7.

The test results for CEC2018 benchmarks functions are evaluated for 5, 10 and 15 number of objectives and are recorded in Table 8. The test results have been compared with NSGA-III$_{TS}$–NL [68] to show its effectiveness.

The design and implementation of the overall SFMAC architecture was done on Cadence Virtuoso CMOS technologies. The Cadence Spectre tool was used to perform a detailed analysis. The architecture uses a clock gating scheme along with pipelining to reduce the power dissipation.

7.1 Explanation of SFMAC using binary values

Let us consider an example to elaborate on the operation of the proposed architecture. The input numbers are as follows:

Input1 = **0**01001011, Input1 exponent = **0**000
Input2 = **1**01010001, Input2 exponent = **1**001

The MSB (9th bit) of input1 and input2 is the sign bit, highlighted in bold. Similarly, the MSB (4th bit) of input1-exponent and input2-exponent is the sign bit, which is highlighted in bold as well. In this example, input1 and input1-exponent are positive, and input2 and input2-exponent are negative. On the other hand, the previous output **0**0000000000000000 with exponent as **0**000. Therefore, the previous output, as well as its exponent, is positive. The execution steps as per the example mentioned above are as follows:

1. Based on the inputs, the product of the two inputs (NUM) is calculated as **1**0001011110111011 (in 16 bits). As one of the input numbers is negative, the resultant is negative.
2. The exponent of the NUM is the addition of the exponents of the inputs. The NUM exponent result is **1**0001 (− 1). The NUM exponent is represented in 5 bits because the addition of two 2-bit numbers can produce a result in 3-bit. Moreover, a negative 3-bit number requires 4 bits to represent. Additionally, the 5th bit is used to signify the sign bit.
3. If the exponents of the NUM and previous outputs are compared, then it can be observed that the exponent of NUM is − 1 and exponent of previous output is + 0. As the NUM is smaller than the last output, it is shifted by 1 bit from the left to right to get the updated value as **1**0000101111011101.
4. The shifted NUM (i.e. **1**0000101111011101) is added with previous output **0**0000000000000000 which produces a result as **1**0000101111011101 with exponent as **0**0000. The same is shown in HEX code as -0BDD $\times$ 2^{+0} in the output curve at the 2nd rising edge of clock 8, as shown in Fig. 14.
5. As the input does not change in the next cycle, the NUM remains the same, i.e. **1**0001011110111011. On the other hand, the latest values of input1-exponent and input2-exponent are **1**011 and **0**010. Therefore, it produces the NUM exponent as **1**0001 (1).
6. As the NUM in this cycle is smaller than the previous cycle output (as the last output's exponent is more significant than NUM exponent), the NUM is shifted by 1 bit towards its right, which produces the updated NUM as **1**0000101111011101 with updated NUM exponent as **0**0000.
7. The updated NUM and previous output are added and produce the result as **1**0001011110111010 with the exponent as **0**0000. The same is shown in HEX code as -17BA $\times$ 2^{+0} in the output curve at the 3rd rising edge of clock 8. The simulation waveform is shown in Fig. 14.

The pipeline mechanism using clock pulse is ensured by activating the consecutive blocks after a fixed period. But the clock for the consecutive blocks is differed by a delay of 1.4 ns. The delay is to latch the previous block's output effectively, which acts as the input for the next block. The delay of clock signals is calculated by the maximum propagation delay of the individual blocks of the SFMAC architecture, as expressed by Eq. (20).

$$\tau_{\text{Clock\_Delay}} = \max\left(\tau_{\text{delay\_EA}}, \tau_{\text{delay\_UCM}}, \tau_{\text{delay\_reg1}} \ldots \tau_{\text{delay\_reg2}}\right) \tag{20}$$

where $\tau_{\text{delay\_EA}}$ is the propagation delay of the EA block; $\tau_{\text{delay\_UCM}}$ is the propagation delay of the universal compressor-based multiplier (UCM); $\tau_{\text{delay\_reg1}}$ and $\tau_{\text{delay\_reg2}}$ are the propagation delay of the register 1 and register 2, respectively. For finding the delay of the internal blocks, the designs are implemented in 90 nm CMOS technology.

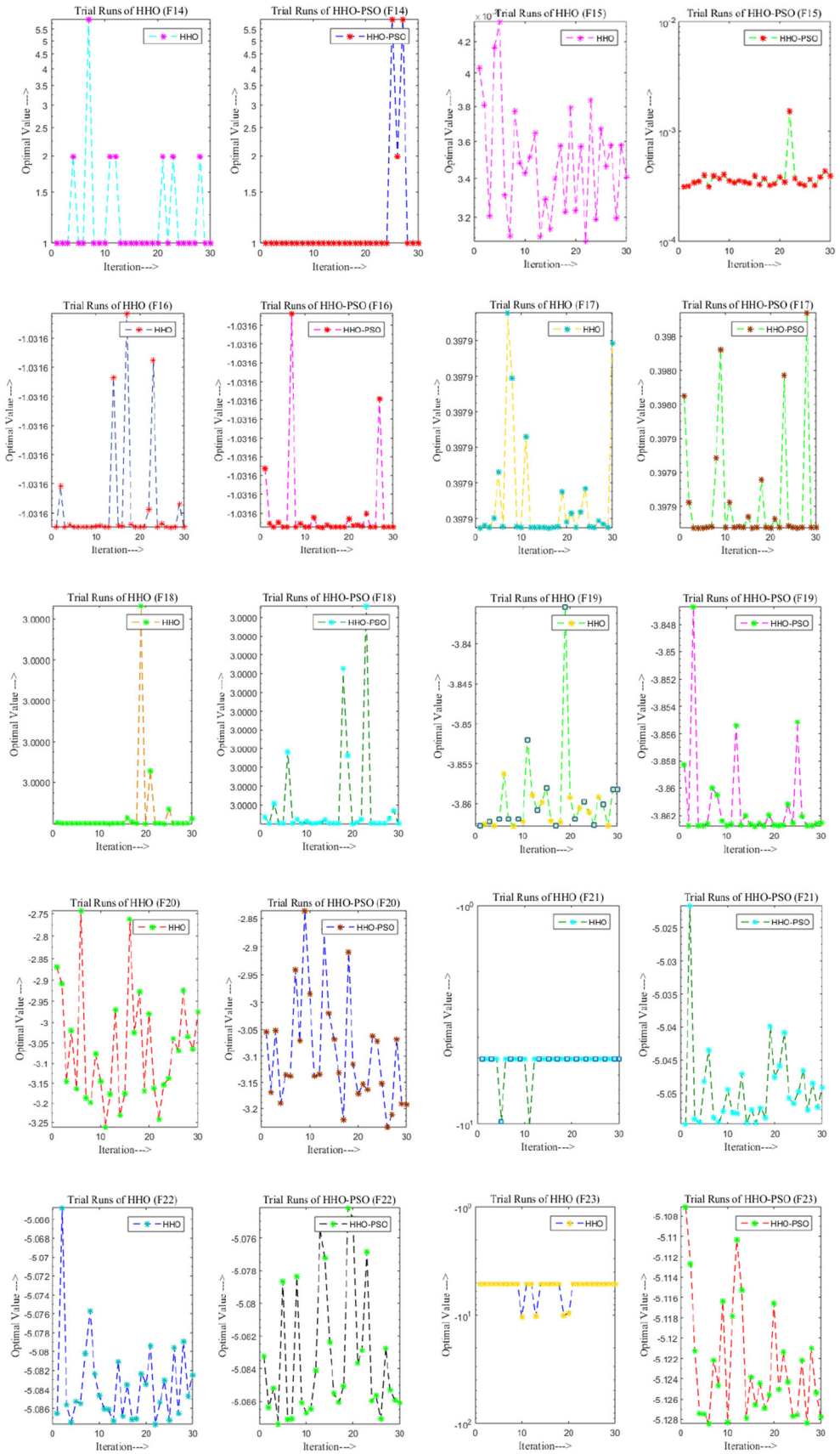


Fig. 13 Trial solutions for Fixed dimension benchmark functions

Table 4 Comparison of CEC 2017 multi-modal benchmark functions

CEC2017 benchmark functions	AHHO	jSO [63]	DES [64]	TLBO-FL [65]	PPSO [66]
CEC2017(F1)	0	0.00E+00	0	3.50E+03	7.48E+02
CEC2017(F2)	8.9401	8.94E+00	5.88E−02	8.50E+16	5.32E+01
CEC2017(F3)	0.000002389	2.3912E−06	0	3.00E+03	1.13E+00
CEC2017(F4)	189.6287	1.90E+02	5.69E+01	9.00E+01	4.39E+01
CEC2017(F5)	43.9065	4.39E+01	4.64E+00	4.00E+01	1.12E+02
CEC2017(F6)	0.000202421	2.0244E−04	3.34E−07	4.90E−01	2.03E+01
CEC2017(F7)	144.897	1.45E+02	3.57E+01	1.40E+02	1.35E+02
CEC2017(F8)	42.151653	4.22E+01	4.55E+00	3.70E+01	8.10E+01
CEC2017(F9)	0.045903546	4.5904E−02	0	3.40E+01	1.36E+03
CEC2017(F10)	9704.36758	9.70E+03	1.39E+02	6.70E+03	3.13E+03

Table 5 Comparison of CEC 2017 hybrid benchmark functions

CEC2017 hybrid benchmark functions	AHHO		jSO [63]	DES [64]	TLBO-FL [65]	PPSO [66]
	Mean value	*p* value				
CEC2017(F11)	104.1954	0.0234	1.04E+02	2.73E+01	8.20E+01	8.43E+01
CEC2017(F12)	17,033	0.1746	1.70E+04	1.21E+03	5.70E+04	2.77E+04
CEC2017(F13)	139.57953	0.3258	1.40E+02	4.87E+01	2.00E+04	3.21E+03
CEC2017(F14)	63.84568	0.477	6.39E+01	2.66E+01	7.10E+03	2.32E+03
CEC2017(F15)	164.67964	0.4282	1.65E+02	3.24E+01	2.20E+04	2.13E+03
CEC2017(F16)	1881.6784	0.3794	1.88E+03	7.64E+01	4.90E+02	8.46E+02
CEC2017(F17)	1273.1457	0.4306	1.27E+03	5.54E+01	1.40E+02	3.31E+02
CEC2017(F18)	156.92753	0.0818	1.57E+02	3.51E+01	3.70E+05	6.99E+04
CEC2017(F19)	106.34753	0.233	1.06E+02	1.64E+01	1.10E+04	1.71E+03
CEC2017(F20)	1380.9854	0.3842	1.38E+03	7.06E+01	2.20E+02	3.48E+02

Table 6 Comparison of CEC 2017 composite benchmark functions

CEC2017 composite benchmark functions	AHHO	jSO [63]	DES [64]	TLBO-FL [65]	PPSO [66]
CEC2017(F21)	263.5572	2.64E+02	2.07E+02	2.30E+02	3.05E+02
CEC2017(F22)	10,661.8733	1.07E+04	1.00E+02	1.00E+02	1.00E+02
CEC2017(F23)	571.18452	5.71E+02	3.50E+02	4.00E+02	6.81E+02
CEC2017(F24)	902.60963	9.03E+02	4.18E+02	4.70E+02	7.39E+02
CEC2017(F25)	761.62537	7.62E+02	3.87E+02	4.00E+02	3.85E+02
CEC2017(F26)	3281.0954	3.28E+03	5.74E+02	1.40E+03	2.04E+03
CEC2017(F27)	586.458356	5.86E+02	5.10E+02	5.30E+02	7.08E+02
CEC2017(F28)	523.664348	5.24E+02	3.18E+02	4.30E+02	3.27E+02
CEC2017(F29)	1237.89454	1.24E+03	4.44E+02	6.20E+02	7.80E+02
CEC2017(F30)	2317.0746	2.32E+03	2.16E+03	2.60E+04	3.32E+03

The evaluated delay values of the internal blocks of SFMAC are given in Table 9.

As there are a total of nine clocked blocks in this architecture, the amount of total delay required is 8 times 1.4 ns (1.4 ns × 8 = 11.2 ns), which means a set of inputs

Table 7 Comparison of CEC 2017 composite benchmark functions for parameters variations and computational time

CEC2017 composite benchmark functions	Effect of variations of parameters of PSO		Computational time (in s)		
	AHHO ($c1$ = 2, $c2$ = 2)	AHHO ($c1$ = 0.95, $c2$ = 0.89)	hHHO-PSO	TLBO-FL [65]	PSO-GWO [67]
CEC2017(F11)	263.5573	263.560	1.23E+02	1.42E+02	1.57E+02
CEC2017(F12)	10,661.8734	10662.000	1.12E+02	1.31E+02	1.45E+02
CEC2017(F13)	571.1845	571.190	1.24E+01	3.10E+01	4.54E+01
CEC2017(F14)	902.6096	902.610	1.36E+02	1.55E+02	1.69E+02
CEC2017(F15)	761.6254	761.630	1.69E+02	1.87E+02	2.02E+02
CEC2017(F16)	3281.0954	3281.100	1.88E+02	2.07E+02	2.21E+02
CEC2017(F17)	586.4574	586.460	1.95E+02	2.14E+02	2.28E+02
CEC2017(F18)	523.6642	523.670	1.85E+02	2.04E+02	2.18E+02
CEC2017(F19)	1237.8944	1237.900	1.75E+03	1.77E+03	1.78E+03
CEC2017(F20)	2317.0755	2317.100	1.74E+02	1.93E+02	2.07E+02

Table 8 Comparison of CEC 2018 benchmark function for 05, 10 and 15 objectives problems

CEC 2018 benchmark functions	No. of objectives = 05		No. of objectives = 10		No. of objectives = 15	
	AHHO	NSGA-III$_{TS}$-NL [68]	AHHO	NSGA-III$_{TS}$-NL [68]	AHHO	NSGA-III$_{TS}$-NL [68]
CEC2018 (F01)	9.73E−04	1.00E−03	0.0011078	1.11E−03	0.0014780	1.50E−03
CEC2018 (F02)	8.72E−04	8.72E−04	0.0018785	1.88E−03	0.0027376	2.74E−03
CEC2018 (F03)	1.63E−04	1.63E−04	0.0017698	1.77E−03	0.0020386	2.04E−03
CEC2018 (F04)	2.14E−04	2.14E−04	0.0001768	1.77E−04	0.0004609	4.61E−04
CEC2018 (F05)	1.56E−04	1.56E−04	0.0003370	3.37E−04	0.0006180	6.18E−04
CEC2018 (F06)	1.29E−04	1.21E−04	0.0026696	2.67E−03	0.0037696	3.77E−03
CEC2018 (F07)	1.42E−03	1.43E−03	0.0023589	2.36E−03	0.0023780	2.38E−03
CEC2018 (F08)	4.86E−04	4.86E−04	0.0006467	6.50E−04	0.0015287	1.53E−03
CEC2018 (F09)	2.21E−04	2.11E−04	*0.0014178*	*1.42E−03*	0.0013185	1.32E−03
CEC2018 (F10)	2.23E−03	2.24E−03	0.0020657	2.07E−03	0.0027754	2.78E−03
CEC2018 (F11)	3.61E−04	3.62E−04	0.0007388	7.40E−04	0.0013075	1.31E−03
CEC2018 (F12)	4.48E−04	4.47E−04	0.0008578	8.58E−04	*0.0020796*	*2.08E−03*
CEC2018 (F13)	4.17E−04	4.19E−04	0.0005488	5.49E−04	0.0012388	1.24E−03
CEC2018 (F14)	3.47E−02	3.49E−02	0.0980867	9.81E−02	0.0680864	6.81E−02
CEC2018 (F15)	1.64E−03	1.67E−03	0.0024560	2.50E−03	0.0080765	8.10E−03

latched at time 0 ns is evaluated and produces the output only after 11.2 ns. Therefore, the clock period is fixed at 12 ns (or 83.333 MHz operational frequency), so the execution of the last clock and latching on the first clock does not get overlapped. Figure 6 shows the output waveform of the proposed SFMAC architecture. The clock in the SFMAC architecture is applied in a pipelined manner, as shown in Fig. 14. Figure 14 also indicates input/output pulses of the proposed SFMAC architecture. The latching of a new set of exponential input is done in clock 0, by activating the EA block which provides the NUM (exponential) output.

The *Write*/$\overline{Read}$ signal is used for selecting the write or read operation, which is an active high signal, whereas the $\overline{Reset}$ signal is used for force resetting the SFMAC system, and it is an active low signal. In clock 1, the UCM is enabled, providing the 16-bit NUM output based on two 8-bit inputs IN1 and IN2. Clock 2 signal is used as a clock signal for the 16-bit register for the multiplier, and there is no clock applied to the ECC block, which produces 5-bit RES output. Clock 3 and RES are applied to the ESC

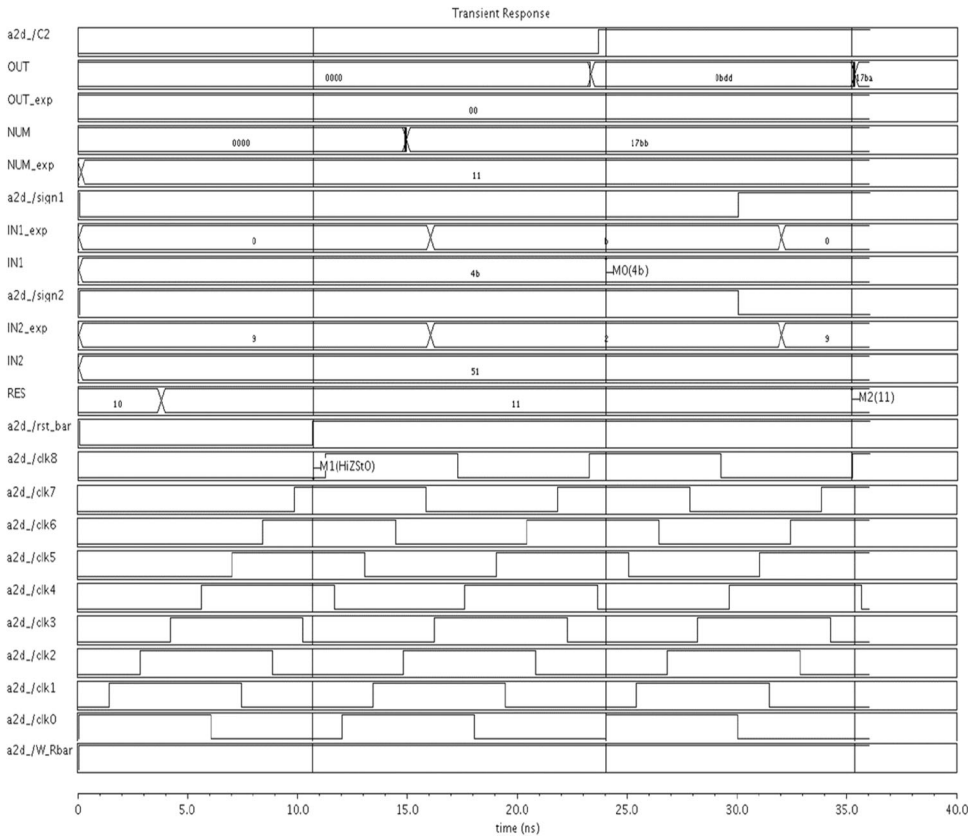


Fig. 14 The simulation waveform of the SFMAC architecture

Table 9 Propagation delay of the internal blocks of SFMAC architecture

Block	Delay (ps)	Inference
UCM	433.7	The maximum delay from A_0 to P_{15}, considering all inputs as high
Register	123.6	Delay from the positive edge of the clock to any bit of the output
Full adder	22.4	Delay from A_0 to OUT_{15}, considering all inputs as high
ESC block (along with ECC block)	1367.5	With same sign bits of both the exponents (as negative or positive) in the EA and ECC block and maximum bit shift in the ESC block
2:1 MUX	12.6	With the critical path from 's' to 'y'
4:1 MUX	18.9	The maximum delay occurred either in 's_0' to 'y', 's_1' to 'y' or 's_2' to 'y'

Table 10 The operation of the 16-bit 4:1 MUX based on the two select lines

XOR of the sign bit of the inputs	The sign bit of the previous output	Operation
0	0	No change or true form
0	1	Pass the output of the 16-bit 2:1 MUX as such
1	0	Pass the output of the 16-bit 2:1 MUX as such
1	1	2's complement

block, which yields the shifted/non-shifted NUM or previous output. Parallelly, the same clock is applied to the 2:1 MUXs for updating the select line of the 16-bit 2:1 MUXs to update the true or complemented NUM or previous output. Clock 4 is applied to the 16-bit 2:1 MUXs to update the true or complemented NUM/previous output. On the

Table 11 Comparison of SFMAC at supply voltage 2 V and simulation period 40 ns in CMOS GPDK 90 nm and TSMC 130 nm technologies

Architecture	Static power in μW (for V_{DD} = 2 V)	Average power in μW (for V_{DD} = 2 V and simulation period = 40 ns)	Area (total number of transistors)
SFMAC (TSMC 130 nm)	2398.76	25,990	25,783
SFMAC (GPDK 90 nm)	476.94	7980	25,783

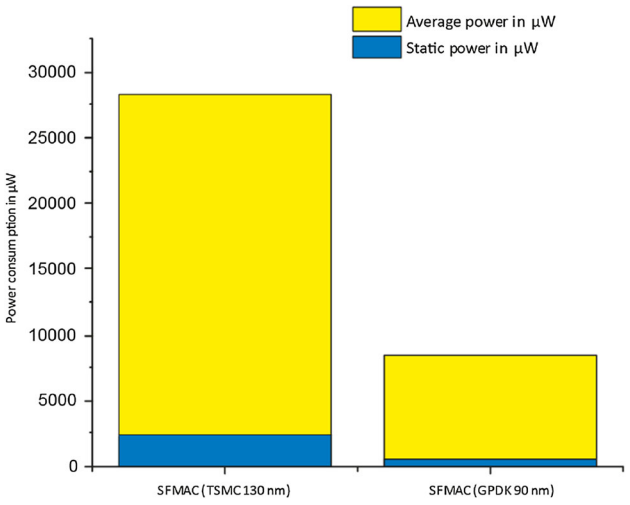


Fig. 15 Graphical comparison of SFMAC in TSMC 130 nm and GPDK 90 nm

other hand, for adding the true or 2's complement form of shifted/non-shifted 16-bit inputs, clock 5 is applied. The 16-bit 2:1 MUX block is activated on the edges of clock 6, which choose between the true/2's complement value of the full adder output. The carry bit of the output of the full adder is applied as the select line. The overall output of the MAC block is based on the selection of XOR of the sign bit of the inputs and the sign bit of the previous output. The input for the 16-bit 4:1 MUX is the output of the 16-bit 2:1 MUX. The inputs for the 16-bit 4:1 MUX are latched at the positive edges of clock 7. The operation of the 16-bit 4:1 MUX is explained in Tables 9, 10. Finally, a 16-bit register is used at the output so that the internal glitches do not change the output value. The 16-bit register block is enabled with clock 8, which yields the OUT (SFMAC output), OUT (output exponent), and C2 (sign bit of the output).

Table 11 shows a power comparison of SFMAC architecture at different CMOS technologies in a specific input vector. The simulation period is kept as 40 ns because:

a. The reset signal (active low) is low till 10.8 ns and
b. The clock signals have a period of 12 ns

Therefore, until 23.2 ns, the output signal remains at '0'. The SFMAC architecture is not only implemented in GPDK 90 nm but also TSMC 130 nm CMOS technology. The power consumption of the implemented designs is calculated using Cadence Spectre Tool. Figure 15 shows the graphical analysis of the static power, average power and area of SFMAC in CMOS GPDK 90 nm and TSMC 130 nm technology. The static power is evaluated for 2 V supply voltage, whereas the average power is measured for a simulation period of 40 ns and at a frequency of 83.33 MHz. The average dynamic power consumption of the SFMAC in TSMC 130 nm is higher than GPDK 90 nm as the transistor sizing is higher in 130 nm technology, which affects the load capacitance C_{load}. Similarly, the static power consumption is also a function of device geometry. Therefore, a circuit consisting of a higher device dimension has higher static power consumption. The average dynamic power of a CMOS circuit is given by Eq. (21).

$$P_{avg} = \alpha_T C_{load} V_{DD}^2 f_{clk} \tag{21}$$

7.2 Comparison with existing architectures

The comparison in terms of power consumption of the proposed MAC architectures with the existing MAC architecture is shown in Table 12. It is challenging to compare the proposed SFMAC architectures with those already available in the literature because most of the possible architectures in the literature have used the HDL-based approach. On the other hand, the proposed architectures are implemented in Cadence Virtuoso 90 nm environment. Moreover, almost 99% of the architectures available in the literature have neither implemented for signed operation nor floating-point designs. Though some architectures in the literature have used the clocking signals for the accumulation of data only (in the register or accumulator), most of the architectures have not used any clocking signal. Any circuit in asynchronous mode cannot be implemented in a real-time application. Therefore, the practical applicability of such architecture needs to be further tested. Most of the architectures explained in Table 12 are implemented for unsigned fixed-point MAC operation, and only one architecture (i.e. Zhang et al. [69]) was implemented for floating-point signed operation.

Table 12 Performance comparison of proposed MAC architecture with existing architectures

Sl no.	Author	Existing architecture description	Implementation on	Power consumption	
1.	Shanthala et al. 2009 [70]	Pipelined Multiply Accumulate Unit (without synchronization) in 180 nm technology, 1.8 V at 83.3 MHz and 8 × 8 bit operation	Cadence Virtuoso	50.26 mW	
2.	Jagadees et al. 2013 [15]	Multiply Accumulate Unit (without synchronization) in 180 nm technology, 1.8 V at 217 MHz and 64 × 64 bit operation	Verilog HDL	177.732 mW	
3.	Hoang et al. 2010 [71]	Pipelined Multiply Accumulate Unit (without synchronization) in 65 nm technology, 1.1 V at 591 MHz and 16 × 16 bit operation	VHDL	8.2 mW	
4.	Esmaeili et al. 2012 [72]	Multiply Accumulate Unit (without synchronization) in 90 nm technology, 1 V at 100 MHz and 16 × 16 bit operation	HDL in Cadence's HSPICE simulator	1.506 mW	
5.	Akbarzadeh et al. 2015 [74]	Pipelined Multiply Accumulate Unit (without synchronization) in 180 nm technology, 1.8 V and 8 × 8 bit operation	HDL in Synopsys Design Compiler	Dynamic Power 3.627 mW	Static Power 2.010 mW
6.	Rahul Narasimhan et al. 2015 [73]	Multiply Accumulate Unit (without synchronization) in 180 nm technology, 1.8 V at 5 MHz and 16 × 16 bit operation	Verilog HDL	MAC using Booth 493.648 mW	MAC using Vedic 1765.241 mW
7.	Karthikeyan et al. 2016 [75]	Multiply Accumulate Unit (without synchronization) in 32 nm CMOS and CNTFET technology and 1 × 1 bit operation	–	CMOS Tech 0.9902 mW	CNTFET Tech 0.6335 mW
8.	Zhang et al. 2018 [69]	Fixed/Floating-Point Multiply Accumulate Unit (without synchronization) in 90 nm technology for 16-bit half-precision multiplication	VHDL	14.07 mW	
9.	Proposed SFMAC using AHHO	Signed floating-point MAC architecture in 90 nm tech., 2 V at 83.33 MHz and 8 × 8 bit operation	Cadence Virtuoso 90 nm CMOS	Static Power 0.476 mW	Average Power 7.98 mW

The differences are visible from Table 12 that the performance of [15, 70, 71] has a significantly higher static as well as average power (in mW) than proposed SFMAC architecture. The performance of [72, 73] is evaluated in 90 nm and 180 nm technologies for 16-bit operations at 1 V and 8-bit operations 1.8 V, respectively. The power consumptions of the existing work mentioned in [72, 74] are lesser than proposed SFMAC architecture (the existing circuit's performance analysis is performed in supply voltage lower than 2 V, whereas for the SFMAC the supply voltage is 2 V), but these two current architectures are capable of performing the MAC operation on fixed-point unsigned number only. Therefore, the existing MAC in serial number [72, 73] has limited scope. Though the architecture is mentioned in [73] implemented in 180 nm technology and 1.8 V supply voltage for 16-MAC operation, the power consumption is way more than the SFMAC architecture. For the architecture mentioned in [75], the implementation is done for 1-bit unsigned fixed-point MAC operation in 32 nm CMOS and CNTFET technology, and hence, comparison with 8-bit SFMAC is not relevant. Though, the architecture mentioned in [69] is the only existing MAC architecture capable of performing on signed floating-point input, the comparison analysis with proposed SFMAC shows that the performance of SFMAC is much better in terms of power consumption.

8 Conclusion

In the proposed research, the phase of exploitation of the existing Harris Hawks optimizer has been upgraded successfully using a particle swarm optimization algorithm. The proposed hybrid HHO-PSO algorithm has been successfully tested for highly constrained, nonlinear and non-convex optimization problems. The objective of this research paper was to design suitable low-power high-speed MAC units for signed floating operation. To achieve the said objective, different MAC architectures, which were already existing in the literature, were studied. It was observed that signed floating-point MAC architecture using basic 2:1/4:1 multiplexer was never proposed in the literature. As the multiplier is the core component of any MAC unit, a high-speed UCM is used to design and implement the novel SFMAC architecture. The step-by-step elaboration of the architecture design is explained in this paper. For design and implementation, the Cadence Spectre tool is used at CMOS 90 nm as well as TSMC 130 nm technologies. The results have proved that the proposed SFMAC architecture consumes less power and a tolerable amount of worst-case delay. Therefore, it has applicability in low-power high-speed DSP architectures.

Acknowledgements The authors are very thankful to Dr Ali Asghar Heidari and Dr Seyedali Mirjalili for providing free access to the MATLAB code of the HHO algorithm. Dr. Vikram Kumar Kamboj wish to thank Dr. O.P. Malik, Professor Emeritus, Department of Electrical and Computer Engineering, Schulich School of Engineering, University of Calgary, Calgary, Alberta, Canada, for their guidance, continuous support and encouragement and for providing advance research facilities for postdoctorate research at University of Calgary, Alberta, Canada.

Compliance with ethical standards

Conflict of interest The authors certify that they have no affiliations with or involvement in any organization or entity with any financial interest (such as honoraria; educational grants; participation in speakers' bureaus; membership, employment, consultancies, stock ownership, or other equity interest; and expert testimony or patent-licensing arrangements), or non-financial interest (such as personal or professional relationships, affiliations, knowledge or beliefs) in the subject matter or materials discussed in this manuscript.

Appendix 1: Test data for unimodal benchmark functions

Benchmark functions	Dimensions	Range	$f_{\min}$
$F_1(\eta) = \sum_{m=1}^{z} \eta_m^2$	30	[− 100, 100]	0
$F_2(\eta) = \sum_{m=1}^{z} \lvert\eta_m\rvert + \prod_{m=1}^{2} \lvert\eta_m\rvert$	30	[− 10, 10]	0
$F_3(\eta) = \sum_{m=1}^{z} (\sum_{n-1}^{m} \eta_n)^2$	30	[− 100, 100]	0
$F_4(\eta) = \max_m\{\lvert\eta_m\rvert, 1 \le m \le z\}$	30	[− 100, 100]	0
$F_5(\eta) = \sum_{m=1}^{z-1} [100(\eta_{m+1} - \eta_m^2)^2 + (\eta_m - 1)^2]$	30	[− 38, 38]	0
$F_6(\eta) = \sum_{m=1}^{z} ([\eta_m + 0.5])^2$	30	[− 100, 100]	0
$F_7(\eta) = \sum_{m=1}^{z} m\eta_m^4 + \text{random}[0, 1]$	30	[− 1.28, 1.28]	0

Appendix 2: Test data for multi-modal benchmark functions

Benchmark functions	Dim	Range	$f_{\min}$
$F_8(\eta) = \sum_{m=1}^{z} -\eta_m \sin(\sqrt{\lvert\eta_m\rvert})$	30	[− 500, 500]	− 418.98295
$F_9(\eta) = \sum_{m=1}^{z} [\eta_m^2 - 10\cos(2\pi\eta_m) + 10]$	30	[− 5.12, 5.12]	0
$F_{10}(\eta) = -20\exp\left(-0.2\sqrt{\left(\frac{1}{z}\sum_{m=1}^{z}\eta_m^2\right)}\right) - \exp(\frac{1}{z}\sum_{m=1}^{z}\cos(2\pi\eta_m)) + 20 + d$	30	[− 32,32]	0
$F_{11}(\eta) = 1 + \sum_{m=1}^{z} \frac{\eta_m^2}{4000} - \pi_{m=1}^{z} \cos\frac{\eta_m}{\sqrt{i}}$	30	[− 600, 600]	0
$F_{12}(\eta) = \frac{\pi}{z}\{10\sin(\pi\tau_1) + \sum_{m=1}^{z-1}(\tau_m - 1)^2 [1 + 10\sin^2(\pi\tau_m + 1)] + (\tau_z - 1)^2 + \sum_{m=1}^{z} u(\eta_m, 10, 100, 4)$	30	[− 50,50]	0
$F_{13}(\eta) = 0.1\{\sin^2(3\pi\eta_1) + \sum_{m=1}^{z}(\eta_m - 1)^2 [1 + \sin^2(3\pi\eta_m + 1)] + (\eta_z - 1)^2 [1 + \sin^2(2\pi\eta_z)]\}$	30	[− 50,50]	0

Appendix 3: Test data for fixed dimensions benchmark functions

Mixed-modal benchmarked functions	Dim	Range	$f_{\min}$
$F_{14}(\eta) = [\frac{1}{500} + \sum_{m=1}^{z} 5 \frac{1}{n+\sum_{m=1}^{z}(\eta_m - b_{mn})6}]^{-1}$	2	[− 65.536, 65.536]	1
$F_{15}(\eta) = \sum_{m=1}^{z} [b_m - \frac{\eta_1(a_m^2 + a_m\eta_2)}{a_m^2 + a_m\eta_3 + \eta_4}]^2$	4	[− 5, 5]	0.00030
$F_{16}(\eta) = 4\eta_1^2 - 2.1\eta_1^4 + \frac{1}{3}\eta_1^6 + \eta_1\eta_2 - 4\eta_2^2 + 4\eta_2^4$	2	[− 5, 5]	− 1.0316
$F_{17}(\eta) = \left(\eta_2 - \frac{5.1}{4\pi 2}\eta_1^2 + \frac{5}{\pi}\eta_1 - 6\right)^2 + 10\left(1 - \frac{1}{8\pi}\right)\cos\eta_1 + 10$	2	[− 5, 5]	0.398
$F_{18}(\eta) = \left[1 + (\eta_1 + \eta_2 + 1)^2\left(19 - 14\eta_1 + 3\eta_1^2 - 14\eta_2 + 6\eta_1\eta_2 + 3\eta_2^2\right)\right] \times \left[30 + (2\eta_1 - 3\eta_2)^2 \times \left(18 - 32\eta_1 + 12\eta_1^2 + 48\eta_2 - 36\eta_1\eta_2 + 27\eta_2^2\right)\right]$	2	[− 2, 2]	3
$F_{19}(\eta) = -\sum_{m=1}^{4} d_m \exp\left(-\sum_{n=1}^{3} \eta_{mn}(\eta_m - q_{mn})^2\right)$	3	[1, 3]	− 3.32
$F_{20}(\eta) = -\sum_{m=1}^{4} d_m \exp\left(-\sum_{n=1}^{6} \eta_{mn}(\eta_m - q_{mn})^2\right)$	6	[0, 1]	− 3.32
$F_{21}(\eta) = -\sum_{m=1}^{5} [(\eta - b_m)(\eta - b_m)^T + d_m]^{-1}$	4	[0,10]	− 10.1532
$F_{22}(\eta) = -\sum_{m=1}^{7} [(\eta - b_m)(\eta - b_m)^T + d_m]^{-1}$	4	[0, 10]	− 10.4028
$F_{23}(\eta) = -\sum_{m=1}^{7} [(\eta - b_m)(\eta - b_m)^T + d_m]^{-1}$	4	[0, 10]	− 10.5363

References

1. Abbassi R, Abbassi A, Asghar A, Mirjalili S (2019) An efficient salp swarm-inspired algorithm for parameters identification of photovoltaic cell models. Energy Convers Manag 179:362–372
2. Faris H et al (2019) An intelligent system for spam detection and identification of the most relevant features based on evolutionary Random Weight Networks. Inf Fusion 48:67–83
3. McCarthy JF (1989) Block-conjugate-gradient method. Phys Rev D 40(6):2149–2152
4. Wu G (2016) Across neighborhood search for numerical optimization. Inf Sci (NY) 329(61563016):597–618
5. Wu G, Pedrycz W, Suganthan PN, Mallipeddi R (2015) A variable reduction strategy for evolutionary algorithms handling equality constraints. Appl Soft Comput J 37:774–786
6. Heidari AA, AliAbbaspour R, RezaeeJordehi A (2017) An efficient chaotic water cycle algorithm for optimization tasks. Neural Comput Appl 28(1):57–85
7. Mafarja M et al (2018) Evolutionary population dynamics and grasshopper optimization approaches for feature selection problems. Knowl Based Syst 145:25–45
8. Wolpert DH, Macready WG (1997) No free lunch theorems for optimization. IEEE Trans Evol Comput 1(1):67–82
9. Abdelgawad A, Bayoumi M (2007) High speed and area-efficient Multiply Accumulate (MAC) unit for digital signal prossing applications. In: 2007 IEEE international symposium on circuits and systems, pp 3199–3202
10. Babu NJ, Sarma R (2016) A novel low power multiply-accumulate (MAC) unit design for fixed point signed numbers. In: Artificial intelligence and evolutionary computations in engineering systems. Springer, pp 675–690
11. Rao MJ, Dubey S (2012) A high speed and area efficient Booth recoded Wallace tree multiplier for Fast Arithmetic Circuits. In: 2012 Asia Pacific conference on postgraduate research in microelectronics and electronics, pp 220–223
12. Luu X-V, Hoang T-T, Bui T-T, Dinh-Duc A-V (2014) A high-speed unsigned 32-bit multiplier based on booth-encoder and wallace-tree modifications. In: 2014 international conference on advanced technologies for communications (ATC 2014), pp 739–744
13. Xia B, Liu P, Yao Q (2009) New method for high performance multiply-accumulator design. J Zhejiang Univ A 10(7):1067–1074
14. Deepak S, Kailath BJ (2012) Optimized MAC unit design. In: 2012 IEEE international conference on electron devices and solid state circuit (EDSSC), pp 1–4
15. Jagadeesh P, Ravi S, Mallikarjun KH (2013) Design of high performance 64 bit MAC unit. In: 2013 international conference on circuits, power and computing technologies (ICCPCT), pp 782–786
16. Francis T, Joseph T, Antony JK (2013) Modified MAC unit for low power high speed DSP application using multipler with bypassing technique and optimized adders. In: 2013 fourth international conference on computing, communications and networking technologies (ICCCNT), pp 1–4
17. Warrier R, Vun CH, Zhang W (W) A low-power pipelined MAC architecture using Baugh-Wooley based multiplier. In: 2014 IEEE 3rd global conference on consumer electronics (GCCE), pp 505–506

18. Xu Z et al (2020) Orthogonally-designed adapted grasshopper optimization: a comprehensive analysis. Expert Syst Appl 150:113282
19. Banerjee N, Mukhopadhyay S (2019) HC-PSOGWO: hybrid crossover oriented PSO and GWO based co-evolution for global optimization. In: 2019 IEEE region 10 symposium (TENSYMP), pp 162–167
20. Shahrouzi M, Salehi A (2020) Imperialist competitive learner-based optimization: a hybrid method to solve engineering problems. Int J Optim Civ Eng 10(1):155–180
21. Sulaiman MH, Mustaffa Z, Saari MM, Daniyal H, Musirin I, Daud MR (2018) Barnacles mating optimizer: an evolutionary algorithm for solving optimization. In: 2018 IEEE international conference on automatic control and intelligent systems (I2CACIS), pp 99–104
22. Faramarzi A, Heidarinejad M, Stephens B, Mirjalili S (2020) Equilibrium optimizer: a novel optimization algorithm. Knowl Based Syst 191:105190
23. Muhammed DA, Saeed SAM, Rashid TA (2020) Improved fitness-dependent optimizer algorithm. IEEE Access 8:19074–19088
24. MostafaBozorgi S, Yazdani S (2019) IWOA: An improved whale optimization algorithm for optimization problems. J Comput Des Eng 6(3):243–259
25. Chen H, Wang M, Zhao X (2020) A multi-strategy enhanced sine cosine algorithm for global optimization and constrained practical engineering problems. Appl Math Comput 369:124872
26. Yimit A, Iigura K, Hagihara Y (2020) Refined selfish herd optimizer for global optimization problems. Expert Syst Appl 139:112838
27. Zhao W, Wang L, Zhang Z (2019) Artificial ecosystem-based optimization: a novel nature-inspired meta-heuristic algorithm. Neural Comput Appl 32:9383–9425
28. Seyyedabbasi A, Kiani F (2019) I-GWO and Ex-GWO: improved algorithms of the Grey Wolf Optimizer to solve global optimization problems. Eng Comput. https://doi.org/10.1007/s00366-019-00837-7
29. Khatri A, Gaba A, Rana KPS, Kumar V (2019) A novel life choice-based optimizer. Soft Comput. 24:9121–9141
30. Tejani GG, Kumar S, Gandomi AH (2019) Multi-objective heat transfer search algorithm for truss optimization. Eng Comput. https://doi.org/10.1007/s00366-019-00846-6
31. Xiao B, Wang R, Xu Y, Wang J, Song W, Deng Y (2019) Simplified Salp swarm algorithm. In: 2019 IEEE international conference on artificial intelligence and computer applications (ICAICA), pp 226–230
32. Chen X, Tianfield H, Li K (2019) Self-adaptive differential artificial bee colony algorithm for global optimization problems. Swarm Evol Comput 45:70–91
33. Kamboj VK, Nandi A, Bhadoria A, Sehgal S (2019) An intensify Harris Hawks optimizer for numerical and engineering optimization problems. Appl Soft Comput. https://doi.org/10.1016/j.asoc.2019.106018
34. Dhawale D, Kamboj VK (2020) hHHO-IGWO: a new hybrid Harris Hawks optimizer for solving global optimization problems. In: 2020 international conference on computation, automation and knowledge management (ICCAKM), pp 52–57
35. Bui DT et al (2019) A novel swarm intelligence—Harris hawks optimization for spatial assessment of landslide susceptibility. Sensors (Basel) 19(16):3590
36. Fan Q, Chen Z, Xia Z (2020) A novel quasi-reflected Harris hawks optimization algorithm for global optimization problems. Soft Comput 24:14825–14843
37. Jia H, Lang C, Oliva D, Song W, Peng X (2019) Dynamic Harris Hawks Optimization with mutation mechanism for satellite image segmentation. Remote Sens. 11(12):1421
38. Too J, Abdullah AR, MohdSaad N (2019) A new quadratic binary Harris Hawk optimization for feature selection. Electronics 8(10):1130
39. Heidari AA, Mirjalili S, Faris H, Aljarah I, Mafarja M, Chen H (2019) Harris hawks optimization: algorithm and applications. Future Gener Comput Syst 97:849–872
40. Kamboj VK, Nandi A, Bhadoria A, Sehgal S (2020) An intensify Harris Hawks optimizer for numerical and engineering optimization problems. Appl Soft Comput J 89:106018
41. Du P, Wang J, Hao Y, Niu T, Yang W (2020) "A novel hybrid model based on multi-objective Harris hawks optimization algorithm for daily PM 2.5 and PM 10 forecasting. Appl Soft Comput 96:106620
42. Abbasi A, Firouzi B, Sendur P (2019) On the application of Harris hawks optimization (HHO) algorithm to the design of microchannel heat sinks. Eng Comput. https://doi.org/10.1007/s00366-019-00892-0
43. Abd M, Asghar A, Fujita H, Moayedi H (2020) A competitive chain-based Harris Hawks Optimizer for global optimization and multi-level image thresholding problems. Appl Soft Comput J 95:106347
44. Chen H, Jiao S, Wang M, Heidari AA, Zhao X (2019) Parameters identification of photovoltaic cells and modules using diversification-enriched Harris Hawks optimization with chaotic drifts. J Clean Prod 244:118778
45. Yousri D, Rezk H (2020) "Identifying the parameters of different configurations of photovoltaic models based on recent artificial ecosystem-based optimization approach. Int J Energy Red 44:11302–11322
46. Article R (2020) Problem solving in cross over line balancing using HHO. J Crit Rev 7(4):275–281
47. Arora K, Kumar A, Kumar V (2019) Sensitivity analysis of load frequency control problem considering impact of wind penetration using improved Harris Hawks optimizer. Int J Innov Technol Explor Eng 9(2):195–205
48. Hussain K, Zhu W, Mohd Salleh MN (2019) Long-term memory Harris' Hawk optimization for high dimensional and optimal power flow problems. IEEE Access 7:147596–147616
49. Islam MZ, Izzri N, Wahab A, Veerasamy V, Nasrun M, Nasir M (2020) A Harris Hawks optimization based single- and multi-objective optimal power flow considering environmental emission. Sustainability 12(13):5248
50. Niu P, Niu S, Liu N, Chang L (2019) The defect of the Grey Wolf optimization algorithm and its verification method. Knowl Based Syst 171:37–43
51. Yang X (2010) Nature-inspired metaheuristic algorithms, 2nd edn. Luniver Press, Cambridge
52. Imran M, Hashim R, Khalid NEA (2013) An overview of particle swarm optimization variants. Procedia Eng 53(1):491–496
53. Jaiswal KB, Kumar N, Seshadri P, Lakshminarayanan G (2015) Low power wallace tree multiplier using modified full adder. In: 2015 3rd international conference on signal processing, communication and networking (ICSCN), pp 1–4
54. Liao M-J, Su C-F, Chang C-Y, Wu A-H (2002) A carry-select-adder optimization technique for high-performance Booth-encoded wallace-tree multipliers. In: 2002 IEEE international symposium on circuits and systems. Proceedings (Cat. No. 02CH37353), 2002, vol 1, pp I–I
55. Itoh N, Naemura Y, Makino H, Nakase Y, Yoshihara T, Horiba Y (2001) A 600-MHz 54/spl times/54-bit multiplier with rectangular-styled Wallace tree. IEEE J Solid-State Circuits 36(2):249–257
56. Paradhasaradhi D, Prashanthi M, Vivek N (2014) Modified wallace tree multiplier using efficient square root carry select adder. In: 2014 international conference on green computing communication and electrical engineering (ICGCCEE), pp 1–5

57. Kuo T-Y, Wang J-S (2008) A low-voltage latch-adder based tree multiplier. In: 2008 IEEE international symposium on circuits and systems, pp 804–807
58. Khan S, Kakde S, Suryawanshi Y (2013) VLSI implementation of reduced complexity wallace multiplier using energy efficient CMOS full adder. In: 2013 IEEE international conference on computational intelligence and computing research, pp 1–4
59. Kshirsagar RD, Aishwarya EV, Vishwanath AS, Jayakrishnan P (2013) Implementation of pipelined booth encoded wallace tree multiplier architecture. In: 2013 international conference on green computing, communication and conservation of energy (ICGCE), pp 199–204
60. Reddy BNM, Sheshagiri HN, Vijayakumar BR, Shanthala S (2014) Implementation of low power 8-bit multiplier using gate diffusion input logic. In: 2014 IEEE 17th international conference on computational science and engineering, pp 1868–1871
61. Sarma R, Bhargava C, Dhariwal S, Jain S (2019) UCM: a novel approach for delay optimization. Int J Perform Eng 15(4):1190–1198
62. Bhattacharyya P, Kundu B, Ghosh S, Kumar V, Dandapat A (2014) Performance analysis of a low-power high-speed hybrid 1-bit full adder circuit. IEEE Trans Very Large Scale Integr Syst 23(10):2001–2008
63. Brest J, Maučec MS (2017) Single objective real-parameter optimization: algorithm jSO, pp 1311–1318
64. Jagodzi D (2017) A differential evolution strategy, vol 1, no 3, pp 1872–1876
65. Kommadath R (2017) Teaching learning based optimization with focused learning and its performance on CEC2017 functions, no. 1, pp 2397–2403
66. Tangherloni A, Rundo L, Nobile MS (2017) Proactive particles in swarm optimization: a settings-free algorithm for real-parameter single objective optimization problems, pp 1940–1946
67. Kamboj VK (2015) A novel hybrid PSO–GWO approach for unit commitment problem. Neural Comput Appl 27:1643–1655
68. Kuk JN, Gonc RA, Almeida CP, Venske SM, Pozo AT, Functions AC-B (2018) A new adaptive operator selection for NSGA-III applied to CEC 2018 many-objective benchmark, pp 7–12
69. Zhang H, Lee HJ, Ko S-B (2018) Efficient fixed/floating-point merged mixed-precision multiply-accumulate unit for deep learning processors. In: in 2018 IEEE international symposium on circuits and systems (ISCAS), pp 1–5
70. Shanthala S, Kulkarni SY (2009) VLSI design and implementation of low power MAC unit with block enabling technique. Eur J Sci Res 30:620–630
71. Hoang TT, Sjalander M, Larsson-Edefors P (2010) A high-speed, energy-efficient two-cycle multiply-accumulate (MAC) architecture and its application to a double-throughput MAC unit. IEEE Trans Circuits Syst I Regul Pap 57(12):3073–3081
72. Esmaeili SE, Al-Kahlili AJ, Cowan GER (2011) Low-swing differential conditional capturing flip-flop for LC resonant clock distribution networks. IEEE Trans Very Large Scale Integr Syst 20(8):1547–1551
73. RahulNarasimhan A, Subramanian RS (2015) High speed multiply-accumulator coprocessor realized for digital filters. In: 2015 IEEE international conference on electrical, computer and communication technologies (ICECCT), pp 1–4
74. Akbarzadeh N, Timarchi S, Hamidi AA (2015) Efficient multiply-add unit specified for DSPs utilizing low-power pipeline modulo $2n + 1$ multiplier. In: 2015 9th Iranian conference on machine vision and image processing (MVIP), pp 120–123
75. Karthikeyan KV, Babu R, Mathan N, Karthick B (2016) Performance analysis of an efficient MAC unit using CNTFET technology. Mater Today Proc 3(6):2525–2531

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.