Efficient FPGA Implementation of Sine Cosine Algorithm using High Level Synthesis

Jeng-Shyang Pan[1,4], Si-Qi Zhang[1], Shu-Chuan Chu[1], Chia-Cheng Hu[2], Jie Wu[3]*

[1] College of Computer Science and Engineering, Shandong University of Science and Technology, China
[2] College of Artificial Intelligence, Yango University, China
[3] School of Electrical and Information Engineering, Zhengzhou University of Light Industry, China
[4] Department of Information Management, Chaoyang University of Technology, Taiwan
jengshyangpan@gmail.com, zhangsiqichn@gmail.com, scchu0803@gmail.com, jjhwu@ksts.seed.net.tw, wujie@zzuli.edu.cn

Abstract

Sine Cosine Algorithm (SCA) finds the best solution to the optimization problem by the periodicity of sine and cosine trigonometric functions. However, it is computationally intensive and contains many parameters to be determined. Fortunately, there are FPGA platforms that can be used to overcome these limitations by improving latency. Sine and cosine calculation in library functions is very complex and time-consuming. Therefore, this paper proposes a hardware-accelerated CORDIC algorithm to improve the sine cosine trigonometric function that needs to be computed in the SCA algorithm. The proposed algorithm (HSCA) combines the accelerated SCA algorithm and the CORDIC algorithm. HSCA performance is tested by using six test functions run on the FPGA. The experimental results show that HSCA is 3.25 times faster and 33% fewer resource utilizations for solving optimization problems, and runs significantly faster on FPGAs with IP cores than on Soc chips in FPGAs. The performance of the HSCA algorithm is demonstrated by applying it to the TDOA localization problem.

Keywords: Sine Cosine Algorithm, Filed Programmable Gate Arrays, Vivado HLS tool, Optimization techniques, TDOA

1 Introduction

Sine Cosine Algorithm (SCA) [1-2] is a novel heuristic optimization algorithm proposed in recent years, which oscillates the optimization based on the mathematical model of sine and cosine functions. It has the advantages of a few parameters, a simple structure, and easy implementation. SCA has few parameters simple algorithm structure and still finds the appropriate optimal solution for handling complex problems [3]. SCA has been widely used in scientific research and industry. SCA has been used to solve computation problems such as future selection [4]. SCA is used in neural networks to optimize neural network problems [5]. SCA also provides optimal solutions for complex applications, such as path planning [6], power systems [7], machine learning [8], and wireless sensor network problems [9-13] that may not be able to meet real-time requirements.

There are many studies on improving the convergence speed, finding accuracy, and reducing falling into local optimum [14-17]. However, there are few kinds of research on improving the execution speed of SCA algorithms and reducing the resource utilization of SCA. Therefore, this paper proposes SCA running on the FPGA platform to improve the algorithm execution efficiency using the hardware acceleration technique.

In recent years, the performance improvement of general-purpose processors such as CPUs has slowed down, and to continue to meet the growing demand for energy-efficient computing in various industries, FPGAs, and representative devices for reconfigurable computing systems, have received widespread attention overnight in many emerging hot areas. Nowadays, FPGAs are introduced in the R&D process of many fields, such as artificial intelligence, extensive data analysis, network communication, and image processing. FPGAs have become a mainstream general-purpose computing technology. In cloud computing, numerous studies and industrial applications [18-19] have proven that FPGA technology can effectively improve the processing performance of various cloud loads while reducing power consumption [20]. In the field of edge computing, FPGAs are similarly widely used. Hsu-Chih Huang proposes a parallel meta-heuristic particle swarm optimization (PPSO) algorithm based on field-programmable gate arrays (FPGAs) [21]. The PPSO consists of three parallel PSOs and a communication operator in an FPGA chip. Pradeep R. Fernando reported on designing an IP core that implements a generic GA engine to solve these problems [22]. Specifically, the proposed GA IP core can be customized according to population size, number of generations, crossover and variation rates, random number generator seeds, and fitness functions. In 2020, Ahmed Hassanein presented a field-programmable gate array (FPGA)-based parallel BSO processor [23]. The development includes sequential modeling algorithms, deriving parallel versions, evaluating functions against a rich set of benchmarks, and performing thorough verification. Qiangqiang Jiang proposed a method for efficient implementation of parallel WOA on FPGA, called FPWOA [24]. It is developed for FPGA architecture features according

*Corresponding Author: Shu-Chuan Chu; E-mail: scchu0803@gmail.com
DOI: https://doi.org/10.70003/160792642024112506007

to the parallelization characteristics of the WOA algorithm. While pursuing higher data processing capability, edge computing also requires processors with miniaturization, low power consumption, and the ability to respond to various application scenarios flexibly. Soc FPGAs [25], which have become popular in recent years, fully cater to the above-level requirements by integrating ARM hard-core processors and FPGAs on a single chip. To date, SoC FPGA [26] products are widely used in smart cameras, autonomous vehicles, drones, video cameras, intelligent voice assistants, and other home electronics.

System development with FPGAs requires a lot of hardware fundamentals and programming in Verilog/VHDL, which is undoubtedly a vast learning cost for software engineers. Fortunately, developers can now use the High-Level synthesis [27] tool to develop hardware programs. HLS is a technique that automatically translates behavior described in a high-level language into hardware with the same functionality, hence the behavior-level synthesis. This technique intends to make hardware design much less complex so that circuits can be defined directly at a higher level of abstraction, significantly reducing hardware development time [28]. The purpose of high-level synthesis is to allow developers to focus less on hardware and algorithm-level design and implementation. Nowadays, high-level synthesis tools usually choose C, C++, OpenCL, and other languages as input [29]. These high-level languages have a higher level of abstraction and are. Therefore, more These high-level languages are more expressive and more efficient than low-level languages [30]. However, since these high-level languages do not contain timing information, they cannot be converted into circuit implementations.

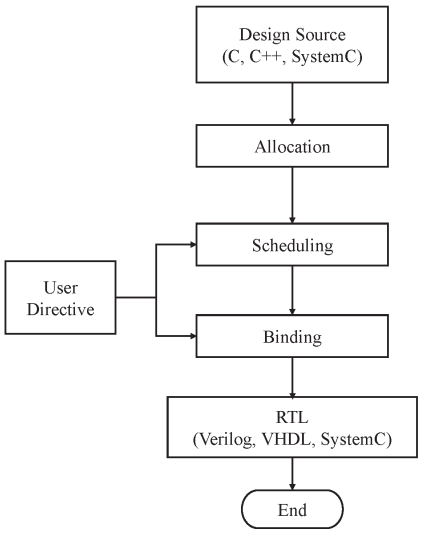


Figure 1. Flowchart of HLS

In Figure 1, HLS can be divided into three main processes: Allocation, Scheduling, and Binding. The Allocation step determines hardware resources, such as addition, multiplication, and registers. The Scheduling determines what operations are executed at each clock cycle. The Binding is to assign the hardware to each operation. The final output is a Verilog/VHDL file directly describing the hardware circuit. The advantage of using HLS is to automate the conversion from high-level language to hardware language and to design hardware with less resource consumption and faster computing speed through optimization. HLS makes it possible for software engineers to participate in hardware development.

Section 2 briefly introduces the SCA algorithm concept and the CORDIC Algorithm's basic concepts. Section 3 provides optimization methods to optimize the SCA and CORDIC algorithm of SCA. Section 4 shows the schematic designed implementation of HSCA on FPGA; In Section 5, the results of experiments are presented and discussed. Finally, Section 6 summarizes all the work in this paper and describes the future directions for in-depth research.

2 Sine Cosine Algorithm

SCA achieves the global search by fluctuating outward and completes the local development by fluctuating toward the optimal solution to gradually converge to the optimal global solution, which has the advantages of simple structure, high flexibility, few initial parameters, and easy implementation [31-32].

Description of the standard SCA algorithm: If the size of the population is N and has a D-dimensional search space, the i individual $(i=1,2,...,N)$ in this search space can map the solution of the optimization problem to the position of each individual. Firstly, the positions of individuals in the search space are generated randomly. Then the objective function is used to calculate the fitness values of all individuals, and the current optimal individual and position are found after sorting. The iterative process of individual updating is expressed as follows:

$$X_i^{t+1}\begin{cases} X_i^t + r_1 \times \sin(r_2) \times ||r_3 P_i^t - X_i^t||, & r_4 < 0.5 \\ X_i^t + r_1 \times \cos(r_2) \times ||r_3 P_i^t - X_i^t||, & r_4 \geq 0.5 \end{cases} \tag{1}$$

In Equation (1), t is the number of current iteration. P_i^t denotes the optimal individual position obtained in the t iteration. r_2, r_3 and r_4 are random factors with the corresponding values in the ranges $[0, 2\pi]$, $[0, 2]$ and $[0, 1]$ respectively. r_2 determines the distance moved during the iteration; r_3 is the random weight coefficient assigned to the current optimal solution; r_4 is the switching condition of the iterative equation of the sine and cosine function.

r_1 denotes the control factor, whose value is linearly decreasing and controls the fluctuation amplitude of the sine and cosine function and also determines the direction of movement of the iteration. If $r_1 < 1$, the solution will move from the current solution towards the region of the optimal solution position, and if $r_1 \geq 1$, it will move in the opposite

direction, and the expression of the control factor is as follows:

$$r_1 = a - a\frac{t}{T} \quad (2)$$

The a is a positive constant. T denotes the maximum number of iterations.

3 SCA and CORDIC Optimization

3.1 Optimization SCA

The detailed steps of SCA are analyzed and discussed. The main steps of the SCA algorithm are illustrated in Figure 2. The subsections analyze each part of the SCA in detail and optimize it using the HLS optimization tools.

3.1.1 Population Initialization Step

In the beginning, SCA generates random populations (candidate solutions). The initial value of each dimension of the particle is random, and the range of values is determined by the solution space of the problem to be solved. The inner loop initializes each dimension of the particle using a random number generated by a random number generator, which LFSR implements. The outer loop in Figure 2 contains the fitness function used to calculate the fitness value of each particle in the population. The fitness function will evaluate each solution, which will result in a fitness value of the solution, and the fitness value will determine the excellence of the solution. The fitness function in this paper consists of 6 test functions containing unimodal, multimodal, and complex problems.

```
Set i:=0; j:=0;
Check the input values scope;
Set N, ub, lb, D from input;
Initialize the Random Number Generator with seed
from input;
For i from 0 to N {
 #pragma HLS PIPLINE
    For j from 0 to  D
    #pragma HLS Unroll
        Pop[i][j]=random value;

   fitness[i] = Fitness(Pop[i]);

}
```

Figure 2. Population initialize pseudo code

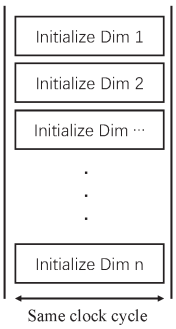


Figure 3. Loop unrolling

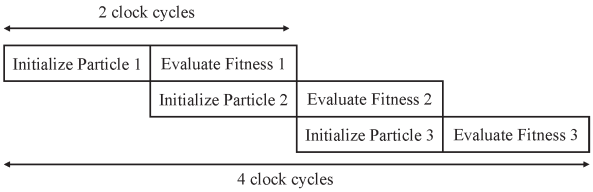


Figure 4. Loop pipelining

HLS detects as much parallelism in the code as possible to reduce latency [33]. The loop pipeline optimization allows the loop operations to be pipelined and the operations within the loop to overlap, as shown in Figure 4. Operations in a single for-loop that have no data dependencies can be optimized using unroll. Therefore, in Figure 3, we use loop unroll to optimize the inner for loop that initializes each particle's D-dimensional value.

3.1.2 Particle Updating Step

The particle update is calculated for each dimension of the particle using Equation (1) to obtain the new value. This step is the most time-consuming part of the whole algorithm execution. The pseudo of population updating is shown in Figure 5.

```
Initialize the Random Number Generator with seed from
input;
For i from 0 to N {
 Calculate r1 using Eq.(2)
 Generate random number for r2r3r4.
 #pragma HLS PIPLINE
    For j from 0 to  D
    #pragma HLS PIPLINE
        Calculation of sine and cosine values using the
CORDIC algorithm
        Compute Pop[i][j] Using Eq.(1)

}
```

Figure 5. Population update pseudo code

Since the calculation of Equation (1) has sine and cosine trigonometric functions, and the particle size in this paper is 10-30 dimensions, unrolling expansion will have a vast resource consumption. Therefore, the inner loop can be optimized using a pipeline so that all dimensions of a particle can be computed in a streamlined manner, where the sine and cosine trigonometric functions are executed using the hardware-accelerated CORDIC algorithm.

3.2 Optimization CORDIC

The CORDIC algorithm [34] is mainly for solving the problem of real-time computation. It is well suited for hardware implementation because it can eventually be decomposed into a series of simple algorithms implemented by addition, subtraction, shift, and lookup tables. Adding and subtracting is still accessible with embedded devices with weak computing power, such as microcontrollers or FPGAs [35]. However, it takes some effort to calculate trigonometric functions (sin, cos, tan) or even complex functions like hyperbolic, exponential, and logarithmic. Usually, these functions need to be converted to hardware-friendly through techniques such as lookup tables or approximations. This

algorithm is efficient in FPGAs and is often used in hardware algorithm implementations.

However, the CORDIC algorithm takes various forms, including three coordinate forms and two rotation modes, which correspond to different primitive function operation structures. Although J. Walther in 1971 gave agreed mathematical expressions for the CORDIC algorithm in three coordinate systems [36], which can be easily implemented in software programming for different CORDIC operations in practical applications, especially in the frontend of signal processing in the physical layer of communication, CORDIC operations often need to be implemented in FPGA hardware. The conventional practice is to design specific RTL code according to the requirements of different function operations, which leads to a significant limitation of code flexibility and portability.

The CORDIC algorithm uses a constant approximation method, and it only requires a simple shift and addition and subtraction to complete the numerical calculation of trigonometric functions. This algorithm improves the operational familiarity of the function and, at the same time, saves the FPGA resources. Thus, it is widely used in FPGA development.

Calculating the values of $\sin\theta$ and $\cos\theta$ can be transformed into finding the coordinate position $P_t(x_t, y_t)$ of a point $P_0(x_0, y_0)$ after rotating θ around the coordinate origin, by the coordinate rotation transformation formula:

$$\begin{bmatrix} x_{target} \\ y_{target} \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{pmatrix} x_0 \\ y_0 \end{pmatrix} = \frac{1}{\sqrt{1+\tan^2\theta}} \begin{pmatrix} 1 & -\tan\theta \\ \tan\theta & 1 \end{pmatrix} \quad (3)$$

The coordinate rotation transformation can be equated to a series of coordinate rotation changes with direction di and angle θ_i:

$$\theta = \sum_{i=0}^{n} d_i \theta_i \quad (4)$$

$$\lim_{i \to \infty} \theta_i = 0 \quad (5)$$

$$d_i = \begin{cases} 1, & \sum(\theta_i - \theta) \geqslant 0 \\ -1, & \sum(\theta_i - \theta) < 0 \end{cases} \quad (6)$$

Substituting Equation (4) into Equation (3), the following equation is obtained:

$$K(n) = \prod_{i=0}^{n} \frac{1}{\sqrt{1+\tan^2\theta}} \quad (7)$$

If the value of $\tan\theta_i$ is restricted to a power of 2, then the rotation operation can be reduced to data shifting (multiplication) and addition. Specifically, it can be set to $\tan\theta_i = 2 - i$. The rotation matrix then becomes:

$$\begin{bmatrix} x_{target} \\ y_{target} \end{bmatrix} = K(n) \begin{bmatrix} 1 & -d_0 2^0 \\ d_0 2^0 & 1 \end{bmatrix} \cdots \begin{bmatrix} 1 & -d_n 2^n \\ d_n 2^n & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} \quad (8)$$

K(n) tends to stabilize as n increases. The value of K can be obtained as follows:

$$K = \lim_{n \to \infty} \prod_{i=0}^{n} \frac{1}{\sqrt{1+2^{-2i}}} \approx 0.60725252935 \quad (9)$$

To simplify the calculation one can also take x_0 as K and y_0 as 0. We can obtain the sin, cosine matrix:

$$\begin{bmatrix} \sin\theta \\ \cos\theta \end{bmatrix} = \begin{bmatrix} 1 & -d_0 2^0 \\ d_0 2^0 & 1 \end{bmatrix} \cdots \begin{bmatrix} 1 & -d_i 2^i \\ d_i 2^i & 1 \end{bmatrix} \begin{bmatrix} K \\ 0 \end{bmatrix} \quad (10)$$

Here, the CORDIC algorithm starts from the positive of the x-axis, which corresponds to an angle of 0 degrees, and then performs four rotations either clockwise or counterclockwise, each with a smaller and smaller angle, to finally obtain the target angle θ. Once the rotation is completed, the angle obtained is very close to the theoretical one. If the length of the vector is K, then the final vector components in x, y correspond to $\cos\theta$ and $\sin\theta$, respectively. The key to improving the CORDIC algorithm is to improve the computational efficiency of the above process.

4 Design and Implementation of SCA on FPGA

The SCA is suitable for use in general computing environments. The prototype implementation of the HSCA IP core is based on the Zynq [37] FPGA platform, as shown in Figure 6. The PL part of the Zynq is the FPGA, where the HSCA IP core can be integrated. The axi-lite [38] peripheral bus is applied to integrate the HSCA IP core and the DMA as devices in the system. At the same time, The DMA is used to connect the HSCA IP core to the PS high-performance bus port for direct access to DDR memory via the axis protocol. This way, the HSCA IP core is independent of the PS and can leave PS resources to other processes.

The PS part of Zynq is a dual-core ARM CPU, which can run a Linux OS for system and data transfer control. The PS acts as a front-end to receive the parameters of the SCA entered by the user and write them to the memory. In our proposed work, the specification is done in C using the Vivado HLS tool. These specifications are then translated into hardware images and programmed on the FPGA for software end functions.

The modules in Figure 6 are emulated according to the user-specified HSCA operator. All modules are written using the Vivado HLS tool. The individual modules in the HSCA designed in Figure 6 are described as follows:

The main task of the initialization module is to obtain from DDR memory the parameters needed for the algorithm's execution, such as the number of populations,

dimensions, and upper and lower bounds of the solution space. When the initialization module is called, it generates the initial population and some initial variables based on the parameters. It sends them to the PUM after the initialization module is executed.

After receiving the initial population from the initialization module, the particle update module calls FM to evaluate the population for fitness. The PUM module uses parallelism to speed up the update and evaluation of particles. Now, the PUM resets itself and repeats the selection process.

FM evaluates the particles received from the PUM and returns the fitness value to memory through the computation of the function. FM can be replaced with any problem that needs to be solved, and only the solution space boundaries and dimensions of the problem need to be known.

When the CORDIC module receives the angle information from the PUM input, it is based on the random value sent by the RNG. The initial θ starts with the positive half-axis of the X-axis, corresponding to an angle of 0 degrees, and then performs clockwise or counterclockwise rotations, each with smaller and smaller angles, to finally obtain the target angle θ. The solution process can be viewed as a successive multiplication of matrices, which can be optimized using a similar technique process.

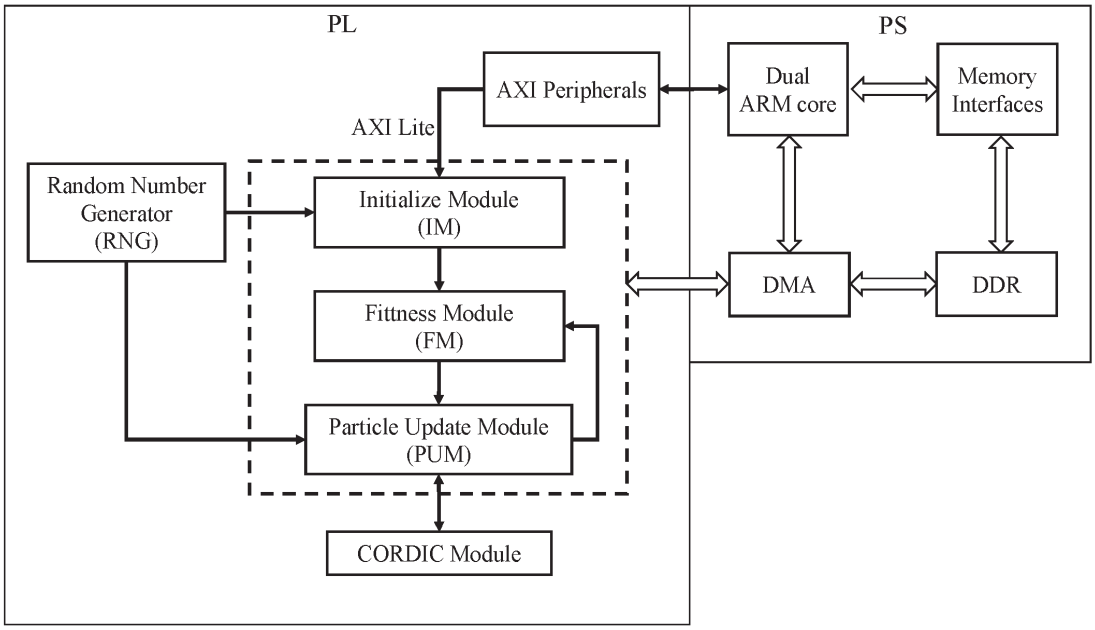


Figure 6. The schematic design of HSCA on FPGA

5 Experiment and Analysis

The optimized Verilog code of SCA by applying different optimization techniques was synthesized on Xilinx HLS software on the Zynq Ultrascale+ AXU2CG-E device. In order to verify the performance of the proposed HSCA algorithm, six representative benchmark functions are selected for testing and compared with the standard SCA algorithm and the improved SCA using library functions sine and cosine. The descriptions of the benchmark functions are shown in Table 2.

In Table 1, the typical CORDIC algorithm uses 0 DSP compared to the Sine and Cosine functions in the library, while the Sine Cosine function uses 86 DSPs. It also reduces the occupancy of LUTs by 84% and FFs by 86%. The latency is reduced by 87% for the improved CORDIC algorithm, but the LUT footprint is increased because the optimization requires more LUTs. The CORDIC algorithm has improved in latency, LUT, and DSP utilization.

In Table 2, f1-f5 are unimodal functions, f6-f9 are multimodal functions, and f10 are complex functions. These three types of test functions can represent most problems in the real world. In order to ensure the fairness of experiments, each function is executed 30 times, and the execution times are averaged. Each function is tested with uniform parameters, a maximum number of iterations of 1000, and a population size of 100.

Table 3 obtains this result for the algorithm with 30 particles and 1000 max iterations. The HSCA executed on FPGA significantly improved each function. Among them, it is at least 16 times faster on the unimodal function, especially on the f1 function, where HSCA executes only 12ms, which is 27 times faster compared to the execution speed of DE. When optimizing the multimodal function, the improvement is 11 and 10 times for f4 and f5, respectively, and 20 times for f3. The reason is that the f4 and f5 functions include the calculation of the cosine function, which increases the complexity of the calculation and leads to an increase in execution time. When solving complex problems, the algorithm still has about 15 times improvement because the algorithm is executed on the hardware circuit of FPGA, the operations for data are obtained from registers, and the circuit for performing basic operations can also be executed in parallel.

Table 1. Latency and resources utilization of CORIDC algorithm and sine, cosine function

Name	Sine, Cosine library functions	Typical CORDIC without optimization	Improved CORDIC algorithm
Latency (ns)	35	38	5
LUT	5661	940	2474
FF	2045	290	253
DSP	86	0	0

Table 2. Benchmark functions

Func	Expression	$f_{\min}$	Range
f_1	$f(x)=\sum_{i=1}^{D} x_i^2$	0	[-10,10]
f_2	$f(x)=\sum_{i=1}^{D-1} 100(x_{i+1}-x_i^2)^2+(x_i-1)^2$	0	[-2.048,2.048]
f_3	$f(x)=\sum_{i=1}^{D} ix_i^2$	0	[-10,10]
f_4	$f(x)=\sum_{i=1}^{D} \lvert x_i\rvert+\prod_{i=1}^{D} \lvert x_i\rvert$	0	[-10,10]
f_5	$f(x)=\max_{1\leqslant i\leqslant D} \lvert x_i\rvert$	0	[-100,100]
f_6	$f(x)=2x_1^2-1.05x_1^4+\frac{x_1^6}{6}+x_1x_2+x_2^2$	0	[-5,5]
f_7	$f(x)=\sum_{i=1}^{D}-20e^{-0.2\sqrt{\frac{1}{d}\sum_{i=1}^{D}x_i^2}}-e^{\frac{1}{d}\sum_{i=1}^{D}\cos(2\pi x_i)}+20+e$	0	[-32.768,32.768]
f_8	$f(x)=\sum_{i=1}^{D}(x_i^2-10\cos(2\pi x_i)+10)$	0	[-5.12,5.12]
f_9	$f(x)=\sum_{i=1}^{D}\frac{x_i^2}{4000}-\prod_{i=1}^{d}\cos\left(\frac{x_i}{\sqrt{i}}\right)+1$	0	[-600,600]
f_{10}	$f_6=\frac{1}{2}\sum_{i=1}^{D}(x_i^4-10x_i^2+5x_i)$	-39.16599	[-5,5]

Table 3. Execution time comparison of each related algorithm (ms)

Function	SCA	GWO	PSO	DE	HSCA	Speed Up
F1	460	630	351	332	12	27
F2	539	697	549	492	29	16
F3	542	732	447	510	34	13
F4	446	657	358	360	21	17
F5	472	660	365	375	17	21
F6	102	134	80	106	4	20
F7	648	785	484	658	44	11
F8	653	789	539	711	53	10
F9	654	814	563	758	75	7
F10	535	686	427	484	28	15

Table 4. Latency and resources utilization of SCA

Name	Typical SCA without optimization	Typical SCA with HLS optimization	Improved SCA with CORDIC optimization
Clock Cycles	16802207	10265107	5162007
Clk. Per(ns)	10	10	10
Latency(ms)	16.8	10.27	5.16
LUT	18321 (38%)	19581 (42%)	10481 (22%)
FF	9292 (10%)	8033 (8%)	3766 (4%)
BRMS	72 (48%)	19 (12%)	11 (7%)
DSP	195 (81%)	163 (67%)	29 (12%)
Power(w)	2.7	2.641	2.434
Speed Up	1	1.61	3.25

Table 5. Comparison for the running Times and results of SCA and HSCA (Population size is 256)

Function	SCA		HSCA		Speed Up
	Mean	Time(s)	Mean	Time(s)	
F1	5.27E-01	2.184	**4.02E-01**	0.353	6.187
F2	**3.93E+01**	2.388	5.97E+01	0.542	4.406
F3	**7.02E+01**	2.438	7.26E+01	0.358	6.810
F4	**2.51E+00**	2.195	3.76E+00	0.365	6.014
F5	**2.05E+00**	2.213	6.23E+03	0.359	6.614
F6	1.03E+00	2.264	1.03E+00	0.381	5.942
F7	2.41E+00	4.652	**2.74E-05**	0.481	9.671
F8	1.70E+01	3.667	**6.04E+00**	0.515	7.120
F9	1.00E+01	5.672	**1.02E+00**	0.524	10.824
F10	-2.91E+02	3.370	**-3.54E-02**	0.426	7.910

Table 6. Comparison for the running Times and results of SCA and HSCA (Population size is 512)

Function	SCA		HSCA		Speed Up
	Mean	Time(s)	Mean	Time(s)	
F1	4.85E-01	5.774	**4.54E-01**	0.646	8.938
F2	**3.32E+01**	5.081	3.76E+01	0.929	5.469
F3	2.88E+00	5.776	**2.52E+00**	0.661	8.738
F4	2.21E+00	5.390	**1.76E+00**	0.688	7.834
F5	**4.56E-01**	4.394	1.46E+00	0.761	5.774
F6	1.03E+00	6.396	1.03E+00	0.667	9.589
F7	**2.25E+00**	7.462	2.33E+00	0.849	8.789
F8	2.01E+01	9.491	**1.65E+01**	1.039	9.135
F9	**1.00E+00**	11.305	1.04E+00	1.156	9.779
F10	-2.99E+02	8.906	**-3.12E+02**	0.914	9.744

The combined results before and after SCA improvement are shown in Table 4. After the improvement of a typical SCA, it can be seen that 61% reduces the Latency, and the execution speed is increased by 1.61 times. 4% increases the resource usage of the LUT because the FPGA needs to use more lookup tables to find data after optimizing the loop parallelism. At the same time, parallelism increases the reuse rate of hardware resources, leading to a decrease in DSP utilization. For typical SCA and HLS-optimized SCA, the sine and cosine functions in the library file are used. The improved SCA is further optimized using the CORDIC algorithm, which further pipelining the operation process of solving trigonometric functions, reduces the DSP occupancy, and further improves the execution efficiency of the algorithm to 3.25 times. FPGA power consumption is divided into static power consumption and dynamic. Static power consumption is when the device is set up and not running. Dynamic power consumption is the average power obtained from user logic switch utilization and switching activities. The main difference between the power consumption of the three strategies in Table 4 is focused on the dynamic power consumption component. Because static power consumption is the 'thermal power' generated at the point on the FPGA, including any power loss on the device, this component accounts for about 10% of the total power consumption. The algorithm program of Typical SCA is not optimized, so the program execution steps and resource consumption are higher compared to HLS-optimized SCA, as shown by the experimental results. For HSL-optimized SCA, the parallelism of the algorithm execution process is increased, resulting in less resource consumption, less dynamic power consumption in this part of the device, and more efficient execution. Improved SCA with CORDIC optimization reduces the program's complexity and decreases resource utilization because the sin and cos functions are not used. HLS also optimizes the algorithm to reduce the execution time. These two parts of optimization reduce the dynamic power consumption and achieve the purpose of reducing power consumption.

The execution time and results of Typical SCA and FPGA implementation of parallel SCA simplified as HSCA, respectively, are shown in Table 5 to Table 6. The maximum number of iterations is set to 1000. The dimensionality of all tested functions is set to 256. The number of populations is set to 256 and 521. In addition, mean and time are the average results and running times obtained from 30 independent runs of each function. Comparing Tables 5 to Table 6 shows that as the population size increases, both SCA and HSCA find better solutions, which indicates that the population size affects the algorithm's ability to find the best solution. The SCA provided excellent optimization ability on the unimodal function when the population size was 256. On the contrary, HSCA achieved superior results on the multimodal function. When the population size is 512, more particles make the difference between SCA and HSCA in the unimodal function, and the multimodal function is not apparent. Among them, HSCA achieved five optimal results. Overall, comparisons of SCA and HSCA utilized to solve ten benchmark functions reveal the efficient optimization of HSCA. It can be seen from Table 5 to Table 6 that the running time of both SCA and HSCA doubles as the number of particles doubles. Moreover, for the same number of particles, both SCA and HSCA have less execution time in the unimodal function than in the multimodal function. Multimodal functions have higher arithmetic complexity than unimodal functions. It is concluded from the execution time that HSCA is more suitable for large-scale complex problems and can significantly improve algorithm efficiency and reduce power consumption.

6 HSCA for TDOA Localization Problem

The TDOA localization method determines the location of the target node by measuring the propagation time difference of the radio wave from the target node propagation signal to multiple anchor nodes. Multiple TDOA measurements form a hyperbolic system of equations that is nonlinear. It is difficult to solve. In this paper, this hyperbolic equation is solved by HSCA algorithm.

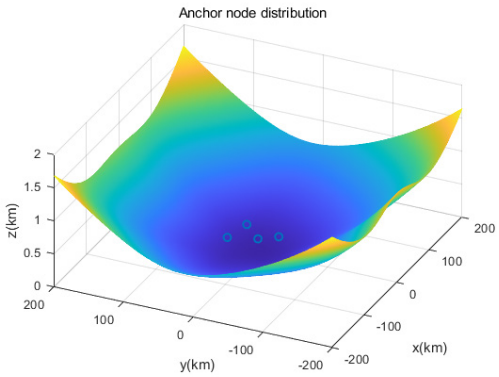


Figure 7. Anchor node distribution

Assume that the anchor nodes are distributed in 3D space in a Y-shape, as shown in Figure 7. $(\mathring{x},\mathring{y},\mathring{z})$ is the location of the target node u to be estimated, and (x_i,y_i,z_i) is the location of the known anchor node s_i. The distance from u to s_i is $\mathring{r}_i$. $\mathring{R}_{i,1}$ denotes the exact distance difference between the distance difference from u to $s_i(i\neq 1)$ and the distance difference from s_1, and the measured value is noted as $R_{i,1}$.

$$R_{i,1}=cd_{i,1}=\mathring{R}_{i,1}+cn_{i,1}=R_i-R_1+cn_{i,1} \tag{11}$$

$$R_i=\sqrt{(x_i-x)^2+(y_i-y)^2+(z_i-z)^2},\ i=2,...,M \tag{12}$$

Where, c is the electromagnetic wave propagation velocity, $d_{i,1}$ is the measured value of TDOA. $n_{i,1}$ is the

noise at the time of TDOA measurement, where the noise is a Gaussian white noise with independent homogeneous distribution of variance σ^2.

$$h = cn + g \tag{13}$$

$$h = [R_{2,1}, R_{3,1}, ..., R_{m,1}]^{T(m-1)\times 1} \tag{14}$$

$$g = \begin{bmatrix} R_2 - R_1 \\ \vdots \\ R_m - R_1 \end{bmatrix} \tag{15}$$

where $M > 3$, the maximum likelihood method is used to estimate $u(x,y)$. Since $R_{i,1}$ follows a Gaussian distribution with mean $(R_i - R_i)$ and variance σ^2, the likelihood function is as follows.

$$\prod_{i=2}^{M}\left[\frac{1}{\sqrt{2\pi}\,\sigma}\exp\left\langle-\frac{(R_{i+1,1}-R_{i+1}+R_1)^2}{2\sigma^2}\right\rangle\right] = \left[\frac{1}{\sqrt{2\pi}\,\sigma}\right]^{M-1}\cdot\exp\left\langle-\frac{(h-g)^T(h-g)}{2\sigma^2}\right\rangle \tag{16}$$

Finding the coordinates of the point u that maximizes the likelihood function is equivalent to finding Equation (16).

$$(x,y) = arg\{\min[(h-g)^T(h-g)]\} \tag{17}$$

The fitness function is:

$$f(p_i) = \frac{1}{(h-g)^T(h-g)},\ p_i = (x_i, y_i) \tag{18}$$

In this paper, the MSE (Mean-Square Error) is used as a measure of positioning accuracy.

$$MSE = ||u^i - u^0||_2 \tag{19}$$

Where, u^i is the optimal solution obtained by the optimization algorithm. The experiment uses 100 particles, and the maximum value of iterations is 500. The convergence curve of the algorithm localization results at the total noise error of -20 dB is shown in Figure 8.

In Figure 8, it is shown that the HSCA algorithm converges faster and uses fewer iterations for TDOA localization. When the number of iterations reaches 200, the results obtained by the HSCA algorithm are completely close to the real target position.

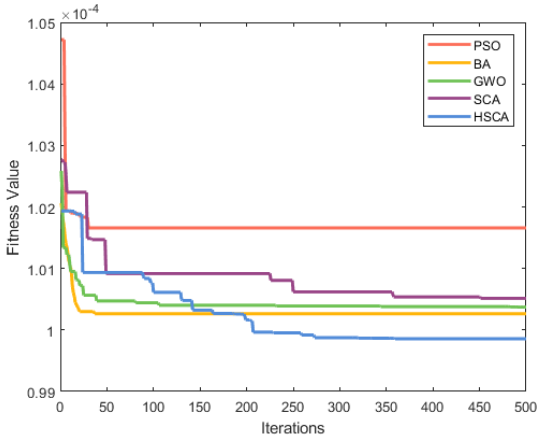


Figure 8. Convergence curve of each algorithm

7 Conclusion

This paper presents HSCA, based on an FPGA platform running using HLS software implementation. Then, the circular optimization methods in HLS, which include pipeline and unrolled, are used to optimize the HSCA, and the optimized CORDIC algorithm is used to compute the trigonometric functions. The experimental results show that the HSCA algorithm can speed up at least ten times compared to other algorithms and at least 3.25 times compared to a typical SCA while resource usage is reduced. Finally, HSCA is applied to TDOA localization and achieves fast convergence speed and accuracy. In future work, it is planned. To parallelize the algorithm to improve its execution efficiency [39-42]. In addition, additional optimization strategies will be considered to reduce power and system energy consumption [43-46].

References

[1] S. Mirjalili, SCA: a sine cosine algorithm for solving optimization problems, *Knowledge-based systems*, Vol. 96, pp. 120-133, March, 2016.

[2] A. B. Gabis, Y. Meraihi, S. Mirjalili, A. Ramdane-Cherif, A comprehensive survey of sine cosine algorithm: variants and applications, *Artificial Intelligence Review*, Vol. 54, No. 7, pp. 5469-5540, October, 2021.

[3] P. Liao, C. Sun, G. Zhang, Y. Jin, Multi-surrogate multi-tasking optimization of expensive problems, *Knowledge-Based Systems*, Vol. 205, Article No. 106262, October, 2020.

[4] A. I. Hafez, H. M. Zawbaa, E. Emary, A. E. Hassanien, Sine cosine optimization algorithm for feature selection, *2016 international symposium on innovations in intelligent systems and applications (INISTA)*, Sinaia, Romania, 2016, pp. 1-5.

[5] S. Loussaief, A. Abdelkrim, Convolutional neural network hyper-parameters optimization based on

genetic algorithms, *International Journal of Advanced Computer Science and Application*, Vol. 9, No. 10, pp. 252-266, 2018.

[6] P. K. Das, Hybridization of kidney-inspired and sine–cosine algorithm for multi-robot path planning, *Arabian Journal for Science and Engineering*, Vol. 45, No. 4, pp. 2883-2900, April, 2020.

[7] K. Dasgupta, P. K. Roy, V.Mukherjee, Power flow based hydro-thermal-wind scheduling of hybrid power system using sine cosine algorithm, *Electric Power Systems Research*, Vol. 178, Article No. 106018, January, 2020.

[8] J. Xia, D. Yang, H. Zhou, Y. Chen, H. Zhang, T. Liu, A. A. Heidari, H. Chen, Z. Pan, Evolving kernel extreme learning machine for medical diagnosis via a disperse foraging sine cosine algorithm, *Computers in Biology and Medicine*, Vol. 141, Article No. 105137, February, 2022.

[9] S. Zhang, F. Fan, W. Li, S.-C. Chu, J.-S. Pan, A parallel compact sine cosine algorithm for tdoa localization of wireless sensor network, *Telecommunication Systems*, Vol. 78, No. 2, pp. 213-223, October, 2021.

[10] Q.-W. Chai, S.-C. Chu, J.-S. Pan, P. Hu, W.-M. Zheng, A parallel woa with two communication strategies applied in dv-hop localization method, *EURASIP Journal on Wireless Communications and Networking*, Vol. 2020, No. 1, pp. 1-10, February, 2020.

[11] S.-C. Chu, Z.-G. Du, J.-S. Pan, Symbiotic organism search algorithm with multi-group quantum-behavior communication scheme applied in wireless sensor networks, *Applied Science*, Vol. 10, No. 3, Article No. 930, February, 2020.

[12] J.-S. Pan, T.-K. Dao, T.-S. Pan, T.-T. Nguyen, S.-C. Chu, J. F. Roddick, An improvement of flower pollination algorithm for node localization optimization in wsn, *Journal of Information Hiding and Multimedia Signal Processing*, Vol. 8, No. 2, pp. 486-499, March, 2017.

[13] S. C. Chu, T.-K. Dao, J. S. Pan, T.-T. Nguyen, Identifying correctness data scheme for aggregating data in cluster heads of wireless sensor network based on naive bayes classification, *EURASIP Journal on Wireless Communications and Networking*, Vol. 2020, No. 1, pp. 1-15, February, 2020.

[14] M. A. Elaziz, D. Oliva, S. Xiong, An improved opposition-based sine cosine algorithm for global optimization, *Expert Systems with Applications*, Vol. 90, pp. 484-500, December, 2017.

[15] S. Gupta, K. Deep, Improved sine cosine algorithm with crossover scheme for global optimization, *Knowledge-Based Systems*, Vol. 165, pp. 374-406, February, 2019.

[16] S. Xiao, H. Wang, W. Wang, Z. Huang, X. Zhou, M. Xu, Artificial bee colony algorithm based on adaptive neighborhood search and gaussian perturbation, *Applied Soft Computing*, Vol. 100, Article No. 106955, March, 2021.

[17] J. Li, D.-D. Xiao, T. Zhang, C. Liu, Y.-X. Li, G.-G. Wang, Multi-swarm cuckoo search algorithm with q-learning model, *The Computer Journal*, Vol. 64, No. 1, pp. 108-131, January, 2021.

[18] E. Monmasson, M. N. Cirstea, FPGA design methodology for industrial control systems—a review, *IEEE transactions on industrial electronics*, Vol. 54, No. 4, pp. 1824-1842, August, 2007.

[19] J. S. Pan, X.-X. Sun, S.-C. Chu, A. Abraham, B. Yan, Digital watermarking with improved sms applied for qr code, *Engineering Applications of Artificial Intelligence*, Vol. 97, Article No. 104049, January, 2021.

[20] V. George, H. Zhang, J. Rabaey, The design of a low energy FPGA, *Proceedings of the 1999 International Symposium on Low Power Electronics and Design*, San Diego, CA, 1999, pp. 188-193.

[21] H.-C. Huang, FPGA-based parallel metaheuristic PSO algorithm and its application to global path planning for autonomous robot navigation, *Journal of Intelligent & Robotic Systems*, Vol. 76, No. 3-4, pp. 475-488, December, 2014.

[22] P. R. Fernando, S. Katkoori, D. Keymeulen, R. Zebulum, A. Stoica, Customizable FPGA IP core implementation of a general-purpose genetic algorithm engine, *IEEE Transactions on Evolutionary Computation*, Vol. 14, No. 1, pp. 133-149, February, 2010.

[23] A. Hassanein, M. El-Abd, I. Damaj, H. U. Rehman, Parallel hardware implementation of the brain storm optimization algorithm using FPGAs, *Microprocessors and Microsystems*, Vol. 74, Article No. 103005, April, 2020.

[24] Q. Jiang, Y. Guo, Z. Yang, X. Zhou, A parallel whale optimization algorithm and its implementation on FPGA, *2020 IEEE Congress on Evolutionary Computation (CEC),* Glasgow, UK, 2020, pp. 1-8.

[25] G. Korcyl, P. Bialas, C. Curceanu, E. Czerwiński, K. Dulski, B. Flak, Evaluation of single-chip, real-time tomographic data processing on FPGA soc devices, *IEEE transactions on medical imaging*, Vol. 37, No. 11, pp. 2526-2535, November, 2018.

[26] K. Wang, N. Xu, K. Hu, An FPGA fast combination placement optimization algorithm research, *2017 IEEE 2nd Advanced Information Technology, Electronic and Automation Control Conference (IAEAC)*, Chongqing, China, 2017, pp. 1258-1262.

[27] R. Nane, V.-M. Sima, C. Pilato, J. Choi, B. Fort, A. Canis, A survey and evaluation of FPGA high-level synthesis tools, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 35, No. 10, pp. 1591-1604, October 2016.

[28] V. Krishnan, S. Katkoori, A genetic algorithm for the design space exploration of datapaths during high-level synthesis, *IEEE transactions on evolutionary computation*, Vol. 10, No. 3, pp. 213-229, June, 2006.

[29] A. Cornu, S. Derrien, D. Lavenier, Hls tools for FPGA: Faster development with better performance, *International Symposium on Reconfigurable Computing: Architectures, Tools and Applications*, Belfast, UK, 2011, pp. 67-78.

[30] C. Li, Y. Bi, Y. Benezeth, D. Ginhac, F. Yang, High-level synthesis for fpgas: code optimization strategies for real-time image processing, *Journal of Real-Time*

Image Processing, Vol. 14, No. 3, pp. 701-712, March, 2018.
[31] H. Wang, W. Wang, Z. Wu, Particle swarm optimization with adaptive mutation for multimodal optimization, *Applied Mathematics and Computation*, *Springer*, Vol. 221, pp. 296-305, September, 2013.
[32] Q. Yang, S.-C. Chu, J.-S. Pan, C.-M. Chen, Sine cosine algorithm with multigroup and multistrategy for solving CVRP, *Mathematical Problems in Engineering*, Vol. 2020, No. 1, pp. 1-10, March, 2020.
[33] R. M. Rizk-Allah, R. A. El-Sehiemy, G.-G. Wang, A novel parallel hurricane optimization algorithm for secure emission/economic load dispatch solution, *Applied Soft Computing*, Vol. 63, pp. 206-222, February, 2018.
[34] B. Lakshmi, A. S. Dhar, Cordic architectures: A survey, *VLSI design*, Vol. 2010, No. 1, pp. 1-19, March, 2010.
[35] R. Andraka, A survey of cordic algorithms for FPGA based computers, *Proceedings of the 1998 ACM/SIGDA Sixth International Symposium on Field Programmable Gate Arrays*, Monterey, CA, 1998, pp. 191-200.
[36] J. S. Walther, A unified algorithm for elementary functions, *g): Proceedings of spring joint computer conference*, Atlantic City New Jersey, 1971, pp. 379-385. 1971.
[37] S. Ahmad, V. Boppana, I. Ganusov, V. Kathail, V. Rajagopalan, R. Wittig, A 16-nm multiprocessing system-on-chip field-programmable gate array platform, *IEEE Micro*, Vol. 36, No. 2, pp. 48-62, March-April, 2016.
[38] M. Ramirez, M. Daneshtalab, J. Plosila, P. Liljeberg, Noc-axi interface for fpga-based mpsoc platforms, *22nd International Conference on Field Programmable Logic and Applications (FPL)*, Oslo, Norway, 2012, pp. 479-480.
[39] C. Sun, Y. Jin, R. Cheng, J. Ding, J. Zeng, Surrogate-assisted cooperative swarm optimization of high-dimensional expensive problems, *IEEE Transactions on Evolutionary Computation*, Vol. 21, No. 4, pp. 644-660, August, 2017.
[40] X. Xue, A compact firefly algorithm for matching biomedical ontologies, *Knowledge and Information Systems*, Vol. 62, No. 7, pp. 2855-2871, July, 2020.
[41] X. Xue, Compact memetic algorithm-based process model matching, *Soft Computing*, Vol. 23, No. 13, pp. 5249-5257, July, 2019.
[42] X. Zou, L. Wang, Y. Tang, Y. Liu, S. Zhan. F. Tao, Parallel design of intelligent optimization algorithm based on FPGA, *The International Journal of Advanced Manufacturing Techonology*, Vol. 94, No. 9-12, pp. 3399-3412, February, 2018.
[43] S.-C. Chu, T.-T Wang, A. R. Yildiz, J.-S. Pan, Ship rescue optimization: a new metaheuristic algorithm for solving engineering problems, *Journal of Internet Technology*, Vol. 25, No. 1, pp. 61-78, January, 2024.
[44] M. Zhu, S.-C. Chu, Q. Yang, W. Li, J.-S. Pan, Compact sine cosine algorithm with multigroup and multistrategy for dispatching system of public transit vehicles, *Journal of Advanced Transportation*, Vol. 2021, pp. 1-16, March, 2021.
[45] E. Alqudah, A. Jarrah, Parallel implementation of genetic algorithm on fpga using vivado high level synthesis, *International Journal of Bio-Inspired Computation*, Vol. 15, No. 2, pp. 90-99, March, 2020.
[46] Z.-C. Dou, S.-C. Chu, Z. Zhuang, A. R. Yildiz, J.-S. Pan, GBRUN: a gradient search-based binary runge kutta optimizer for feature selection, *Journal of Internet Technology*, Vol. 25, No. 3, pp. 341-353, May, 2024.

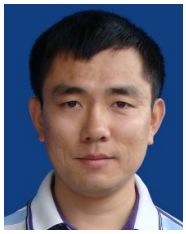
Biographies

Jeng-Shyang Pan received the B.S. degree in electronic engineering from the National Taiwan University of Science and Technology in 1986, the M.S. degree in communication engineering from National Chiao Tung University, Taiwan, in 1988, and the Ph.D. degree in electrical engineering from the University of Edinburgh, U.K., in 1996. He is currently the Professor of Shandong University of Science and Technology. He is the IET Fellow, U.K., and has been the Vice Chair of the IEEE Tainan Section and Tainan Chapter Chair of IEEE Signal Processing Society. His current research interest includes the information hiding, artificial intelligence and wireless sensor networks.

Si-Qi Zhang received her B.S. degree from Nanyang Institute of Technology, China, in 2020. He is currently pursuing the master degree with the Shandong University of Science and Technology, Qingdao, China. His recent research interests include swarm intelligence, node location, and FPGA.

Shu-Chuan Chu received a Ph.D. degree in 2004 from the School of Computer Science, Engineering and Mathematics, Flinders University of South Australia. She joined Flinders University in December 2009 after 9 years at the Cheng Shiu University, Taiwan. She has been a Research Fellow and Associate Professor in the College of Science and Engineering of Flinders University, Australia since December 2009. Currently, she is a Research Fellow with a Ph.D. advisor in the College of Computer Science and Engineering of Shandong University of Science and Technology from September 2019. She also serves as an editorial board member for Engineering Applications of Artificial Intelligence (EAAI), Journal of Internet Technology (JIT) and Research Reports on Computer Science (RRCS). Her research interests are mainly in Swarm Intelligence, Intelligent Computing, and Wireless Sensor Networks.

Chia-Cheng Hu received the M.S. degree in Engineer Science from National Cheng Kung University in 1995. He received his Ph.D. in department of Computer Science and Information Engineering, National Taiwan University, Taiwan in 2005. He is currently a Professor with the College of Artificial Intelligence, Yango University.



Jie Wu received the M.Eng. degree in electrical engineering from the Hubei University of Technology, Wuhan, China, in 2005, and the Ph.D. degree from the VŠB-Technical University of Ostrava, Czech Republic, in 2012. He worked with the Hubei University of Technology. Since 2017, he has been a Visiting Professor with the Department of Electrical and Computer Engineering, University of Kentucky, Lexington, KY, USA. He is currently an Associate Professor with the Zhengzhou University of Light Industry, Zhengzhou, China. He works in a multi-disciplinary environment involving artificial intelligence and electrical engineering.