Research Article

Pipeline Implementation of Polyphase PSO for Adaptive Beamforming Algorithm

Shaobing Huang, Li Yu, Fangjian Han, and Yiwen Luo

School of Electronic Science and Engineering, National University of Defense Technology, Changsha, China

Correspondence should be addressed to Li Yu; yuli@nudt.edu.cn

Received 14 March 2017; Accepted 13 September 2017; Published 19 December 2017

Academic Editor: Haiyu Huang

Copyright © 2017 Shaobing Huang et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Adaptive beamforming is a powerful technique for anti-interference, where searching and tracking optimal solutions are a great challenge. In this paper, a partial Particle Swarm Optimization (PSO) algorithm is proposed to track the optimal solution of an adaptive beamformer due to its great global searching character. Also, due to its naturally parallel searching capabilities, a novel Field Programmable Gate Arrays (FPGA) pipeline architecture using polyphase filter bank structure is designed. In order to perform computations with large dynamic range and high precision, the proposed implementation algorithm uses an efficient user-defined floating-point arithmetic. In addition, a polyphase architecture is proposed to achieve full pipeline implementation. In the case of PSO with large population, the polyphase architecture can significantly save hardware resources while achieving high performance. Finally, the simulation results are presented by cosimulation with ModelSim and SIMULINK.

1. Introduction

Potential interference has been the major concern for system designers in military and critical civilian wireless communication since it may obscure the original received signal. As we all know, the traditional filters process signals in frequency domain, which are usually incapable of interference cancellation in cases when the interference signals occupy the same frequency band as the desired signal. In this case, if we attempt to suppress high-power interferences, the low-power signals of interest will be eliminated. Adaptive beamforming [1], known as a spatial filtering method, has been a powerful technique to enhance signals of interest while suppressing the interference and the noise signal as a result of the linear combination of the array antenna. Most of the adaptive beamforming algorithms, according to whether the training sequence is used or not, could be divided into two classes [2]: blind adaptive algorithm and nonblind adaptive algorithm. And, in our research, the nonblind algorithms are employed.

LMS [3] approaches may have been the most widely used nonblind adaptive beamforming algorithm in engineering applications due to its robustness and simplicity. However, it exhibits a slow convergence and easily tracks into local optimal solution, which would be a fatal flaw when the digital

wireless communication system has a high-performance requirement of real-time implementation.

Particle Swarm Optimization (PSO), which was proposed by Professors Eberhart and Kennedy in 1995 [4], is now one of the most important and widely used swarm intelligence algorithms. Using some simple principles, the PSO algorithms mimic the behavior of birds flocking to guide the swarm particles to search for global optimal solution. Compared to other evolutionary algorithms such as the genetic algorithms [5], simulated annealing algorithms [6], ant colony algorithms [7], and others, the PSO algorithms is much easier to implement and shows great performance in convergence speed and in searching global optimal solutions. Therefore, it has been successfully used in many engineering applications in recent years, including adaptive filters, which can be regarded as real-world optimization problems [8–13].

Similar to other iterative evolutionary computation approaches, the PSO algorithm is also a population-based optimization technique, the main drawback of which is long execution times, specifically when solving large scale complex engineering problems. Therefore, with the advantage of naturally parallel searching capabilities, parallel implementation of the PSO algorithms has been proposed to overcome the problems mentioned above, achieving high performance in

comparison with software solutions [14–17]. However, the PSO algorithm's hardware cost will increase rapidly when its population enlarges, since every increase in swarm size will result in a linear increase in the consumption of hardware resources. This weakness has restricted the use of the PSO algorithm in wide applications of digital signal processing methods.

Recently, advances in Very Large Scale Integration (VLSI) technology have seen significant interest in using Field Programmable Gate Array (FPGA) to speed up scientific and engineering computation with its parallel implementation and configurable hardware technology [18–20]. Taking advantage of powerful designed architecture, such as pipelining and parallel computing, FPGA could achieve much greater processing speed than common software solutions.

FPGA implementation of the PSO algorithm is a feasible and cheap solution because of its parallel high-performance computing and configurable character. Several different parallel architectures have been proposed to implement the PSO algorithm. Most of the previous work dealing with the implementation of the PSO algorithms based on FPGA uses fixed-point arithmetic since the conventional FPGA technology just provides integer and fixed-point arithmetic [21–25]. This approach could reduce the hardware cost in the logic area; however, the simplification is likely to result in resolution degradation because of its small dynamic range. A simple implementation of adaptive filters with the PSO algorithm based on FPGA has been presented in literature [23]. In the anti-interference communication field, especially in military wireless communication, the narrow interference signal's power is usually more than 30 dB higher than the signal of interest which requires a large dynamic range; namely, the algorithm operates over small and large numbers during the PSO execution. In addition, the iterative PSO algorithm needs high precision to offset the effect of update error. Obviously, fixed-point arithmetic could not satisfy these two requirements. Hence, we propose the adaptive beamforming algorithm with PSO using the user-defined floating-point arithmetic which would reduce the loss of precision while decreasing the consumption of hardware resources as much as possible. Although few previous works [18, 26] have implemented the PSO based on floating-point arithmetic, they are still presented using common parallel architectures in which each particle has to use independent hardware units to achieve signal processing. This results in a large consumption of hardware resources and power, which is an adverse issue for digital communication systems.

In this paper, we present a novel pipelined architecture based on FPGA to implement an adaptive beamforming algorithm using PSO based on the minimum mean square error (MSE) criterion. The proposed architecture is based on user-defined floating-point arithmetic [8]. This implementation architecture mainly applies to modern digital anti-interference communication systems in which the baseband chip cycle is much greater than the system clock period. As a consequence, a large time redundancy is generated, of which full use could be made. Essentially, this novel architecture reuses hardware resources meaning that all particles share the same hardware units to evaluate fitness

and update position. This hardly makes any difference in achieving high performance of the system because of the large fixed time redundancy. Using digital polyphase filtering signal processing technology could save a large amount of hardware resources and power consumption since essentially only one hardware processing unit i is needed for one particle. In addition, the existing floating-point arithmetic on FPGA designed by XILINX executes a formatting operation after finishing every addition or multiplication operation which would no doubt increase the consumption of resources. Further, the existing floating-point arithmetic uses the IEEE-754 standard, which may not be enough to achieve large dynamic range and high precision. For the two reasons given above, the implementations of adaptive beamforming with the PSO algorithm are based on suitable user-defined floating-point arithmetic.

The remainder of this paper is organized as follows. The model of adaptive beamforming and the PSO algorithm is presented in Section 2. Section 3 describes the related operations covering FPGA implementation of adaptive beamforming with the PSO algorithm. Section 4 provides the entire proposed implementation architectures. The simulation methods and results are given in Section 5. Finally, we present our conclusions in Section 6.

2. Adaptive Beamforming

In a real digital anti-interference communication system, an adaptive beamformer only processes baseband signals rather than the RF (Radio Frequency) signals or IF (Intermediate Frequency) signals. Figure 1 shows the entire simplified adaptive beamforming system based on the Uniform Linear Array (ULA) with N isotropic antennas. The output of the ULA $x(t)$ is given by [10]

$$X(t) = S(t) a(\theta_d) + \sum_{i=1}^L S_i(t) a(\theta_i) + n(t), \quad (1)$$

where $S(t)$ denotes the signal of interest with the Direction of Arrival (DOA) θ_d and $S_i(t)$ denotes the interference signals with the DOA θ_i . $a(\theta_d)$ and $a(\theta_i)$ denote the steering vectors for the signal of interest and interfering signals, respectively. $n(t)$ is the additive white Gaussian noise (AWGN). The RF signals from the ULA will be mixed with the LOF (Local Oscillator Frequency) by the local oscillator and then output the specified IF signals. Signal $\mathbf{x}(n) = [x_0(n), x_1(n), x_2(n), \dots, x_{N-1}(n)]^T$ is the output of the AD converter, as the input signal for the Digital Downconverter (DDC). The main role of DDC is to transform the IF discrete signals $\mathbf{x}(n)$ down to the complex baseband signal $\mathbf{X}(n)$ (where $\mathbf{X}(n) = [X_0, X_1, X_2, \dots, X_{N-1}]^T$), which is the input signal for the adaptive beamformer.

Figure 2 shows the working principle of the adaptive beamformer. The aim of adaptive beamforming is to use an a priori desired signal $S_d(t)$ to estimate the signal of interest from the received signal outside of the interference and noise.

As shown in Figure 2, the output of the adaptive beamformer is the linear combination of the weight vectors and the output of DDC. The criterion is to maximize the output in the

FIGURE 1: Baseband signal processing by adaptive beamformer.

FIGURE 2: Adaptive beamforming network.

direction of signal of interest and to get null in the direction of the interferences. The weight vectors are updated in each iteration by using the adaptive beamforming algorithm based on the minimum mean square error (MSE) criterion. Therefore, the adaptive beamforming problem can be described as follows: the output of the adaptive beamforming Y is the linear combination of the input signal $\mathbf{X}(n)$ with complex weight vectors $\mathbf{W}(n)$ (where $\mathbf{W}(n) = [W_0, W_1, W_2, \dots, W_{N-1}]^T$). Then the error signal $E(n)$ is minimized between the desired signal $D(n)$ and the output $Y(n)$. Finally, $E(n)$ is used to update the weight vectors $W(n)$.

As described above, a simple example using LMS based on MSE criterion as the adaptive beamforming algorithm can be expressed as [1]

$$Y(n) = \mathbf{W}^H(n) \mathbf{X}(n), \quad (2)$$

where

$$\begin{aligned} \mathbf{W}(n) &= [w_0(n), w_1(n), \dots, w_{N-1}(n)]^T, \\ \mathbf{X}(n) &= [x_0(n), x_1(n), \dots, x_{N-1}(n)]^T \end{aligned} \quad (3)$$

where H denotes Hermitian transpose and T denotes transpose. The error signal $E(n)$ is given by

$$E(n) = D(n) - Y(n). \quad (4)$$

And the weight vectors updated equation is presented in the following:

$$\mathbf{W}(n+1) = \mathbf{W}(n) + \mu \mathbf{X}(n) E^*(n), \quad (5)$$

where parameter μ is the correlation of the power spectrum of the input signal, representing the step size which controls the convergence speed.

3. Adaptive Beamforming Based on PSO Algorithm

In this section, an adaptive beamforming algorithm using PSO based on MSE criterion is proposed. Searching the optimal solution for adaptive beamforming can be regarded

FIGURE 3: The flowchart of the algorithm.

as a Multiobject Optimization Problem (MOP). And the MOP can be described in the following formula [26, 27]:

$$\begin{aligned} V\text{-min} \quad & f(t) = [f_1(t), f_2(t), \dots, f_{N-1}(t)]^T \\ \text{s.t.} \quad & t \in X, X \in R^m, \end{aligned} \quad (6)$$

where t is the feasible solution and $V\text{-min}$ means the minimization of the functions group. As for the adaptive beamformer, the object is to search for the optimal weight vectors using the given input signals and the desired signals. The weight vectors $W(n)$ can be regarded as a set of the functions $f_i(t)$. Therefore, the criterion is described as follows:

$$\min J = \min |E_p(n)|^2, \quad (7)$$

where J is the fitness function of PSO and $p = 0, 1, \dots, N-1$. Considering formulas (2) and (4), (7) is rewritten as

$$\min J = \min |D(n) - W_p^H(n) X(n)|^2, \quad (8)$$

where $W(n)$ means the position vector in the PSO algorithm.

To solve the MOP model of the adaptive beamforming by PSO, we consider a D -dimensional problem space. The position of the i th particle is expressed as $S_i = (s_{i1}, s_{i2}, \dots, s_{iD})$, which is represented as a weight vector and the speed of the change of position of X_i is $V_i = (v_{i1}, v_{i2}, \dots, v_{iD})$.

$i = 0, 1, \dots, N-1$, where N is the population size. In each iteration k , the PSO update equation is expressed as

$$\mathbf{V}^{k+1} = \omega \mathbf{V}^k + c_1 r_1 (\mathbf{P}_{pbest}^k - \mathbf{S}^k) + c_2 r_2 (\mathbf{P}_{gbest}^k - \mathbf{S}^k) \quad (9)$$

$$\mathbf{S}^{(k+1)} = \mathbf{S}^k + \mathbf{V}^{(k+1)}, \quad (10)$$

where ω is the inertia weight and it mainly plays the role of balancing the local search and global search [14]. c_1 and c_2 represent the acceleration constants, usually both set to 2, which is easy to implement by a shift operation on FPGA. $r_1 = U[0, 1]$ and $r_2 = U[0, 1]$ are two random numbers ranging from 0 to 1 [8]. $P_{pbesti} = (p_{best1i}, p_{best2i}, \dots, p_{bestDi})$ represents the individual best position, and $P_{gbest} = (p_{gest1}, p_{gest2}, \dots, p_{gestD})$ represents the best global position in the search space.

As for the specified optimization problem by the PSO algorithm, the fitness function could be described as (8), in which parameter p is the particles' population. The flowchart of the adaptive beamforming based on the PSO algorithm is given in Figure 3.

4. Related Operation Based on Floating-Point Arithmetic

The algorithms implemented on FPGA heavily depend on the algorithmic precision. The user-defined floating-point

FIGURE 4: User-defined floating-point pseudorandom number generator.

FIGURE 5: User-defined floating-point addition operation.

arithmetic allows the designer to make appropriate use of the bit-width of the floating-point representation according to the balance of logic area consumption and the precision requirement of the algorithm implementation. As stated in (9) and (10), the related operations of the algorithm include multiplication, addition, and random number generation. For the user-defined floating-point data, the multiplication operation is easy to realize by multiplying the IP (Intellectual Property) core provided by XILINX. Therefore, our main work focuses on the pipeline addition operation and random number generation.

4.1. Floating-Point Uniform Random Number Generator. PSO is a stochastic searching algorithm, which is based on several particles randomly moving in a feasible space. In order to compute (9) and (10), where r_1 and r_2 are supposed to be set randomly, we need to use the uniform Random Number Generators (RNGs). The position and velocity of the population in PSO also require the RNGs to generate uniform random initial values. In our proposed scheme, the RNG module is built by the configurable bit-width Linear Feedback Shift Registers (LFSRs), whose input is commonly driven by the feedback XOR (exclusive OR) function of several bits of the overall shift registers. The mantissa is a period of $2^{\text{bit-width}}$. LFSRs on FPGA are operated on fixed-point data. Hence, we could define two LFSRs to generate the mantissa and exponent in floating-point format, respectively.

For the sake of simplicity of computation, all signals are power normalized; therefore, as presented in Figure 4, the signed bit of the exponent LFSR is set as 1. That is to say, the generating exponent is always a negative integer. To avoid an integer that is too small, the bit-width of LFSR's exponent is set as 4. And the bit-width of its mantissa is supposed to be configurable enough according to the requirement of precision. In this way, the algorithm avoids the fixed-point-to-float-point conversion.

4.2. User-Defined Floating-Point Pipeline Addition Operation. The floating-point addition operation consists of the sequence of mantissa and exponent operations: shift, swap, round, and format [28, 29].

The floating-point pipeline addition in our proposed implementation architectures consists mainly of two parts: the basic-adder and formatting operation shown in Figure 5, in which SHR and SHL mean shift to the right and left, respectively. The basic adder first compares the exponents of two input operands; then the bigger one is incremented as the exponent of the output sum. At the same time, according to the compared result, it swaps and shifts the mantissa of the smaller number to align the two-incoming numbers. Then the two mantissas are added and the sum is truncated by discarding the lowest bit. The formatting operation first preprocesses the exponent and mantissa of the sum of the basic adder. Then it calculates how many duplicated sign bits

FIGURE 6: Architecture of two-incoming floating-point adder.

FIGURE 7: Architecture of four-incoming floating-point adder.

there are and finally outputs the exponent by a subtraction operation and the mantissa by SHL operation according to the number of duplicated sign bits.

A conventional floating-point addition IP core, provided by XILINX, does the formatting operation after every two-incoming addition operation. However, the formatting operation consumes much more hardware resources compared to the basic adder because of the operation of calculating the duplicated sign bits.

In our proposed architecture, we use the eight-incoming floating-point adder in formula (2) and the two-incoming and four-incoming floating-point adder for others. However, the use of the formatting operation should be minimized since it consumes greater resources compared to the floating-point adder based on the standard IEEE-754. Hence the architectures of two-incoming and four-incoming floating-point adders can be implemented in the way shown in Figures 6 and 7, in which it is unnecessary to conduct formatting after every basic-adder operation; instead it conducts formatting after summing all incoming numbers. In this way, the architecture of eight-incoming floating-point adder is presented in Figure 8.

5. Pipeline Polyphase Architecture of PSO

In this section, we first explain what the time redundancy is and then discuss how to use it to achieve a novel pipeline polyphase PSO (PPPSO) architecture for adaptive beam-forming. The polyphase term is derived from polyphase filtering, a time-sharing multiplex technology which can make good use of hardware resources units while not affecting the high performance of the algorithm in our proposed

FIGURE 8: Architecture of eight-incoming floating-point adder.

FIGURE 9: System clock cycle and data chip rate.

architecture. Finally, one particle's whole hardware unit and its main parts are presented.

5.1. Time Redundancy. In modern digital communication system, AD converters sample signals very fast as a consequence of extremely high requirement of data throughput and huge hardware resources consumption if it processes the signals directly after AD converters. In fact, it is not a feasible solution since there are not enough hardware resources, and it is unnecessary as well. In general, the sampling signals will be transformed by DDC and achieve the baseband signals with a low chip rate (e.g., 500 K chip/s). However, the system clock of a 7-series XILINX FPGA can easily achieve 250 M rate which is 500 times faster than the chip rate. An example is shown in Figure 9.

As we can see in Figure 4, every baseband chip continues for 500 system clock cycles, only one of which is needed in a conventional digital signal processing (DSP) scheme based on FPGA. And this leads to a large time redundancy; that is to say, the baseband signal chip is invalid in the other 499 system clock cycles, which is no doubt an enormous waste of hardware resources. Therefore, we propose a pipeline polyphase scheme to make full use of this part of resources. To make a better illustration, we define Time Redundancy Rate (TRR) as follows:

$$TRR = \left\lfloor \frac{\text{baseband chip cycle}}{\text{system clock cycle}} \right\rfloor, \quad (11)$$

where $\lfloor \cdot \rfloor$ means rounding down to the nearest integer.

One of the most important characteristics of PSO is that all particles in the same population are independent of the optimal solution (exchanging information only by P_{gbest}). Therefore, only one hardware unit is shared by all particles to evaluate fitness value and update positions. This greatly reduces the use of hardware resources. Undoubtedly,

FIGURE 10: Whole architecture of PPPSO.

FIGURE 11: Timing diagram of critical signals.

the greater the TRR is, the larger the population of the PSO algorithm can be set, and higher performance can be achieved, theoretically.

5.2. Adaptive Beamformer with PPPSO Algorithm. The whole architecture of the adaptive beamformer based on the PPPSO algorithm is presented in Figure 10. It consists of three parts: (1) P_{pbest} and P_{gbest} updating module: the individual best and global best values update or not according to the evaluation of the fitness function value; (2) position and velocity update module: the swarm particle updates according to formulas (9) and (10); formula (3) signals storage module: it mainly makes use of Random Access Memory (RAM).

As depicted in Figure 10, the P_{pbest} and P_{gbest} updating module receives the input signals (shown as X) and the desired signals (shown as D), then calculates fitness values according to the fitness evaluation function, and finally updates the values of individual best and global best. The individual and global best, together with r_1 , r_2 , and ω , apply to the position and velocity updating module to accomplish the updating process. Finally, our proposed pipeline polyphase

architecture requires storage of all critical coefficients, including position, velocity, P_{pbest} , and P_{gbest} .

r_1 and r_2 will be generated at each system clock cycle by RNG function mentioned above. As for the inertia coefficient ω , the proposed PPPSO architecture adopts the suggestion from [14], setting it as a dynamic function of iteration index, given by

$$\omega^k = \omega_{\max} - k \times \frac{\omega_{\max} - \omega_{\min}}{K}, \quad (12)$$

where $\omega_{\max}$ and $\omega_{\min}$ represent the maximum and minimum value of ω , respectively; k is the current iteration index of the PSO algorithm; and K is the maximum iteration index when the iterative process ends.

The timing diagram of the critical signals in our proposed PPPSO architecture is presented in Figure 11. As depicted in Figure 11, a polyphase period has an N (population size of the algorithm) system clock cycle, in which the PPPSO algorithm finishes one iteration.

That is to say, each particle will independently (they only exchange searching information at the end of a polyphase

FIGURE 12: The P_{pbest} and P_{gbest} update module.

period by the global best) finish its individual best update in a system clock cycle, benefited from the pipeline polyphase signal processing technique. In a polyphase period, the input signals, desired signals, and the inertia ω remain unchanged for all particles. Take position (shown as P in Figure 11) as an example to show how the pipeline architecture works. P_{ij} represents the i th data of the j th ($1 \leq j \leq N$) phase data channel which means j th particle's position value. Therefore, in a whole polyphase period, every particle would receive the same X and D as the input of the whole architecture to finish the update process. Since it is a pipeline process, every particle in a specified phase channel could share the same hardware units to achieve its own update using the previous position's value in the same phase channel and they do not affect each other. In this way, when one polyphase period finishes, each particle will have finished searching its own individual best value and finished searching the optimal solution.

5.3. The Individual Best and Global Best Update Module. The individual best and global best update module (depicted as Figure 12) is another critical step of the whole pipeline polyphase architecture. It contains the fitness function to evaluate the fitness value of every particle's position. The individual best and global best are considered to update or not according to computed value of fitness function.

As shown in Figure 12, the fitness function value is calculated using X , D and P as incoming data. Then the individual and global best will update or not according to the new evaluated value of the fitness function. The global best just updates one time at the end of the polyphase period according to the compared result of the global best position's fitness value of the current and previous iteration. However, it must compare the corresponding evaluated value at every different specified phase channel, which is the method to update the individual best. Hence each individual best of the particles will be stored in RAM in order, as shown in Figure 12, to achieve the comparison of each particle's individual best position of the current and last iteration.

In our proposed adaptive beamformer with the PPPSO algorithm, a four-antenna simple ULA is applied. Hence, each particle has four dimensions.

The fitness function uses the MSE criterion to minimize the error value as stated in formula (8), in which W denotes the position of the particle in the PPPSO algorithm. It is

FIGURE 13: Method to calculate the complex error.

noted that the adaptive beamformer with the PPPSO is a complex-based algorithm so that all baseband signals are complex-based. And the complex error is calculated as shown in Figure 13.

5.4. Particle Update Equation. The position and velocity update module is shown in Figure 14. As stated in formulas (9) and (10), the updating process of each particle in each dimension requires five additions (or subtractions), three multiplications, and two uniform RNGs.

As depicted in Figure 14, all operations in the hardware units of position and velocity updating module work in a full-parallel pipeline way. These operation hardware units need to work together in every system clock cycle because of the pipeline requirement. However, our scheme makes all particles share just one particle updating module, which makes good use of the pipeline polyphase implementation.

6. Simulation Results and Analysis

The proposed architectures for an adaptive beamformer based on the PPPSO algorithm have been developed in hardware description language using Verilog HDL and VHDL (Very High Speed Integrated Circuits Hardware Description Language). All the architectures are synthesizable in the XILINX ISE 14.7 tool and are based on the parameterizable floating-point packages with user-defined bit-width. Our proposed architecture mainly aims to the PPPSO algorithm with a large scale population (more than 64 in size). As mentioned above, the TRR is easy to achieve 500 in XILINX

FIGURE 14: Position and velocity update module.

FIGURE 15: Cosimulation with ModelSim and MATLAB/SIMULINK.

7-series devices. Hence, the population can reach a scale of 500 in size theoretically.

Mentor Graphics ModelSim is the most conventional HDL simulator to validate the timing of signals in the whole architecture. However, it is complicated if only ModelSim is used to validate the results of the whole adaptive beamforming system. Hence, for convenience and simplification, a cosimulation technique by ModelSim and MATLAB/SIMULINK with HDL Verifier is applied to verify the simulated results. HDL verifier automates Verilog and VHDL design verification and analyzes its response, providing interfaces to link MATLAB/SIMULINK with ModelSim. In this way, we are able to compare the complete calculated results from ModelSim and the theory results from MATLAB to verify the responses. The cosimulation schematic diagram is depicted in Figure 15.

We consider a ULA with four elements for simulating the real situation. The SNR (Signal-to-Noise Ratio) is set as 1 dB and the ISR (Interference Signal Ratio) is 30 dB. The azimuth (AZ) of the signal and interference are set as 0° and 60° , respectively. The desired signals and interference signals are composed of PN sequences and sine function, respectively. The system supposes that the ULA receives signals including AWGN and horizontal narrow-band interference so that the pitch angle is 90° . These parameters are shown in Table 1.

As for the initial parameters, Huang et al. [8] suggested a fixed value of 2.0 for both acceleration coefficients. The inertia weight is set to be ranging from 0.9 to 0.4, as a linear function

TABLE 1: Simulation parameter.

N	SNR/dB	ISR/dB	Signal AZ/ $^\circ$	Interference AZ/ $^\circ$
4	1	30	0	60

TABLE 2: Synthesis results for the PPPSO with different population size.

Population size	FF	LUTs	DSP48	RAM
128	45840 (5.29%)	51948 (11.99%)	138 (3.88%)	25 (1.7%)
192	45840 (5.29%)	51976 (12.00%)	138 (3.88%)	25 (1.7%)
256	45840 (5.29%)	51985 (12.00%)	138 (3.88%)	25 (1.7%)
320	45840 (5.29%)	52009 (12.01%)	138 (3.88%)	25 (1.7%)

as stated in formula (11). The maximum and minimum of the velocity are 0.125 and -0.125 , respectively.

6.1. Synthesis Results. The synthesis results for the FPGA implementation, based on a double precision PPPSO architecture, are presented in Table 2. The hardware resource consumption is reported in terms of FF (Flip-Flops), LUTs (Look-Up Tables), DSP blocks, and RAM memory for a XILINX Virtex7 xc7vx690tffg1926. As depicted in Table 2, it can be observed that the implementation architecture requires around 5.29% FF, 12% LUTs, 3.88% DSP48, and 1.7% RAM of available resources for all different polyphase (population size) implementations. No matter how large the population size is, the cost in logic area does not increase very much (the small increment for LUTs for the signals register can be ignored). The proposed architecture is effectively implemented in the hardware when a large scale population is required. Table 3 depicts the synthesis results for architectures based on user-defined floating-point arithmetic with various bit-width when the polyphase number (population size) is 128.

TABLE 3: Synthesis results for architectures based on user-defined floating-point arithmetic with various bit-width.

Bit-width (man + exp)	LUTs	FF
32 + 8	32481 (7.50%)	31193 (3.60%)
36 + 8	36836 (8.50%)	33489 (3.87%)
40 + 8	40698 (9.39%)	35889 (4.14%)
48 + 8	46654 (10.77%)	40538 (4.68%)
52 + 12	45840 (11.99%)	51948 (5.29%)

FIGURE 16: Fitness function (MSE) performance.

The hardware resources cost of DSP48 and RAM is unchangeable because of the fixed use of multipliers and RAM. As shown in Table 3, the cost of LUTs and FFs is gradually decreased with shorter bit-widths of mantissa and exponent. Taking into consideration Table 3, designers have the option to balance the hardware unit consumption and performance of precision. We suggest that the algorithm should use much shorter bit-widths of mantissa and exponent while not doing so affects convergence of the algorithms. From our simulation results, a value of 36 for bit-width of mantissa and a value of 8 for bit-width of exponent would already satisfy the requirement for the precision.

6.2. Simulation Results. As mentioned above, it is convenient to verify the simulation results using cosimulation technology with ModelSim and SIMULINK as shown in Figure 15. All simulation results are based on user-defined floating-point arithmetic with a value of 36 for bit-width of mantissa and a value of 8 for bit-width of exponent.

Figure 16 depicts the results of the MSE performance of the PPPSO algorithm with different sizes of population (128, 196, 256, and 320, resp.). A 10-ensemble Monte Carlo Method is applied to our simulation with different initial values for

FIGURE 17: ULA amplitude pattern for PPPSO with different population size.

all swarm particles. It can be observed in Figure 16 that MSE learning curves of the PPPSO are very steep since the number of swarm particles is very large. Although the MSE learning curves shown in Figure 16 are closely convergent, the convergent speed of the algorithm with 256-size and 320-size populations is obviously greater than it is with 128-size and 192-size populations.

Figure 17 shows the amplitude pattern of the PPPSO algorithm with different population sizes by using global best position, in the situation that signals are amid interferer and AWGN with a SIR and SNR values mentioned in Table 1. The algorithm in all of these situations can achieve a great performance to null the signals from 60° direction, namely, the interferer's direction, and achieve a high gain for signals at 0° direction, namely, the interested signal's direction. In general, they all are able to achieve a wider main lobe and can null the signal at direction of the interferer while attempting to achieve maximum reception in the specified direction of desired signal.

7. Conclusions

This paper describes a pipeline polyphase PSO architecture implementation on FPGA for an adaptive beamformer, using the efficient user-defined floating-point arithmetic. The user-defined floating-point arithmetic can perform computations with a large dynamic range and suitable precision while saving hardware resources consumption for the digital anti-interference communication application. The major advantage of our proposed architecture is to allow the use of the PSO algorithm with a large scale population by polyphase signal processing technology. In order to use polyphase architectures to implement the proposed algorithm rather than a full-parallel architecture, a pipeline hardware architecture of one swarm particle's processing unit is required, in which the hardware processing unit could be shared by all other swarm particles, with the consequence of saving a large cost of logic area.

Synthesis results demonstrate that using FPGA to implement the adaptive beamformer based on the PSO algorithm is an entirely acceptable solution. Moreover, the proposed architecture allows the designers to explore the balance of precision and performance by using the user-defined floating-point arithmetic.

In order to simplify the simulation process, the cosimulation technique with ModelSim and SIMULINK is applied to validate the results of the whole adaptive beamforming system with a four-antenna ULA. The PPPSO architectures with various large scale populations are simulated. The MSE learning curve and amplitude pattern are applied to measure performance. The simulation results demonstrate that it is efficient to implement the PPPSO algorithm with large scale populations.

In the future, we intend to explore the balance for exactly suitable precision requirement and the hardware logic area. Furthermore, a complicated time-varying situation is also supposed to take more real scenario into account.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

References

- [1] S. Haykin, "Adaptive Filter Theory," in *Person Education*, pp. 83–87, Asia, 4th edition, 2002.
- [2] S. Hossain, M. T. Islam, and S. Serikawal, "Adaptive beamforming algorithms for smart antenna systems," in *Proceedings of the 2008 International Conference on Control, Automation and Systems, ICCAS 2008*, pp. 412–416, Republic of Korea, October 2008.
- [3] A. Senapati, K. Ghatak, and J. S. Roy, "A comparative study of adaptive beamforming techniques in smart antenna using LMS algorithm and its variants," in *Proceedings of the 1st International Conference on Computational Intelligence and Networks, CINE 2015*, pp. 58–62, India, January 2015.
- [4] R. C. Eberhart and J. Kennedy, "A new optimizer using particle swarm theory," in *Proceedings of the 6th International Symposium on Micro Machine and Human Science (MHS '95)*, pp. 39–43, Nagoya, Japan, October 1995.
- [5] D. Beasley, R. R. Martin, and D. R. Bull, "An overview of genetic algorithms," in *Part I. Fundamentals, University computing*, pp. 58–58, An overview of genetic algorithms, Part I. Fundamentals, 1993.
- [6] R. A. Ruttenbar, "Simulated annealing algorithms: an overview," *IEEE Circuits and Devices Magazine*, vol. 5, no. 1, pp. 19–26, 1989.
- [7] G. Bilchev and I. C. Parmee, "The ant colony metaphor for searching continuous design spaces," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics): Preface*, vol. 993, pp. 25–39, 1995.
- [8] S. Huang, L. Yu, F.-J. Han, and W. Ding, "Adaptive beamforming algorithm for interference suppression based on partition PSO," in *Proceedings of the 7th IEEE Annual Information Technology, Electronics and Mobile Communication Conference, IEEE IEM-CON 2016*, Canada, October 2016.
- [9] D. J. Krusienski and W. K. Jenkins, "A particle swarm optimization - Least mean squares algorithm for adaptive filtering," in *Proceedings of the Conference Record of the Thirty-Eighth Asilomar Conference on Signals, Systems and Computers*, pp. 241–245, November 2004.
- [10] U. Mahbub, C. Shahnaz, and S. A. Fattah, "An adaptive noise cancellation scheme using particle swarm optimization algorithm," in *Proceedings of the 2010 IEEE International Conference on Communication Control and Computing Technologies, ICC-CCT 2010*, pp. 683–686, India, October 2010.
- [11] Z. Zhao, S. Xu, S. Zheng, and J. Shang, "Cognitive radio adaptation using particle swarm optimization," *Wireless Communications and Mobile Computing*, vol. 9, no. 7, pp. 875–881, 2009.
- [12] J. J. Liang, A. K. Qin, P. N. Suganthan, and S. Baskar, "Comprehensive learning particle swarm optimizer for global optimization of multimodal functions," *IEEE Transactions on Evolutionary Computation*, vol. 10, no. 3, pp. 281–295, 2006.
- [13] Q. Qi, J. Wang, Q. Li, T. Li, and Y. Cao, "Resource orchestration for multi-Task application in home-To-home cloud," *IEEE Transactions on Consumer Electronics*, vol. 62, no. 2, pp. 191–199, 2016.
- [14] Y. Shi and R. Eberhart, "A modified particle swarm optimizer," in *Proceedings of the IEEE International Conference on Evolutionary Computation and IEEE World Congress on Computational Intelligence*, (Cat. No.98TH8360), pp. 69–73, Anchorage, Alaska, USA, May 1998.
- [15] C.-J. Lin and H.-M. Tsai, "FPGA implementation of a wavelet neural network with particle swarm optimization learning," *Mathematical and Computer Modelling*, vol. 47, no. 9–10, pp. 982–996, 2008.
- [16] S. Mehmood, S. Cagnoni, M. Mordonini, and M. Farooq, "Particle swarm optimisation as a hardware-oriented meta-heuristic for image analysis," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics): Preface*, vol. 5484, pp. 369–374, 2009.
- [17] B.-I. Koh, A. D. George, R. T. Haftka, and B. J. Fregly, "Parallel asynchronous particle swarm optimization," *International Journal for Numerical Methods in Engineering*, vol. 67, no. 4, pp. 578–595, 2006.
- [18] D. M. Muñoz, C. H. Llanos, L. Dos S. Coelho, and M. Ayala-Rincón, "Comparison between two FPGA implementations of the particle swarm optimization algorithm for high-performance embedded applications," in *Proceedings of the 2010 IEEE 5th International Conference on Bio-Inspired Computing: Theories and Applications, BIC-TA 2010*, pp. 1637–1645, China, September 2010.
- [19] N. K. Quang, N. T. Hieu, and Q. P. Ha, "FPGA-based sensorless PMSM speed control using reduced-order extended Kalman filters," *IEEE Transactions on Industrial Electronics*, vol. 61, no. 12, pp. 6574–6582, 2014.
- [20] A. Cilaro, "New techniques and tools for application-dependent testing of FPGA-based components," *IEEE Transactions on Industrial Informatics*, vol. 11, no. 1, pp. 94–103, 2015.
- [21] H. Guo, H. Chen, F. Xu, F. Wang, and G. Lu, "Implementation of EKF for vehicle velocities estimation on FPGA," *IEEE Transactions on Industrial Electronics*, vol. 60, no. 9, pp. 3823–3835, 2013.
- [22] G. Kókai, T. Christ, and H. H. Frhauf, "Using hardware-based particle swarm method for dynamic optimization of adaptive array antennas," in *Proceedings of the 1st NASA/ESA Conference on Adaptive Hardware and Systems, AHS 2006*, pp. 51–58, Turkey, June 2006.

- [23] Z. Gao, X. Zeng, J. Wang, and J. Liu, "FPGA implementation of adaptive IIR filters with particle swarm optimization algorithm," in *Proceedings of the 2008 11th IEEE Singapore International Conference on Communication Systems, ICCS 2008*, pp. 1364–1367, China, November 2008.
- [24] P. Reynolds, R. Duren, M. Trumbo, and R. Marks, "FPGA implementation of particle swarm optimization for inversion of large neural networks," in *Proceedings of the 2005 IEEE Swarm Intelligence Symposium, 2005. SIS 2005.*, pp. 389–392, Pasadena, CA, USA.
- [25] X. Cai, S. Ngah, H. Zhu, Y. Tanabe, and T. Baba, "Pipeline architecture of particle swarm optimization," in *Proceedings of the 9th IEEE/ACIS International Conference on Computer and Information Science, ICIS 2010*, pp. 3–8, Japan, August 2010.
- [26] H. Tamaki, H. Kita, and S. Kobayashi, "Multi-objective optimization by genetic algorithms: a review," in *Proceedings of the 1996 IEEE International Conference on Evolutionary Computation, ICEC'96*, pp. 517–522, May 1996.
- [27] K. Tang, Z. Li, L. Luo, and B. Liu, "Multi-strategy adaptive particle swarm optimization for numerical optimization," *Engineering Applications of Artificial Intelligence*, vol. 37, pp. 9–19, 2015.
- [28] S. R. Vangal, Y. V. Hoskote, N. Y. Borkar, and A. Alvandpour, "A 6.2-GFlops floating-point multiply-accumulator with conditional normalization," *IEEE Journal of Solid-State Circuits*, vol. 41, no. 10, pp. 2314–2322, 2006.
- [29] A. Beaumont-Smith, N. Burgess, S. Lefrere, and C. C. Lim, "Reduced latency IEEE floating-point standard adder architectures," in *Proceedings of the 14th IEEE Symposium on Computer Arithmetic, ARITH-14*, pp. 35–42, April 1999.