A Hardware Architecture of Particle Swarm Optimization

Yiqin Lu, Peikun Wang*, Jiancheng Qin
School of Electronic and Information Engineering, South China University of Technology, Guangzhou, China.

* Corresponding author. Tel.: +86-13430240395; email: paykoon45@gmail.com
Manuscript submitted February 17, 2016; accepted May 15, 2016.
doi: 10.17706/jcp.12.5.442-450

Abstract: Particle Swarm Optimization (PSO) is a useful algorithm to deal with non-linear problems such as route economic management optimization, vehicle routing optimization and so on. Several different kinds of improved PSO algorithms is provided to further increase its searching performance, which means PSO can deal with various kinds of situation through these improved algorithms. Moreover, Multi-Swarm strategy of PSO (MSPSO) is introduced to avoid premature and reach the optimal solution with less iteration time. However, software implementation of MSPSO is too time-consuming to be employed into real-time application when particles number and iterations time are huge, even on high-speed computer. Moreover, the synchronous hardware architecture of MSPSO is ineffective since it cannot achieve the maximum performance of each module during the calculation. In order to accelerate the processing speed of MSPSO, an asynchronous architecture of MSPSO based on Field-Programmable Gate Array (FPGA) is proposed in this research. The asynchronous architecture can improve the efficiency by executing the function of each module independently with maximum performance. In addition, Asynchronous Wrapper (AW) with handshaking protocol is adopted to connect core modules and peripheral modules, which can greatly enhance the stability of data exchange. The experimental results confirm that the asynchronous approach can drastically reduce the calculation time compared with synchronous approach.

Key words: Particle swarm optimization, asynchronous architecture, asynchronous wrapper.

1. Introduction

Particle Swarm Optimization (PSO), a new stochastic heuristic algorithm which originally presented by Eberhard and Kennedy in 1995 [1], can be employed into non-linear application to find an optimal solution based on swarm searching. Recently, PSO is widely used since it can approach to a better solution with less iteration times compared with other algorithms [2], [3], such as evolutionary algorithm and genetic algorithm. Meanwhile, PSO can be easily developed on software to match numerous applications. To further increase the searching performance of PSO, several kinds of improved PSO algorithms is proposed, such as quantum-behaved PSO (QPSO) [4], PSO with random time-varying inertia weight and acceleration coefficients (PSO-RTVIWAC) [5], PSO with passive congregation (PSOPC) [6] and so on. These PSO algorithms are respectively expert in local searching area and global area.

However, simple strategy of PSO algorithm brings the problem that it cannot find the optimal solution when the complexity of the searching situation is large. On the contrary, it results in the premature convergence on local optimal solution, which is worse than global optimal solution. To overcome the above drawback, a Multi-Swarm Strategy of PSO (MSPSO) is introduced [7]-[9]. With the Multi-Swarm Strategy, the premature convergence can be avoided and hence, it can figure out an excellent solution, even if it is

operating under high complexity and huge dimensions. Nevertheless, software implementation of MSPSO requires large amount of calculation time to reach the optimal solution, even if it is executed on high-performance computer.

To accelerate the processing speed, hardware implementations of MSPSO are introduced. It includes three categories, which are Hardware/Software co-design implementation [10], [11], parallel hardware implementation [12]-[14], and serial hardware implementation [15], respectively. In particular, serial architecture [15] can balance the processing speed and hardware cost, which means that it can achieve a higher speed with less hardware cost. However, it is developed based on synchronous structure. Hence, the faster modules takes much redundant time to wait slower modules for synchronization. In addition, each module executes its function sequentially, which results in low efficiency of the architecture.

In order to overcome the above drawbacks, an asynchronous structure of MSPSO is proposed in this paper. This development aims to improve the efficiency of MSPSO hardware architecture and further accelerate the processing speed. Under the asynchronous structure, each module runs at highest frequency with executing its function independently. Moreover, Asynchronous Wrapper is adopted to guarantee the stability of data exchange between core modules and peripheral modules.

The rest of the paper is organized as follow. In Section 2, the mechanism of MSPSO algorithm and previous synchronous architecture are briefly introduced. In Section 3, the details about asynchronous structure of MSPSO is introduced. In Section 4, the comparison of experimental results between previous hardware architecture and proposed hardware architecture is introduced. In Section 5, the conclusion of this research is introduced.

2. Related Work

2.1. PSO with Multi-swarm Strategy

The original PSO algorithm which presented by Eberhard and Kennedy [1] is a powerful algorithm to figure out optimal solution. PSO algorithm imitates the behavior of animal herd to approach to the optimal solution. The basic unit of swarm is particle, which represents a candidate solution of a particular application. Each particle moves around a specified area to search the optimal solution by adjusting their velocity and position. The velocity and position of each particle is influenced by two element, which including personal best solution and global best solution. The personal best solution and global best solution is updated according to the fitness value, which defined by the specified application. After several hundred or more times iteration, particles get close to optimal solution, which is denoted by global best solution. The equation of PSO is shown as Eq. (1) and Eq. (2):

$$v_{id}^{t+1} = w \cdot v_{id}^{t} + c_1 \cdot r_1 \cdot (p_{id} - x_{id}^{t}) + c_2 \cdot r_2 \cdot (p_{gd} - x_{id}^{t}) \tag{1}$$

$$x_{id}^{t+1} = x_{id}^{t} + v_{id}^{t+1} \tag{2}$$

The velocity value and position value of particle i is represented by v_{id}^{t} and x_{id}^{t}. The calculation requires t times iteration times and the complexity of the problem is denoted as d dimensions. Global best value and personal best value are p_{gd} and p_{id}, respectively. r_1 and r_2 are the random number, and acceleration coefficients of PSO algorithm are defined by ω, c_1, c_2. Several improved PSO is proposed to further improve the searching efficiency of original PSO, for instance the quantum-behaved PSO (QPSO), PSO with random time-varying inertia weight and acceleration coefficients (PSO-RTVIWAC), PSO with passive congregation (PSOPC) and so on. PSO-RTVIWAC and PSOSPC expert in searching optimal solution on local area, while QPSO has high searching efficiency on global area.

However, particles tend to converge prematurely with only one types of PSO algorithm. Multi-Swarm Strategy (MSPSO) is introduced to solve this difficulty. Under the Multi-Swarm Strategy, particle swarm is divided into several groups, and each group exploits different kinds of PSO algorithm. Therefore, several kinds of PSO algorithm can be combined in one design, which can drastically improve the flexibility and searching efficiency of PSO system.

2.2. Previous Hardware Architecture of MSPSO

To improve the processing speed of PSO, several hardware implementation of PSO is proposed, which including Hardware/Software co-design implementation [10]M [11], parallel hardware implementation [12]-[14], and serial hardware implementation [15]. The Hardware/Software co-design implementation can achieve a less development time of fitness calculation, but its speed is limited due to the low speed of soft-core and the data exchange between hard-core and soft-core. The parallel hardware implementation can achieve high speed but it requires huge hardware cost. Among these structures, the serial hardware implementation can balance the performance and hardware cost, and its structure is shown as Fig. 1. The hardware architecture includes core modules (PCM and FCM) and peripheral modules (RAM and FCM). The calculation of core modules are most time-consuming since their structure are more complex, while the calculation of peripheral modules are much simpler. PCM (Particle Calculation Module) implements Eq. (1) and Eq. (2) to figure out the velocity and position value. The fitness value is calculated by FCM (Fitness Calculation Module). RAM is utilized to store the calculation results. The Control Unit is employed to coordinate the operation of core modules and update the best position value.

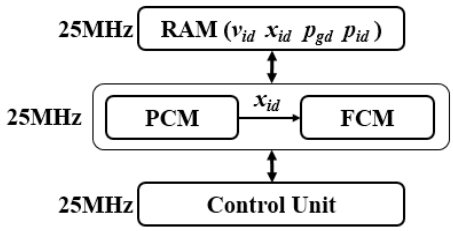


Fig. 1. Synchronous hardware structure.

The time sequence of previous hardware architecture is shown as Fig. 2. The searching procedure consists of the calculation of position and fitness through PCM and FCM, and the updating of best position value through Control Unit. This process is repeated until all iterations are completed. One dimension calculation consists of 4 steps. First, Control Unit sends commands to PCM and FCM to inform them that the calculation will start, then PCM and FCM do the calculation, finally PCM and FCM exchange data with RAM.

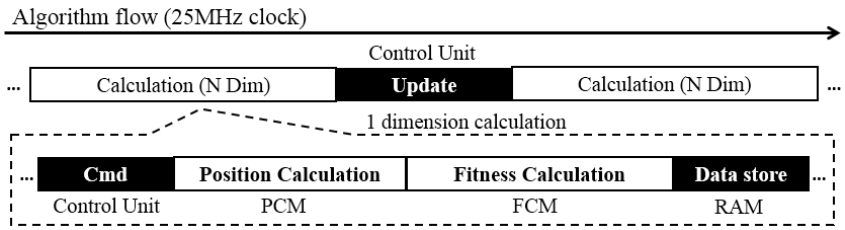


Fig. 2. Time sequence of synchronous structure.

However, the synchronous hardware architecture is ineffective since it cannot execute the maximum performance of each module. On the contrary, the slowest module decides the performance of the system. While PCM and FCM can only achieve 25MHz due to the complicated computation of position and fitness, Control Unit and RAM can operate at a much higher frequency. The system clock frequency requires to be set as 25MHz for synchronization, which means the system will be slowed down greatly. Moreover, each

module executes its function sequentially. Generally, PCM requires 4 to 5 clock periods to complete position calculation, and FCM requires more than 6 clock periods to complete fitness calculation, which is decided by the complexity of fitness function. Hence Control Unit and RAM needs to takes much time to wait the operation of PCM and FCM. The hardware structure of MSPSO should be further improved.

3. Proposed Hardware Implementation

3.1. Framework

In order to overcome the above difficulties, this paper proposes an asynchronous hardware structure which providing two distinctive features to drastically accelerate the processing speed. One is the redesigned time sequence which is aiming to make the processing more efficient. Another is the Asynchronous Wrapper (AW) which is utilized to guarantee the correctness of data transmission.

The constitution of asynchronous structure is shown as Fig. 3. Each module works independently with its highest frequency. Different with synchronous structure, the asynchronous structure adds two modules, including Updating Module and AW. The Updating Module is utilized to do best value updating, which are the task of Control Unit on previous synchronous structure. Hence, the updating process and commanding are mutually independent, which means that these two process can be operated asynchronously and concurrently to greatly reduce the waiting time. In addition, each module is connected with different clock sources, which means that each module can operate at different clock frequencies and phases. However, since the start time of data exchange cannot be decided in asynchronous structure, there will be data hazard on the data exchange. To avoid the data hazard, Asynchronous Wrapper with handshaking protocol is employed into this research, which provides a robust data transmission protocol to establish a stable communication channel.

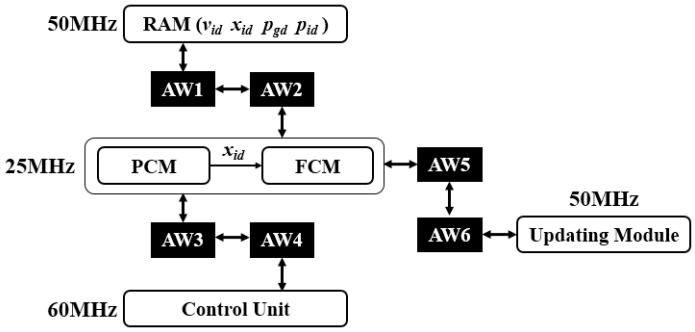


Fig. 3. Asynchronous hardware structure.

3.2. The Redesigned Time Sequence

As Fig. 4 shown, the time sequence of MSPSO is redesigned by the asynchronous structure.

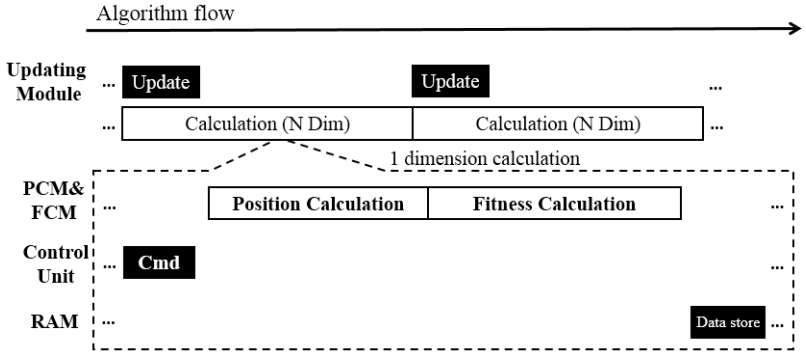


Fig. 4. Time sequence of asynchronous structure.

Two remarkable improvements is achieved. First, each module operates as fast as possible in despite of the modules which have lower performance. While the clock frequencies of PCM and FCM are hard to be increased due to the complex computation, the Updating Module, Command and RAM achieve much higher frequencies since their processes and structure are simpler. Second, instead of executing its function sequentially, each module runs in parallel at different phase. As a result, each module can continuously execute own functions without waiting other modules. Moreover, peripheral modules can fetch data from core modules once the calculation of core modules are over. Therefore, the waiting time for data exchange can be totally eliminated. The processing speed is drastically improved by the redesigned time sequence.

3.3. Asynchronous Wrapper

Since each module operates independently with different clock frequencies and phases due to the asynchronization, a structure is required to decide the start time of data exchange. In our proposed approach, Asynchronous Wrapper is applied to guarantee the stability of data exchange between different clock domains. Each module is connected with an Asynchronous Wrapper as an interface to access other modules.

Asynchronous Wrapper was originally proposed and employed into Global Asynchronous and Locally Synchronous circuit by D. S. Bormann *et al*. [16]. It is revised and simplified in this research to match PSO system. The Asynchronous Wrapper is composed of Clock Generator and Port Controller as shown in Fig. 5. The Clock Generator is used to generate clock which is applicable to corresponding module. The data exchange between different clock domain is achieved through Port Controller.

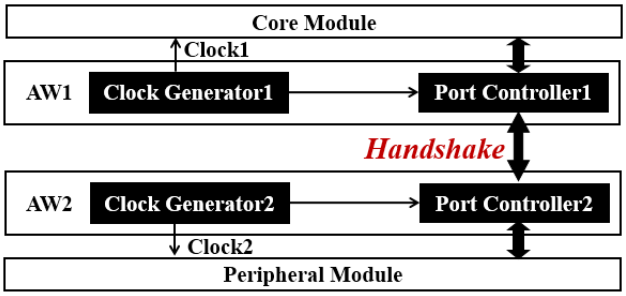


Fig. 5. The structure of Asynchronous Wrapper.

As Fig. 6 shown, Port Controller consists of Sender, Receiver and Latch. Latch is utilized to store data temporarily. To realize a robust communication, Sender and Receiver employ an Acknowledge (Ack)-Request (Req) pair, which utilizing four-phase handshaking signal to decide the start time of data transmission.

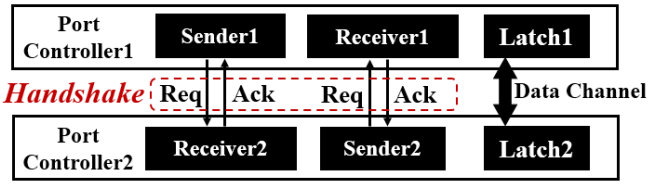


Fig. 6. The detail design of Port Controller.

As Fig. 7(a) shown, the four-phase handshaking protocol in this research is defined by a finite state machine as shown follow:

- State1: Sender sets Req as high level to start data exchange, and puts data on data channel.
- State2: Receiver detects high level on Req and receives data, then returns a high level on Ack.
- State3: Sender detects high level on Ack, then empties data channel and return a low level on Req.

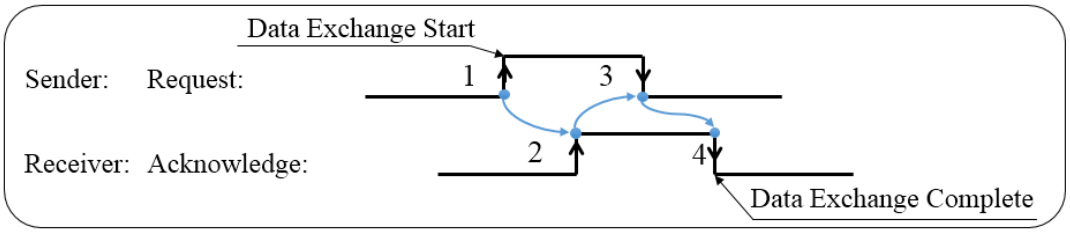
- State4: Receiver detects low level on Req, then returns a low level on Ack.

After State4, the Port Controller will be ready for next communication.

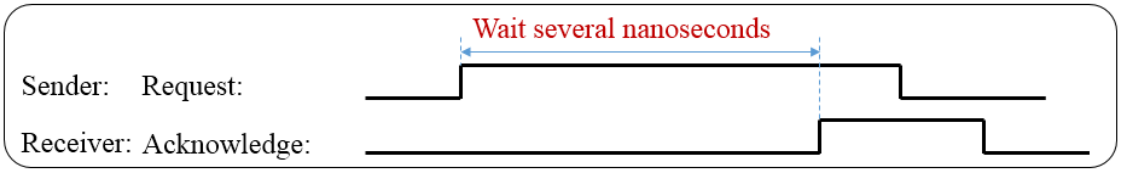
The Asynchronous Wrapper can complete the four-phase handshaking protocol within few nanoseconds since the proposed Asynchronous Wrapper is designed based on combinational circuits with small delay. The Asynchronous Wrapper optimizes the data exchange between core modules and peripheral modules as described below. First, core modules execute own function and once the calculation is completed, calculation results are temporarily stored in Asynchronous Wrapper, then transferred to the adjacent clock domain through four-phase protocol. Meanwhile, instead of waiting for the data exchange, core modules start the next calculation immediately. Since the data exchange is processed during the calculation of core modules, it does not take extra time.

However, it cannot guarantee that the four-phase handshaking protocol can be completed smoothly in one time. Once the Sender and Receiver cannot respond timely, other strategies are required to ensure the communication, as shown in Fig. 7(b), (c), (d). When Receiver do not return high level timely on State2, Sender will keep Req at high level for several nanoseconds until Receiver responds as shown in Fig. 7(b). Similarly, when there is no answer back from Receiver on State4, Sender will try to send Req several times until Receiver returns low level as shown in Fig. 7(c). Additionally, when Sender do not reply on State3, Receiver will transmit Ack several times until Sender responds as shown in Fig. 7(d). In case Sender or Receiver do not respond entirely, the calculation of relevant iteration will be restarted.

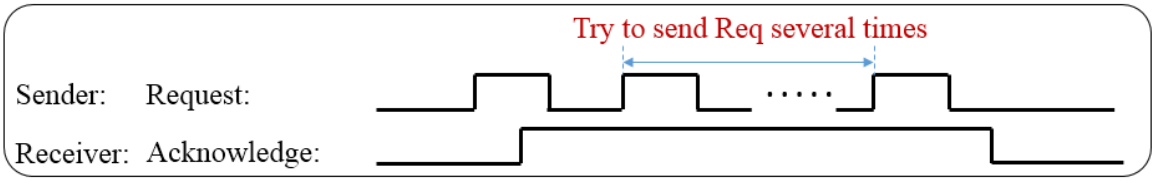
Under the control of Asynchronous Wrapper, the stability is improved greatly, and the optimal solution can be figured out correctly without data hazard.



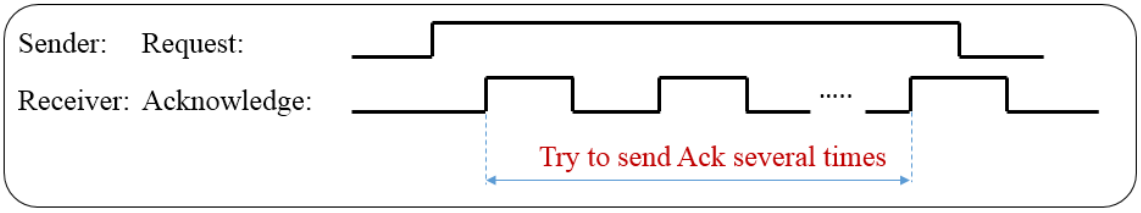
(a) Mechanism of 4-phase handshaking protocol



(b) When Receiver do not return high level



(c) When Receiver do not return low level



(d) When Sender do not return low level

Fig. 7. Four-phase handshaking protocol.

4. Experimental Results

The proposed architecture is developed based on Verilog hardware description language in QuartusII 13.0. It is implemented on the Cyclone II EP2C70F896C6 Field Programmable Gate development board, which is same to the implementation platform of previous hardware architecture. The experimental parameters are set to be same as the previous architecture as shown in Table 1. To test the performance of proposed architecture, both of the proposed and previous hardware employ Rosenbrock benchmark function as fitness function, which is shown in Table 2.

According to the experimental results shown in Table 3, the previous hardware requires 230ms to complete its calculation, while the proposed hardware needs 127ms to locate its target under the same condition. Therefore, the improve ratio of proposed hardware architecture achieves 1.81 times of improve ratio. As a trade-off, the hardware cost is increased simultaneously as Table 4 shown. The number of logic elements in proposed hardware is 5134, which is 12% more than the previous hardware.

Table 1. The Experimental Parameters

Dimension	Particle number	Iteration times
60	50	1000

Table 2. The Rosenbrock Benchmark Function

Fitness Function	Range
$f(x_i)=\sum_{i=1}^{N}(100(x_{i+1}-x_i^2)^2+(1-x_i)^2)$	[-30,30]

Table 3. The Hardware Performance Comparison

	One Particle	One Iteration	Total Calculation	Improved Ratio
Previous Hardware	4.61μs	230μs	230ms	1.81 times
Proposed Hardware	2.54μs	127μs	127ms	

Table 4. The Hardware Cost Comparison

	Logic Elements(LEs)	Improve Ratio
Previous Hardware	4572	-12%
Proposed Hardware	5134	

5. Conclusion

This paper proposed an asynchronous architecture for hardware implementation of Multi-Swarm PSO to improve the efficiency and further accelerate the processing speed. The proposed architecture provides two remarkable features. First, the time sequence is redesigned to make each module operates independently. Hence, each module can achieve maximum performance, which leads to the drastic performance improvement of the system. In addition, Asynchronous Wrapper is employed as an interface to connect different clock domains and guarantee the stability of the system. The experimental results confirm that the improved ratio of proposed hardware is 1.81 times with more 12% Logic elements, which is within an acceptable range.

Acknowledgment

This work is supported by the National Nature Science Foundation of China and fundamental Research

Funds for the Central Universities SCUT. The first author is Yiqin Lu, and thanks for his contribution in this paper.

References

[1] Kennedy, J., & Eberhart, R. C. (1995). Particle swarm optimization. *Proceedings of IEEE Int. Conf. on Neural Netw.*: *Vol. 4* (pp. 1942-1948).

[2] Shi, Y., & Eberhart, R. C. (1998). A modified particle swarm optimizer. *Proceedings of IEEE Int. Conf. Evolutionary Computation* (pp. 69-73).

[3] Parrott, D., & Li, X. (2006). Locating and tracking multiple dynamic optima by a particle swarm model using speciation. *IEEE Trans. Evol. Comput.*, *10(4)*, 440-458.

[4] Liu, J., Sun, J., & Xu, W. (2006). Improving quantum-behaved particle swarm optimization by simulated annealing. *Proceedings Computational Intelligence and Bioinformatics, Lecture Notes in Computer Science* (pp. 130-136).

[5] Zhu, H., Y., Tanabe, & Baba, T. (2008). A random time-varying particle swarm optimization for the real time location systems. *IEEJ Transactions on Electronics, Information and Systems, 128-C(12)*, 1747-1760.

[6] He, S., Wu, Q. H., Wen, J. Y., Saunders, J. R., & Paton, R. C. (2004). A particle swarm optimization with passive congregation. *BioSystems*, *78(1-3)*, 135-147.

[7] Li, X. (2004). Adaptively choosing neighborhood bests using species in a particle swarm optimizer for multimodal function optimization. *Proceedings of Conf. on Genetic Evol. Comput*. (pp. 105-116). Seattle, WA.

[8] Pulido, G. T., & Coello, C. A. (2004). Using clustering techniques to improve the performance of a particle swarm optimizer. *Proceedings of Genetic Evol. Comput. Conf.* (pp. 225-237). Seattle, WA.

[9] Chen, G., & Yu, J. (2005). Two sub-swarms particle swarm optimization algorithm. *Proceedings of Int. Conf. on Natural Comput.* (pp. 515-524). Changsha, China.

[10] Li, S.-A., Wong, C.-C., Yu, C.-J., &Hsu, C.-C. (Oct. 2010). Hardware/software co-design for particle swarm optimization algorithm. *Proceedings of IEEE Int. Conf. on Syst. Man, and Cybern* (pp. 3762-3767). Istanbul.

[11] Farmahini-Farahani, A., Vakili, S., Fakhraie, S. M., Safari, S., & Lucas, C. (March 2010). Parallel scalable hardware implementation of asynchronous discrete particle swarm optimization. *Proc. Eng. Appl. Artif. Intell., 23(2)*, 177-187.

[12] Tewolde, G. S., Hanna, D. M., & Haskell, R. E. (Mar. 30-Apr. 2, 2009). Multi-swarm parallel pso: hardware implementation. *Proceedings of IEEE Swarm Intell. Symp*. (pp. 60-66). Nashville, TN.

[13] Munoz, D. M., Llanos, C. H., Coelho, L. D. S., & Ayala-Rincon, M. (Dec. 2010). Hardware particle swarm optimization based on the attractive-repulsive scheme for embedded applications. *Proceedings of Int. Conf. on Reconfigurable Comput. and FPGAs, Quintana Roo* (pp. 55-60).

[14] Munoz, D. M., Llanos, C. H., Coelho, L. D. S., & Ayala-Rincon, M. (April 2011). Hardware particle swarm optimization with passive congregation for embedded applications. *Proceedings of 7th Southern Conf. on Programmable Logic* (pp. 173-178).

[15] Wang, P., Chen, K.-T., Fan, K., & Baba, T. (Mar. 2015). A Hardware implementation of particle swarm optimization with adaptive multi-swarm strategy. *Proceedings of RISP Int. Workshop on Nonlinear Circuits and Signal Processing* (pp. 182-185).

[16] Bormann, D. S., & Cheung, P. Y. (Oct. 1997). Asynchronous wrapper for heterogeneous systems. *Proceedings of International Conf. Computer Design*.

Yiqin Lu obtained his bachelor degree in the wireless communication technology in 1990, and his master degree in the communication and electronic system in 1993 respectively. He received his Ph.D degree in circuit and system from same university in 1996. He had studied as an exchanging Ph.D student, and then done research as post-doctor in Dept. of Computer Science in City University of Hong Kong during 1994-1998. He has been a lecturer from 1999 to 2001, and an associated professor from 2001 to 2006 in South China University of Technology (SCUT). From 2006, he has been a professor of the university. Prof. Lu is now the director of the South China Regional Network Center of China Education and Research Network (CERNET), the dean of the Office of Information Technology of SCUT, the director of Information and Network Engineering and Research Center of SCUT, the associate director of Near Distance Wireless Communication and Network Engineering and Research Center of the Ministry of Education of China, the associate director of the Key Laboratory of Guangdong Province of Wireless Communication Network and Terminal. He is an expert of the China Academic Degrees & Graduate Education Development Center (CDGDC). He is an IEEE member.

Peikun Wang is a graduate student majoring in communication and information systems in School of Electronic and Information Engineering at South China University of Technology, China. His research interests include hardware achievement of the optimization algorithms and application of FPGA. Peikun Wang received his bachelor's degree in School of Electronic and Information Engineering at South China University of Technology.

Jiancheng Qin is a candidate for doctor's degree in School of Computer Science, Beijing University of Posts and Telecommunications. In 1999, Qin graduated from School of Computer Science and Technology, Beijing University of Posts and Telecommunications, and earned the 4-year bachelor degree of computer engineering. In 2008, he earned the 2-year master degree in School of Software Engineering, Beijing University of Posts and Telecommunications. In 2011, he got the Ph.D degree in School of Computer, Beijing University of Posts and Telecommunications. His current job is in School of Electronic and Information Engineering, South China University of Technology. His major fields of study are cloud computing and intelligent terminals.