FPGA Implementation of Simulated Kalman Filter Optimization Algorithm

Nurul H. Noordin, Z. Ibrahim, M. H. J. Xie, R. Samad, N. Hasan
Faculty of Electrical and Electronic, Universiti Malaysia Pahang
26600 Pekan, Pahang Darul Makmur, Malaysia
hazlina@ump.edu.my

***Abstract*—Optimization is listed as one of the important topics in today's electronic system due to the presence of many non-linear problems in our daily life. The ability of these optimisation algorithms to perform in a real-time environment is crucial. This paper presents a novel FPGA implementation of the Simulated Kalman Filter Optimisation Algorithm. This system utilizes a distributed RAM to update the intermediate variables and the output of each iteration is stored in the block RAM. The address of the block RAM is displayed on the LCD. The hardware performance of the SKF is then compared to the PSO. Results show that the SKF has higher processing speed as well as less number of logic blocks and IO blocks were utilised.**

***Index Terms*— FPGA Design; Simulated Kalman Filter Optimization Algorithm**

I. INTRODUCTION

In solving discrete optimisation problems, algorithms such as genetic algorithm (GA) has been originally developed to operate in binary search space [1]. However, not all optimisation algorithms are originally developed to operate in a binary search space. An example of these algorithms is the Simulated Kalman Filter (SKF) optimisation, introduced by Ibrahim et al. in 2015 [2]. The SKF algorithm has improved fundamentally [5-7] and has been applied in solving engineering problems [8-9].

In recent years, Particle Swarm Optimization (PSO) attracts more attention as one of the most well-known optimization algorithms and presents its potential talent in many disciplines. Strong character inspired by the herd intelligence, PSO, which was introduced by Eberhart and Kennedy in 1995 [3], understood as a simulation of flocking animals, learn and share information when a group of insects or birds find their foods in the search space. A group of bees that are finding flowers is taken as an example. Bees fly around in their search space and constantly share individual information about the space. The sharing action among them is so-called the nature of the social behaviour. It will share its information with other bees if either one of the bees can find a better way or the position closest to the spring, which is then followed by the other members.

The SKF works quite similar to Particle Swarm Optimisation Algorithm (PSO), which had been previously introduced by Kennedy, J. and R. Eberhart [10]. Hardware implementation of PSO had been previously studied and improvements from the hardware perspective are continuously done [10-12]. In the proposed architecture, there are five modules which are Particle computing block (PCB), Fitness computing block (FCB), Random number ROM, RAM and Control unit. The initial velocity and position are randomly assigned to the particles for the case of the first iteration in the PCB unit. Velocity and position are then calculated using Equation (1) and Equation (2). The velocity and position of a particle are the output of this module for both cases.

$$v_d^{t+1} = w \cdot v_d^t + c_1 \cdot (p_d - x_d^t) + c_2 \cdot (g_d - x_d^t) \quad (1)$$

$$x_d^{t+1} = k \cdot v_d^t + x_d^t \quad (2)$$

Fitness computing block (FCB) calculates the fitness value of each particle. This fitness value specifies to each application and different from one application to another. For the position, the first module calculates the distance between the particles and each sensor using Equation (3), and then the particle fitness is calculated using Equation (4). After that, 22 pbest and gbest are updated using Equation (5) and Equation (6).

$$D_{d,m} = \sqrt{(X_m - x_d)^2 + (Y_m - y_d)^2} \quad (3)$$

$$F_i(t) = \sum_{m=0}^{3} \left(D_{i,m} - R_m\right)^2 \quad (4)$$

$$pbest \begin{cases} pbest & if\ (fitness[current] \geq fitness[pbest]) \\ current & if\ (fitness[current] \geq fitness[pbest]) \end{cases} \quad (5)$$

$$gbest \begin{cases} gbest & if\ (fitness[current] \geq fitness[gbest]) \\ current & if\ (fitness[current] \geq fitness[gbest]) \end{cases} \quad (6)$$

The calculation of the equations in the optimization algorithm may be time-consuming. Therefore, calculations for a number of random values are being done in the PCB, pre-stored in the random number ROM and recycled throughout the whole process. This feature reduces chip utilisation and time of calculations.

Finally, the control unit act as the brain of the system. The control unit controls the progress of the whole system by sending control signals and waiting for the feedback signals. The five-state state machine has been used in the designing the control unit in the previous exercise.

Previously, SKF algorithm was only simulated and evaluated at the software level. As the development of

electronics and information industry, the need for real-time high-speed processing becomes more and more sincere [13]. A particular challenge is the need for higher processing speed for complex non-linear applications. For example, in a positioning system, the urgency to calculate the position is much higher when the object is moving at a high speed. The software has to update the accurate position each time after dozens of iterations, in which it may miss the exact position of the actual current position of the target.

Many real-world nonlinear problems can be translated into cases of optimisation which are difficult to resolve with conventional optimisation algorithms that involve complex calculations. As the nonlinear problems in various fields get more and more complex, optimisation becomes one of the most important topics in computer science, engineering, management, economics, and many other fields.

Optimisation algorithm mainly acts to get the best result from the solution set by any given circumstances. Conventional optimisation algorithms perform intolerably complex calculations when meeting nonlinear optimisation. When the nonlinear problem is getting more complex, the requirement of higher processing speed for the complex nonlinear application becomes higher. The execution of SKF on software can no longer fulfil the requirements due to its long processing time. This latency is a challenge especially in a real-time application such as image processing, signal processing and positioning system. As an example, positioning system requires robust computation in order to catch up with the speed of the objects. Apart from that, onboard processing that is able to cater deep learning is required in modern application systems. Thus, this paper presents a novel FPGA implementation of the SKF algorithm. The proposed structure is implemented on Xilinx Spartan-3E and the performance of the proposed SKF FPGA is compared to the PSO FPGA and evaluated in terms of speed, cost and the accuracy.

II. Simulated Kalman Filter Algorithm

A novel estimation-based optimisation algorithm called simulated Kalman filter (SKF) has been introduced due to the inspiration by the estimation capability of Kalman filter [2]. Every agent in SKF is considered as a Kalman filter. Based on the Kalman filtering mechanisms and the measurement process, every agent estimates the global minimum or maximum. Measurement, which is required in the Kalman filter, is mathematically simulated and modelled. The positions of each agent during the search process are updated. The SKF process and its algorithm state machine are illustrated in Figure-1 and Figure-2, respectively.

III. Results and Discussions

The proposed FPGA structures, for SKF and PSO, are evaluated with the following parameters; Fitness Function of *(x-a)*, error covariance (P) of 1000, process noise (Q) of 0.5 and measurement noise (R) of 0.5. The fitness function set for both the SKF and PSO is $f(x) = (x - 25)(x = 32)$. The algorithm is simulated in Xilinx ISE 13 and the performance is then evaluated.

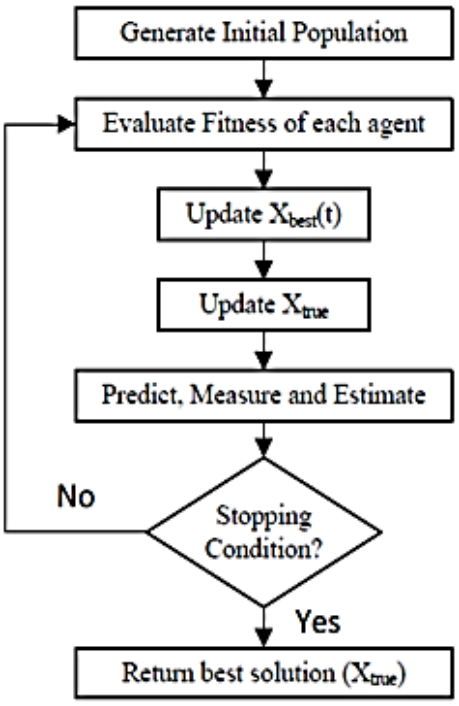

Figure 1: The Simulated Kalman Filter (SKF) flowchart.

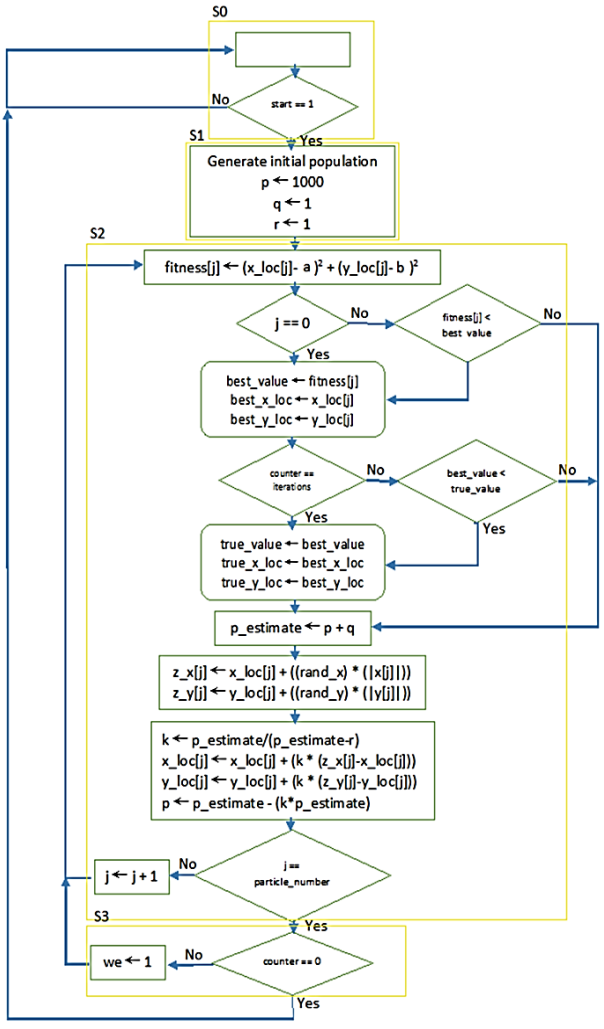

Figure 2: The Algorithmic State Machine of the Simulated Kalman Filter.

Both hardware and software implementation of SKF optimisation algorithm accurately solves the fitness function set earlier in the experiment. The duration of each clock cycle is 2 ns and the frequency used is 500 MHz. SKF converged with less number of iterations compared to PSO, as shown in Figure 3.

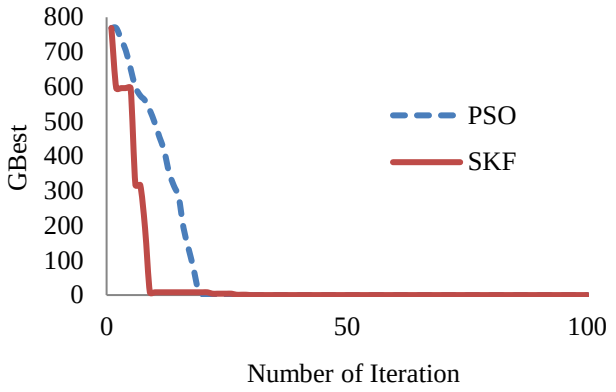

Figure 3: Convergence rate comparison between PSO and SKF.

To solve the fitness, the shortest run taken in PSO is 48 ns for 12 iterations, and the longest run took 701 iterations (2804 ns). The hardware implementation of SKF requires shorter processing time compared to the hardware PSO. For example, the PSO hardware requires nearly twice the processing time compared to the SKF hardware.

By comparing the reliability of all three approaches, the hardware implementation of PSO achieved better reliability with 95%, which is very reliable compared to both the software and hardware implementations. For hardware implemented SKF, only 65 runs among the 100 runs obtained the accurate result. Meanwhile, the hardware implemented PSO has 95 runs among 100 runs which converge to the accurate result. In terms of discrepancy, though the hardware implementation of SKF has lower reliability compared to PSO, the final output has only slight discrepant, which is 0.0990% from the accurate result as shown in Table 1. The software implementation of SKF can hardly be calculated because the maximum and minimum limits are uncertain.

Table 1
Time Stamp, Reliability and Discrepancies Comparison Between SKF and PSO

Name	Fastest run (ns)	Slowest run (ns)	Reliability (%)	Discrepancy (%)
SKF	32	48	65	0.0990
PSO	1220	2804	75	0.0072

The average convergence speed of hardware implemented SKF is much faster compared to the other three approaches as shown in Table 2. It is significant that hardware implemented approaches have higher speed compared to the software implemented approach.

Table 2
Comparison of Average Convergence Speed for All Approaches

Platform	Hardware		Software	
	SKF	PSO	SKF	PSO
Time	296.74 ns	1047.58 ns	8962.85 us	96908.02 us

IV. FPGA PERFORMANCES

The proposed structure is implemented on Xilinx Spartan 3E and evaluated in term of its performance on the FPGA.

Table3
Comparison SKF and PSO on FPGA

Properties	SKF	PSO
Shortest period	77.41 ns	63.98 ns
Max Frequency	12.92 MHz	15.63 MHz
Min input arrival	4.76 ns	4.76 ns
Max output required	4.06 ns	4.06 ns
Slices	3192 (68%)	4335 (93%)
Slice Flip Flops	710 (7%)	1270 (13%)
4 input LUTs	5870 (63%)	8106 (87%)
Used logic	5864	8097
Used Shift registers	6	9
Number of IOs	20	20
Number of bonded IOBs	20 (8%)	20 (8%)
IOB Flip Flops	3	3
Number of BRAMs	3	3
Power consumption	90.3 m	83.3 mW

SKF utilizes fewer logic devices as compared to PSO. The numbers of logic used are 5846 and 8097, respectively. However, from the perspective of power consumption, SKF consume slightly higher power compared to PSO. This is mainly due to the requirement for logic and signals in SKF FPGA implementation.

V. CONCLUSIONS

The hardware performance of the SKF algorithm has been evaluated and discussed in this paper. In the implemented structure, built-in distributed RAM, built-in block RAM and LCD were used. The processing speed of the SKF hardware is higher compared to its software implementation. Also, the hardware SKF requires lesser time to converge as compared to the hardware PSO. However, in return, hardware SKP consumes more power and achieves less reliability as compared to the hardware PSO. From the perspective of hardware, SKF requires less logic and IO blocks compared to the PSO hardware. Moving forward, the implementation of pipeline approach may result in better performance.

ACKNOWLEDGEMENT

The authors gratefully acknowledge use of the services and facilities of the Faculty of Electrical and Electronics Engineering at the Universiti Malaysia Pahang, funded by UMP Research Grant RDU 1703222.

REFERENCES

[1] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*: Addison-Wesley Longman Publishing Co., Inc., 1989.

[2] Z. Ibrahim, N. H. A. Aziz, N. A. A. Aziz, S. Razali1, M. I. S. A. Razak, S. W. Nawawi, and M. S. Mohamad, "A Kalman filter approach for solving unimodal optimization problems," *ICIC Express Letters*, vol. 9, p. 7, 2010.

[3] J. Kennedy and R. Eberhart, "Particle swarm optimization," in Neural Networks, 1995. Proceedings., IEEE International Conference on, 1995, pp. 1942-1948 vol.4.

[4] Z. Ibrahim, N. H. Abdul Aziz., N. A. Ab. Aziz, and S. Razali, M. S. Mohamad, "Simulated Kalman filter: A novel estimation-based metaheuristic optimization algorithm," *Adv. Sci. Lett.*, vol. 22, p. 2941–2946, 2016.

[5] B. Muhammad, Z. Ibrahim, K. H. Ghazali, K. Z. Mohd Azmi, N. A. Ab Aziz, N. H. Abd Aziz, and M. S. Mohamad, "A new hybrid simulated Kalman filter and particle swarm optimization for continuous numerical optimization problems," *ARPN J. Eng. Appl. Sci.*, vol. 10, no. 22, p. 17171–17176, 2015.

[6] Z. Md Yusof, I. Ibrahim, S. N. Satiman, Z. Ibrahim, N. H. Abd Aziz, and N. A. Ab Aziz, "BSKF: Binary simulated Kalman filter," *Third Int. Conf. Artif. Intell. Model. Simul.*, p. 77–81, 2015.

[7] N. H. Abdul Aziz, Z. Ibrahim, N. A. Ab Aziz, and S. Razali, "Parameter-less simulated Kalman filter," *International Journal of Software Engineering and Computer Systems*, vol. 3, pp. 129-137, 2017.

[8] K. Lazarus, N. H. Noordin, Z. Ibrahim, and K. H. Abas, "Adaptive beamforming algorithm based on simulated Kalman filter," *Asia Multi Conference on Modelling and Simulation*, p. 19-23, 2016.

[9] K. Lazarus, N. H. Noordin, Z. Ibrahim, M. F. Mat Jusof, A. A. Mohd Faudzi, N. Subari, and K. Z. Mohd Azmi, "An opposition-based simulated Kalman filter algorithm for adaptive beamforming," *IEEE Int. Conf. Appl. Syst. Inov.*, pp. 91–94, 2017.

[10] Y. Shi and R. Eberhart, "A modified particle swarm optimizer," in *1998 IEEE International Conference on Evolutionary Computation Proceedings. IEEE World Congress on Computational Intelligence (Cat. No.98TH8360)*, 1998, pp. 69-73.

[11] X. Cai, S. Ngah, H. Zhu, Y. Tanabe, and T. Baba, "Pipeline Architecture of Particle Swarm Optimization," in *2010 IEEE/ACIS 9th International Conference on Computer and Information Science*, 2010, pp. 3-8.

[12] L. Shih-An, W. Ching-Chang, Y. Chia-Jun, and H. Chen-Chien, "Hardware/software co-design for particle swarm optimization algorithm," in *2010 IEEE International Conference on Systems, Man and Cybernetics*, 2010, pp. 3762-3767.

[13] M. Jia, "Hardware implementation of particle swarm optimization and its application for adaptive signal processing," 2013.