Embedded Parallel Systolic Architecture for Multi-Filtering Techniques Using FPGA

Muataz H. Salih
School of Electric and Electronic Engineering
Underwater Robotics Research Group\USM
Penang, Malaysia
moutazsaleh@yahoo.com

M. R. Arshad
School of Electric and Electronic Engineering
Underwater Robotics Research Group\USM
Penang, Malaysia
rizal@eng.usm.my

***Abstract*— Computing systems typically suffer from delay in data processing. This delay is caused by computational power, architecture of the processor unit, synchronization signals, and so on. To enhance the performance of these systems by increasing the processing power, a new architecture and clocking technique is carried out in this paper. This new architecture design called Embedded Parallel Systolic Filters (EPSF) that can process data gathered from sensors and landmarks are proposed in our study using a high-density reconfigurable device (FPGA chip). The results show that EPSF architecture and bit-flag with a flicker clock perform significantly better in multiple input sensors signals under both continuous and interrupted conditions. Unlike the usual processing units in previous tracking and navigation systems used in robots, this system allows autonomous control of the robot through a multiple technique of filtering and processing. Furthermore, it provides fast performance and a minimal size for the entire system that minimizing the delay about 70%.**

Keywords-embedded system design; FPGA system design; systolic architecture; underwater detection

I. INTRODUCTION

Modern real time systems and applications depend on a sufficiently high processing throughput and massive data bandwidths needed in computations [1]. The need for real-time, high performance computational algorithms and architectures is one of the driving forces of modern signal processing technology and is behind the expansion of the semiconductor industry [2]. The symbiotic integration of once dissimilar memory and logic processes is very promising for processor array and memory integration on a single chip, which is the key to reliable massively parallel systems [3, 4].

Developments in the integrated circuit technology have led to a rising interest in parallel or highly concurrent algorithms and architectures [5, 6]. As a result of large available transistor counts on a single chip, different projects have emerged with the vision of integrating processor and memory on a single chip [7]. Current high-density devices (FPGA) provide the possibility for mixing memory and logic processes on the same chip. Many SoC projects and studies have been done in the past few years to show benefits of system-scale integration [8]. Most of the studies have been dealing with vector processors or small-scale MIMD processor systems. The two well-known projects are IRAM (Intelligent RAM) [9] and CRAM (Computational RAM) [10]. These projects may mark the real start of different parallel SoC systems, indeed also systolic arrays.

Our paper focuses on multi-filtering techniques by systolic arrays, where algorithm execution can be simultaneously triggered on all data elements of data set with different clock cycles. The advantage of such arrangement is that the replication of execution resources is employed. Each datum or part of a data set can be associated with a separate processing element. Processing elements form a parallel processing ensemble that is triggered by multi-clock, where each processing element operates on a different data element.

II. SYSTOLIC ARRAY

Research in the area of systolic arrays began at the end of the 1970s [11]. The original motivation behind the systolic array concept was its potential for very-large-scale integration (VLSI) implementation. Only a few systolic algorithms have been implemented in VLSI chips. The main obstacle is their limited flexibility, as general-purpose (high volume) designs are the main driving force for commercial use. The second problem is the available technology at the time of introduction. Nevertheless, several multiprocessor projects have been directly inspired by the systolic array concept such as Warp Processor developed at Carnegie-Mellon, the Saxpy Matrix-1, or the Hughes Research Labs Systolic/Cellular System. Some other projects are covered in [12, 13].

Systolic algorithms are parallel versions of sequential algorithms suitable to run on array processors. Systolic arrays are massively parallel architectures, organized as networks of identical and relatively simple processing elements that synchronously execute operations. Systolic algorithms address the performance requirements of special-purpose systems by achieving significant speedup through parallel processing and the prevention of I/O and memory bandwidth bottlenecks. Data are pumped rhythmically from the memory through the systolic array before the result is returned to the memory.

Data enter the systolic array only at the boundary. Once placed into the systolic array, data are reused many times before they become output. Several data flows move at

978-1-4244-7406-6/10/$26.00 ©2010 IEEE

constant velocities through the array and interact with each other where processing elements execute the same function repeatedly. Only the initial data and results are transferred between the host computer/global memory and the systolic array. Systolic arrays are finding their way into many practical applications ranging from radar signal processing to low-level image processing problems [12].

III. PROPOSED DESIGN

The conception of systolic or chained processing can be described as the implementation technique that partitions the execution of a given operation into the number of subsequent steps as far as possible from the same duration. Furthermore, each section assigned to a particular step can exploit standalone technical resources. Executions of single packet data requires less clock cycles and overhead can be effectively hidden by a parallel operation. Synchronous systolic is composed of stages, each of which is dedicated to a different stage of processing. Individual stages are separated by embedding additional registers. As addressed in our design using appropriate circuitry structures, systolic or chained processing might impose different types of conflicts (data and control).

To solve this problem, a flag bit is assigned for each stage indicating its status. A flag bit of 0 means the stage has finished the process and is ready to accept other data; otherwise, data coming from the previous stage will enter in a delay unit of 100 ns each cycle and will make the previous stages work on another smaller clock cycle at 50 ns until the flag bit becomes 0. This embedded architecture uses the most familiar filters for data processing in navigation and tracking systems for robots [14].

We propose the development of a homogeneous, modular, expandable systolic array combined with a large global memory. Integration of the systolic array and memory on the same piece of real state alleviates these problems. Homogeneity of the systolic array is a very important factor in the design of such a system.

A new technique of embedded parallel systolic filters (EPSF) is proposed in this paper and is depicted in Fig.'s 1 and 2. The EPSF combines the multiple Parallel Element (PE) layers of the original systolic array into a single PE layer with a set of feedback registers in a pipeline structure. Data are originally passed to stage B for pre-processing, and the flag-bit is then observed to decide whether to enter stages A or the delay unit until the flag-bit changes its status. An extra module of control signal logic circuit is required to produce the control signals for data input selection, cell memory clearance, and operation mode control.

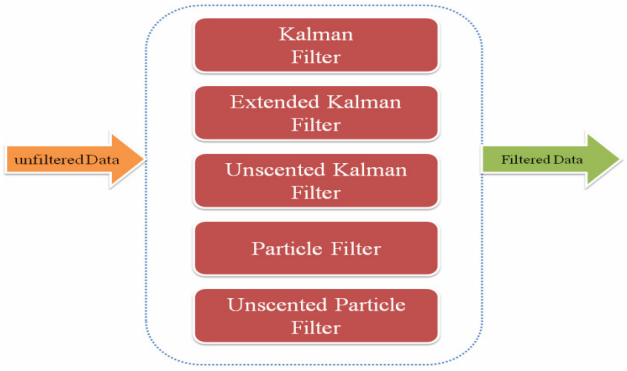

Figure 1. Embedded Parallel Filters

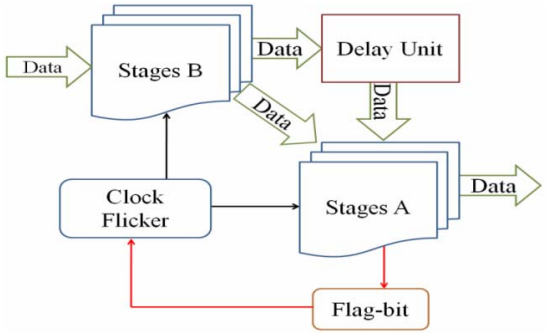

Figure 2. Filter design architecture

IV. FPGA IMPLEMENTATION

The main problem addressed in this section is the possibility of increasing the throughput of systolic arrays beyond the limit set by a systolic cycle without applying systolic algorithm transformation/mapping techniques. This can be achieved by combining embedded multithreading and systolic array computing called EPSF. Unlike classic algorithm-level transformations, multithreading can increase the throughput of the systolic array without changing the original systolic algorithm. If a certain algorithm is highly effective, the incorporation of multithreading on the pipelined systolic processing elements can improve the throughput of the same algorithm by a constant factor.

Data streams share processing element resources including inter-processing element communication data paths. It should be noted that we assume the same processing element I/O bandwidth, that is, the same bisection bandwidth as in the original single threaded systolic array. The performance increase is due to the elimination of true data hazards within each algorithm, better functional unit utilization, and larger amounts of usable instruction level parallelism uncovered within longer basic blocks constituting instruction loops. Functional unit pipelines within processing elements are kept busy by interleaving independent threads running simultaneously through the multithreaded systolic computation. A side effect of multithreading is that as the efficiency of each processing element improves, a group of algorithms can complete the execution in a shorter time than it would in the serial case.

We examined the multithreading systolic computation logically, where all threads are concurrently executed on the systolic array with the granularity of one processing element clock cycle. Implementing a multithreaded design uses a data packet transfer approach. For each iteration of the systolic program, N data elements from all N threads are input, processed, and output.

Data elements of the input data vectors of N threads are time multiplexed into a multithreaded systolic array at a rate of one element per processing element clock cycle. Data elements of the result vectors are output at the same rate. This process constitutes a multithreaded systolic cycle. It repeats for all subsequent elements and all threads. As threads are independent data sets, no data dependencies are present within the processing element pipelines. The implementation of the EPSF is performed with the memory components created inside the FPGA chip. Memory is used for frame and parameter buffers, while other circuits on the FPGA are used for pixel and parameter calculations.

Each filter contains two systolic stages for filter calculations. Input data represent data coming from the array of sound sensors (receivers) as shown in Fig. 3, becoming more sensitive and directive, enabling the system to discriminate between sounds coming from different directions.

V. PERFORMANCE COMPARISON

Testing and verification of the design were carried out by implementing the proposed technique for filters on a target reconfigurable platform based on FPGA device. This was accomplished by describing the techniques in VHDL and then synthesizing them for the FPGA chip. One motivation of this work is the evaluation of the filters' architecture in a realistic hardware environment. Therefore, we selected our chip and methodology with the goal of synthesizing the EMPSoC system and running it on DE2 board from Altera, as shown in Fig. 4.

The application is targeted for FPGA devices for many reasons. One of the goals of this paper is to design EPSF as part of EMPSoC for underwater applications such as autonomous underwater vehicle (AUV) navigation and tracking using a structural design more robust than behavioral.

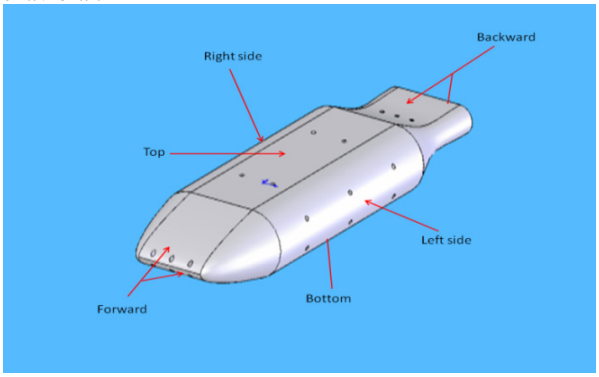

Figure 3. Sensors arrangements on AUV

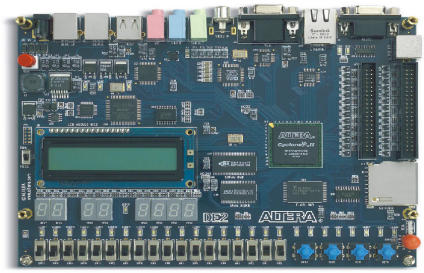

Figure 4. Altera development board (DE2)

The following graphs show the various performance results for each filter architecture. Figs. (5-9) show the total synthesis time, which is the time required by Quartus II 8.0 to compile the VHDL, perform the synthesis stage, and generate a programming file added to the time required to load the logic module flash with configuration data. Figs. (10-14) show the maximum number of cells, Nmax, which fit in the chip for each filter architecture. Figs. (15-19) show the maximum clock frequency at which the computing cells can run. This value is reported by the Quartus II 8.0 after synthesis as an estimate of the maximum clock frequency that should remain valid through acceptable device tolerances and operating conditions. Figs. (20-24) show the number of clock cycles, Ncycles, required to complete one iteration as determined by the number of states in the cell controller's FSM and by the type of operation performed in each state.

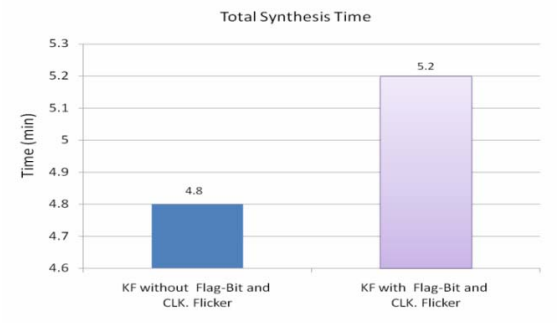

Figure 5. Total synthesis time for the KF

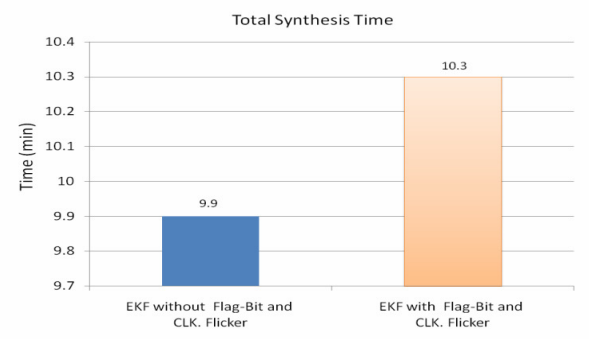

Figure 6. Total synthesis time for the EKF

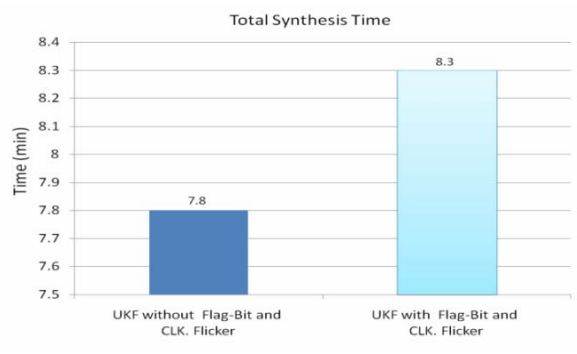

Figure 7. Total synthesis time for the UKF

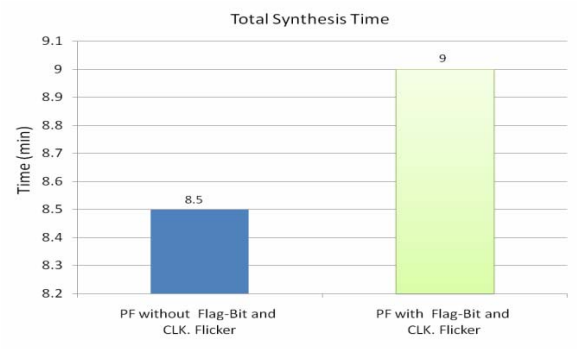

Figure 8. Total synthesis time for the PF

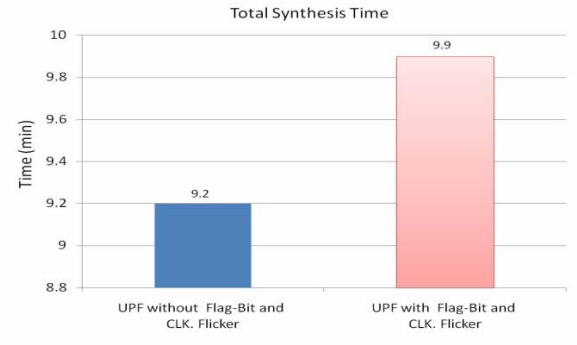

Figure 9. Total synthesis time for the UPF

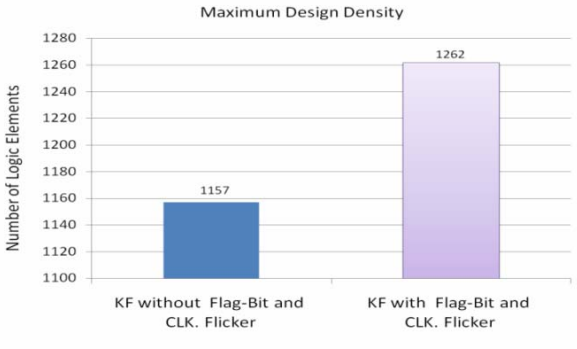

Figure 10. Maximum number of cells that fit in the FPGA for the KF

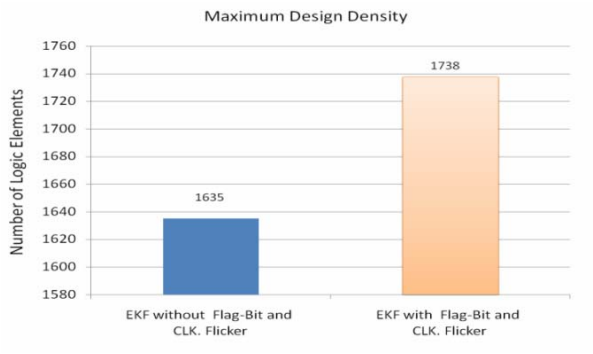

Figure 11. Maximum number of cells that fit in the FPGA for the EKF

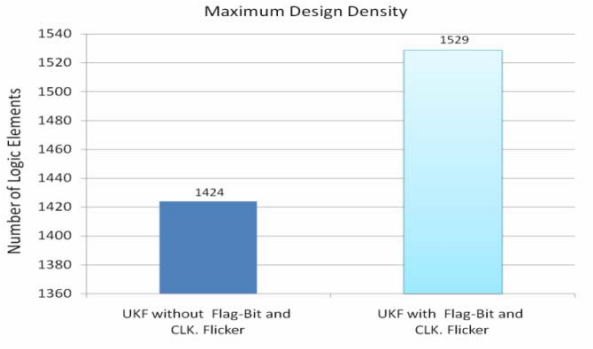

Figure 12. Maximum number of cells that fit in the FPGA for the UKF

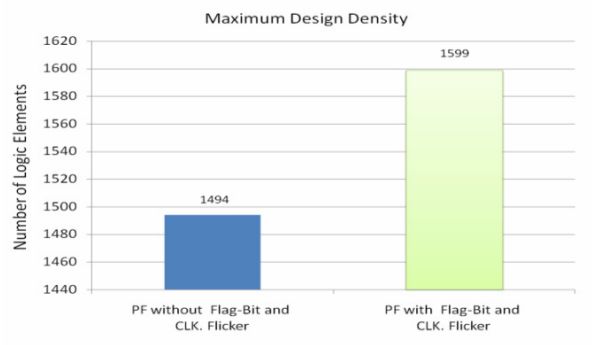

Figure 13. Maximum number of cells that fit in the FPGA for the PF

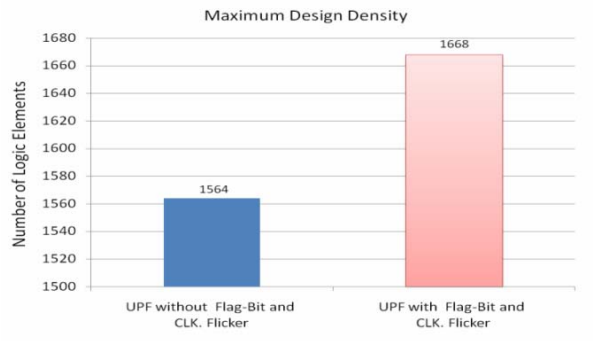

Figure 14. Maximum number of cells that fit in the FPGA for the UPF

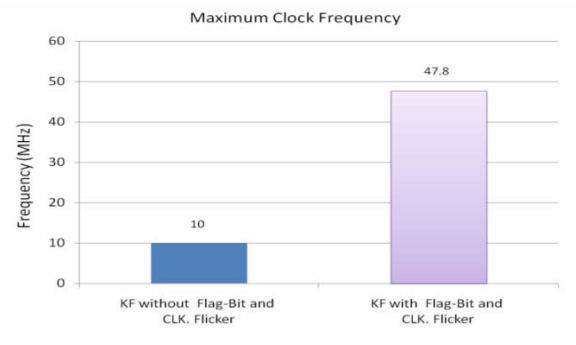

Figure 15. Maximum clock frequency for the KF

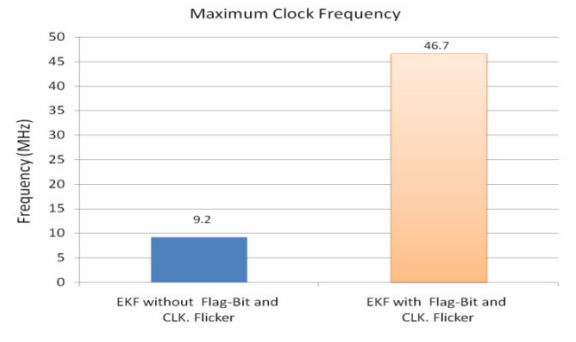

Figure 16. Maximum clock frequency for the EKF

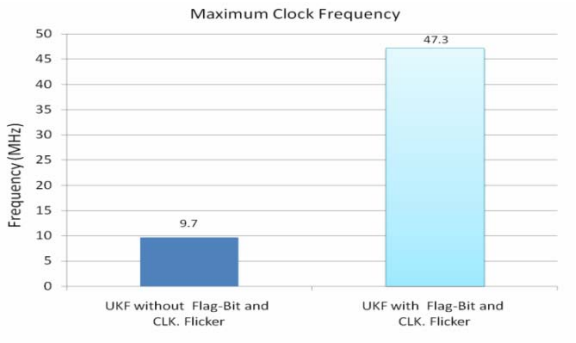

Figure 17. Maximum clock frequency for the UKF

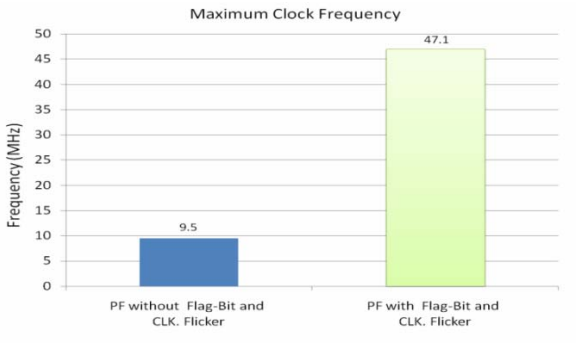

Figure 18. Maximum clock frequency for the PF

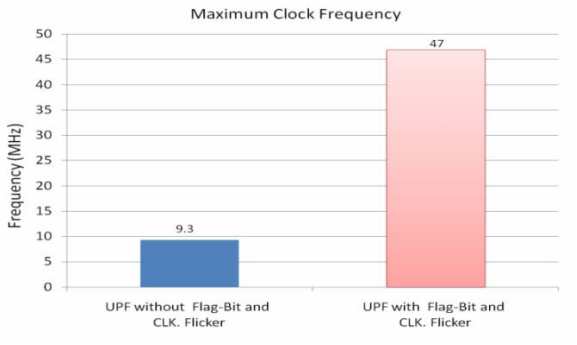

Figure 19. Maximum clock frequency for the UPF

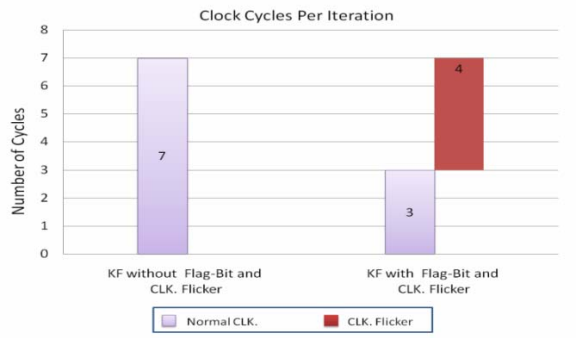

Figure 20. Number of clock cycles required for one iteration for the KF

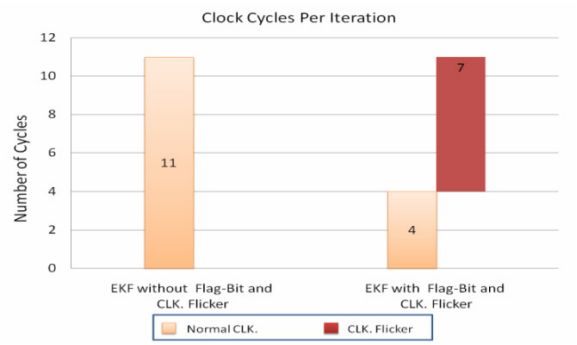

Figure 21. Number of clock cycles required for one iteration for the EKF

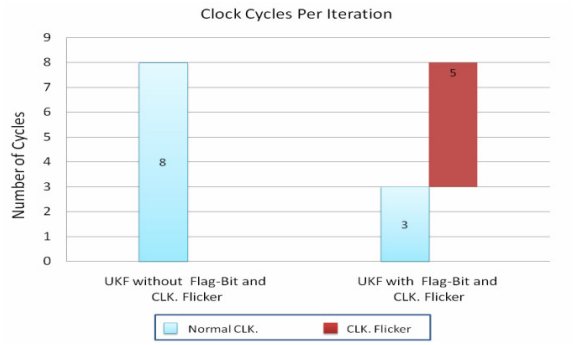

Figure 22. Number of clock cycles required for one iteration for the UKF

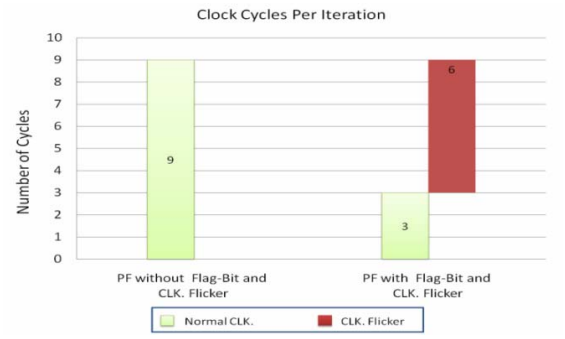

Figure 23. Number of clock cycles required for one iteration for the PF

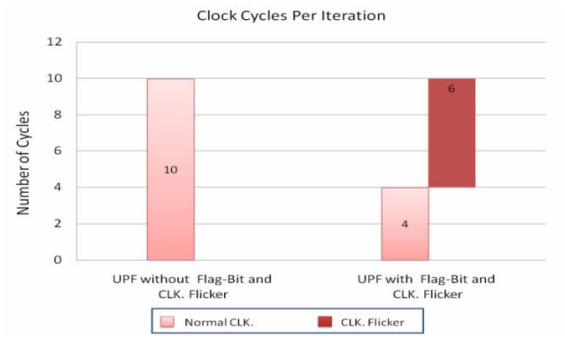

Figure 24. Number of clock cycles required for one iteration for the UPF

VI. CONCLUSION

The assuming integration of the systolic array and a global memory is presented in this paper through an SoC implementation investigation. Two novel approaches have been presented. First is the EPSF architecture, which is the solution for minimizing the delay occurring in the processing units of the system. In using this architecture, the decision maker module has a variety of information on which to build its decision. We have introduced a system-level technique that takes advantage of the clock managers present in most modem FPGAs. Flicker clock with flag-bit, which is used to reduce the processing delay and enables dynamic operation by eliminating glitches and transient and non-transient divider output errors that are very harmful when introduced into clock signals. The flicker can be included either as part of the FPGA clock managers or as a user circuit.

The results show that maximum clock frequency at which the computing cell can run with our techniques is 47.8 MHz for KF, 46.7 MHz for EKF, 47.3 MHz for UKF, 47.1 MHz for PF and 47 MHz for UPF. Whereas it is less than 10 MHz without our techniques. Finally, each filter has the same Ncycles with or without applying our techniques, but the difference that KF has 4 clk flicker from 7, EKF has 7 clk flicker from 11, UKF has 5 clk flicker from 8, PF has 6 clk flicker from 9 and UPF has 6 clk flicker from 10. As can be seen, from the results show that this technique can be used efficiently in most FPGA families.

ACKNOWLEDGMENT

The authors would like to thank the Underwater Robotics Research Group (URRG) in USM for their assistance and MOSTI for providing the research grant (grant no. 605124).

REFERENCES

[1] J. Borkenhagen, R. Eickemeyer, and R. Kalla :"A Multithreaded PowerPC Processor for Commercial Servers", IBM Journal of Research and Development, Vol. 44, No. 6, pp. 885-98, November 2000.

[2] Doug Burger and James R. Goodman. "Billion-Transistor Architectures: There and Back Again". IEEE Computer Society, Vol. 37, Issue 3, pp. 22–28, March 2004.

[3] D. Tullsen, S. Eggers, and H. Levy, "Simultaneous Multithreading: Maximizing On- Chip Parallelism," Proc. 22nd Ann. Int'l Symp. Computer Architecture, ACM Press, pp. 392-403, New York, 1995.

[4] H. Peter Hofstee. "Power Efficient Processor Architecture and the Cell Processor". In HPCA '05: Proceedings of the 11th International Symposium on High-Performance Computer Architecture, IEEE Computer Society, pp. 258– 262, Washington, DC, USA, 2005.

[5] L. Chen and Z. Hu, "Optimizing fast Fourier transform on a multi-core architecture", IEEE International Parallel and Distributed Processing Symposium, Vol.15 no.6, pp.491-504, 2007.

[6] Tarek Abdelrahman, Ahmed Abdelkhalek, Utku Aydonat, Davor Capalija, David Han, Ivan Matosevic, Kirk Stewart, Faraydon Karim, and Alain Mellan. The MLCA: A Solution Paradigm for Parallel Programmable SOCs. In IEEE North-East Workshop on Circuits and Systems (NEWCAS), pp. 253-253, June, 2006.

[7] S. Dutta, R. Jensen, and A. Rieckmann. "Viper: A multiprocessor SOC for advanced set-top box and digital TV systems". IEEE Design and Test of Computers, Vol. 18, no.5, pp. 21-31, Sep-Oct 2001.

[8] Farayadon Karim, Alain Mellan, Anh Nguyen, Utku Aydonat, and Tarek S. Abdelrahman. "A Multi-Level Computing Architecture for Embedded Multimedia Applications". IEEE Micro, Vol. 24, no. 3, pp. 55–56, 2004.

[9] D. Patterson, et al., "A case for intelligent RAM", IEEE Micro, vol. 17, no. 2, pp. 34-44, March/April 1997.

[10] D. Elliott, et.al., "Computation RAM: The case for SIMD computing in memory", 26th ISCA, 1997.

[11] H. T. Kung, C. E. Leiserson "Systolic Arrays (for VLSI)," *Technical Report CS 79-103*, Carnegie Mellon University, 1978.

[12] High Performance VLSI Signal Processing: Innovative Architectures and Algorithms, vol. I, II, Edited by: K. J. R. Liu, K. Yao, IEEE Press, 1998.

[13] Special issue on: Systolic Arrays, Computer, vol. 20, no. 7, July 1987.

[14] S. Haykin, Adaptive Filter Theory, 4th Edition, Prentice Hall, USA, 2002.