VOL. 10, NO. 17, SEPTEMBER 2015 ISSN 1819-6608
ARPN Journal of Engineering and Applied Sciences

www.arpnjournals.com

DESIGN AND IMPLEMENTATION OF A DSP ARCHITECTURE FOR WIRELESS SENSOR NODES

Ayana John and Subodh Raj M. S
Department of ECE, Sahrdaya college of Engineering, Thrissur, India
E-Mail: ayanajohn08@gmail.com

ABSTRACT

This paper proposes two different architectures to reduce power in wireless sensor nodes. Along with these two architectures, carry looks ahead adder logic and SAD Algorithms using folded tree architecture are also explained. The energy needed for the wireless communication is very high. Radio communication has highest energy consumption. Power parallel prefix technique is used in this paper to reduce the energy and power. Parallel prefix adders have the best performance in VLSI Design. The main aim of this paper is to design and implementation of newly proposed folded tree architecture. Trunk and twig phase are the two different phases of the folded tree architecture. The energy consumption can be significantly reduced by employing a more appropriate processing element. There are different types of computations in microcontrollers. Folded tree architecture is based on the on the node data processing. Measurements of the silicon implementation show an improvement of 10-20 × in terms of energy as compared to traditional modern micro controllers found in sensor nodes.

Keywords: parallel prefix, radio communication, wireless communication, folded tree.

INTRODUCTION

Environmental sensing, industrial inspection, and military surveillance are the main application of the wireless sensor networks. It consists of sensors, a radio, and a microcontroller [1]. Energy Consumption is expensive in radio communication. The communication ratio to compute energy cost range from 100 to 3000. The communication of data must be traded for on the node processing which in turn can convert the many sensor readings into a few useful data values.

Section II deals with on-the-node processing of wireless sensor nodes. Section III covers the proposed approach to exploit these properties. Section IV is on the programming and usage of the resulting folded tree architecture. Section V discusses the application specific integrated circuit (ASIC) implementation of the design. Section VI measures its performance. Section VII illustrates the usefulness to WSNs with four relevant case studies and the work is concluded in Section VIII.

CHARACTERISTICS OF WSNS AND RELATED REQUIREMENTS FOR PROCESSING

Minimize Memory Access

Modern micro controllers are based on the principles of divide and conquer strategy of ultra fast processors. In addition the lack of task specific operations leads to inefficient execution that results in longer algorithms and significant memory book keeping.

Combine Data and Control Flow Principles

Two approaches exist to manage the data stream and the instruction stream in the core functional unit. Under control flow the data stream is a consequence of the data stream. Traditional processor architecture is a control flow machine with programs that execute sequentially as a stream of instructions. The data flow program notices the data dependencies. The latter approach has been hugely successful in specialized high throughput applications mainly multimedia and graphics processing. The main characteristics of wireless sensor networks are data driven, many to few, application specific.

Many-to-Few: Radio transmissions are very expensive in terms of energy. They must be kept to a minimum in order to extend node lifetime. Data communication must be traded for on-the-node computation. The main aim is to save energy. Here sensor readings can be reduced to a few useful data values.

EXISTING METHOD

In existing method binary tree architecture is used. The disadvantages of this method is at a time only one node act as root nodes, other node act as leaves. Therefore at a time only one data is send. Then the power as well as energy is increased [3]. Here the requirement and interconnection is high range. This new proposed approach gives the limited power as well as energy. Here the time requirement is low as well as interconnection path when it is increased. So the new architecture, folded tree architecture is proposed to send the data in the way of wireless communication techniques.

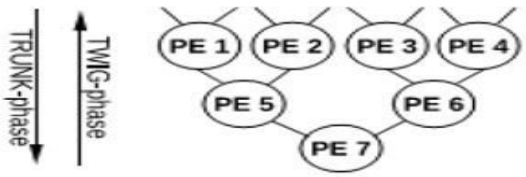


Figure-1. Binary tree architecture.

WIRELESS SENSOR NETWORKS

The application of a wireless sensor network (WSN) of spatially distributed autonomous sensors to monitor physical or environmental conditions includes temperature, sound, pressure, etc. and to cooperatively pass their data through the network to a main location. The modern networks are bi-directional, also enabling control of sensor activity [4]. The development of wireless sensor networks was motivated by military applications such as battlefield surveillance. Today such networks are used in many industrial and consumer applications include industrial process monitoring and control, machine health monitoring, and so on [5].

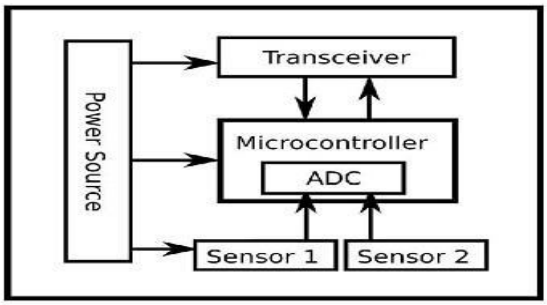


Figure-2. Block diagram of WSN.

Factors influencing sensor network design

- Fault Tolerance
- Scalability
- Production cost
- Transmission Media
- Power consumption

PROPOSED APPROACH

WSN Applications and On-The-Node Data Aggregation Notwithstanding the seemingly vast nature of WSN applications, a set of basic building blocks for on-the-node processing can be identified.

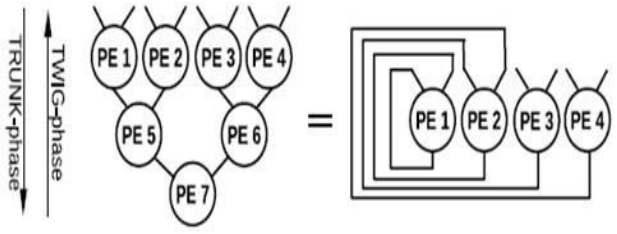


Figure-2. A binary tree (left, 7 PEs) is functionally equivalent to the novel. Folded tree topology (right, 4 PEs) used in this architecture.

Common on-the-node operations performed on input data collected directly from the node's sensors or through in-the-network aggregation include filtering, fitting, sorting, and searching. We published earlier that these types of algorithms can be expressed in terms of parallel prefix operations as a common denominator. Possible applications of sensor networks are of interest to the most diverse fields.

Parallel Prefix Operation

Prefix sums are trivial to compute in sequential models of computation, by using the formula $y_i = y_{i-1} + x_i$ to compute each output value in sequence order. However, despite their ease of computation, prefix sums are a useful primitive in certain algorithms such as counting sort and they form the basis of the scan higher-order function in functional programming languages. Prefix sums have also been much studied in parallel algorithms, both as a test problem to be solved and as a useful primitive to be used as a subroutine in other parallel algorithms. Abstractly, a prefix sum requires only a binary associative operator $\oplus$, making it useful for many applications from calculating well-separated pair decompositions of points to string processing [6].

Mathematically, the operation of taking prefix sums can be generalized from finite to infinite sequences; in that context, a prefix sum is known as a partial sum of a series [7]. Prefix summation or partial summation form linear operators on the vector spaces of finite or infinite sequences; their inverses are finite difference operators.

In functional programming terms, the prefix sum may be generalized to any binary operation (not just the addition operation); the higher order function resulting from this generalization is called a scan, and it is closely related to the fold operation. Both the scan and the fold operations apply the given binary operation to the same sequence of values, but differ in that the scan returns the whole sequence of results from the binary operation, whereas the fold returns only the final result. For instance, the sequence of factorial numbers may be generated by a scan of the natural numbers using multiplication instead of addition.

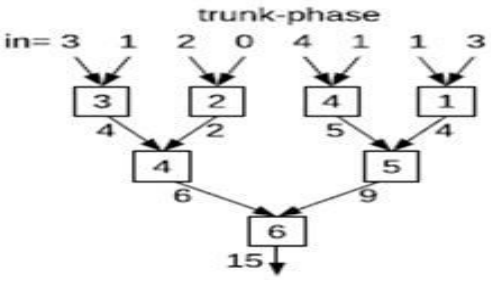


Figure-3. Parallel prefix technique.

Trunk Phase

In the trunk phase the left value L is saved locally as Lsave and it is added to the right value R, which is passed on toward the root. This continues until the parallel prefix element 15 is found at the root. Note that each time, a store and calculate operation is executed.

Twig Phase

The twig phase starts, during which data moves in the opposite direction, from the root to the leaves. Now the incoming value, beginning with the sum identity element 0 at the root, is passed to the left child, while it is also added to the previously saved Lsave and passed to the right child. In the end, the reduced prefix set is found at the leaves.

Folded Tree:

However, a straightforward binary tree implementation of Blelloch's approach as shown in Figure. costs a significant amount of area as n inputs require $p = n - 1$ PEs. To reduce area and power, pipelining can be traded for throughput. With a classic binary tree, as soon as a layer of PEs finishes processing, the results are passed on and new calculations can already recommence independently. The idea presented here is to fold the tree back onto itself to maximally reuse the PEs as shown in figure-4. In doing so, p becomes proportional to $n/2$ and the area is cut in half. Note that also interconnect is reduced. On the other hand, throughput decreases by a factor of log2 *(n) but* since the sample rate of different physical phenomena relevant for WSNs does not exceed 100 kHz, this leaves enough room for this trade-off to be made. This newly proposed folded tree topology is depicted in Figure on the right, which is functionally equivalent to the binary tree on the left.

FOLDED TREE PROGRAMMING

First the trunk phase is considered. The figure shows four processing elements. The letters L and R indicates left and right value of inputs A and B. According to Blelloch approach, L is saved as Lsave and the sum L+R is passed. To see exactly how the folded tree functionally becomes a binary tree, all nodes of the binary tree are assigned numbers that correspond to the PE, which will act like that node at that stage. As can be seen, PE1 and PE2 are only used once, PE3 is used twice and PE4 is used three times. This corresponds to a decreasing number of active PEs while progressing from stage to stage. The first stage has all four PEs active. The second stage has two active PEs: PE3 and PE4[8]. The third and last stage has only one active PE: PE4

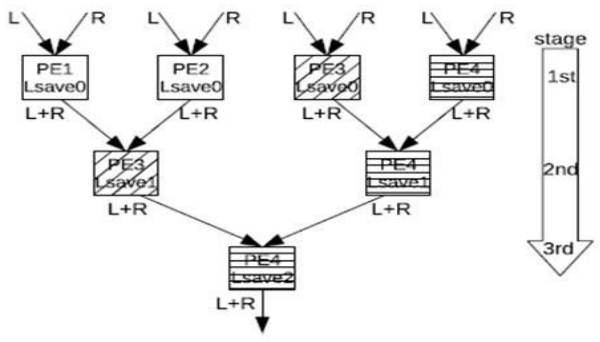


Figure-4. Trunk phase.

More importantly, it can also be seen that PE3 and PE4 have to store multiple Lsave values. PE4 must keep three: Lsave0 through Lsave2, while Pe3 keeps two: Lsave0 and L save1. PE1 and PE2 each only keep one: Lsave0. The trunk phase PE program here has three instructions, which are identical, apart from the different RF addresses that are used. Due to the fact that multiple Lsave"s have to be stored, each stage will have its own RF address to store and retrieve them. This is why PE4 needs three instructions, PE3 needs two instructions and PE1 and PE2 need one instruction.

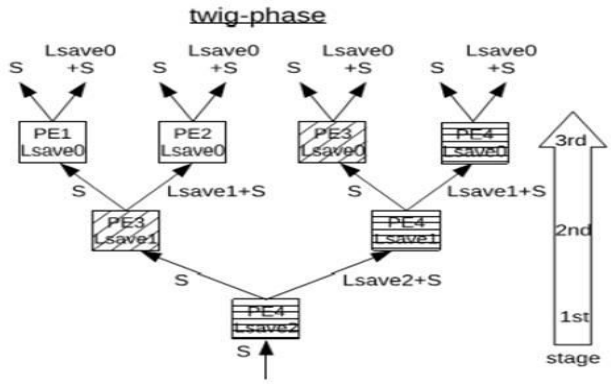


Figure-5. Twig phase.

In twig phase the tree operates in the opposite direction. According to Blelloch approach S is passed to the left and the sum S+L save is passed to the right. Note that here as well none of these annotations are global. The way the PEs are activated during the twig phase again influences how the programming of the folded tree must happen.

CARRY LOOK AHEAD ADDER LOGIC

The carry look ahead adder logic is the application for folded tree algorithm. Regardless of the number base, the basic idea of carry-look ahead logic is that at each stage i of the incrementer or adder, there are new outputs that take the place of the carry output. The carry-look ahead logic composes these new outputs along with the carry in to the entire adder to compute the carry inputs to each stage. The look ahead logic is tree structured, with the result that an n digit sum can be computed in O(log n) time, whereas with the simple ripple-carry adder or incremented, this would take O(n) time because of the need to propagate the carry through all of the digits of the number in sequence.

Each of the carry-look ahead half adders form the leaves of the tree combines carry in with one bit of the addend to produce one bit of the successor, producing a propagate signal that serves as input to the tree. The internal nodes of the tree (marked x) combine the propagate signals toward the root of the tree while taking the carry in to that subtree and computing one bit of carry that is sent back toward the leaves. At the root of the tree is a single block (marked y) that computes the carry output from the entire adder, if it is needed, from the propagate

signal for the entire number and the carry in to the entire adder.

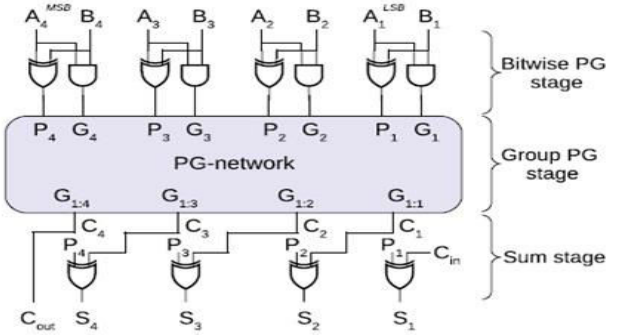


Figure-6. Addition with propagate-generate (PG) logic.

The half-adders at the leaves of the tree have two outputs, s_i, the sum output from that leaf, and p_i, the propagate output. The sum output is identical to the sum output of a conventional half adder, depending on the data in, a_i, and the carry in c_i. The propagate output is new. The propagate output from any stage of the increment circuit indicates whether that stage will propagate a carry from carry-in to that stage to carry-out from that stage. In general, the propagate signal used for incrementing is identical to the data in.

SAD ALGORITHM

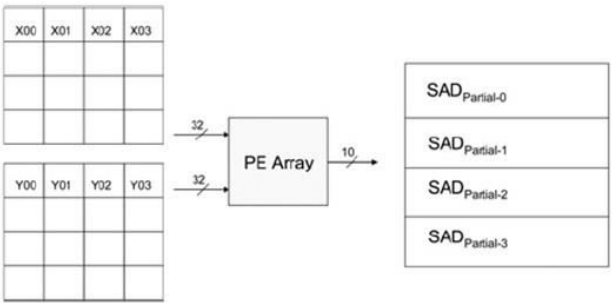


Figure-7. PE array access from buffers.

SAD is an extremely fast metric due to its simplicity; it is effectively the simplest possible metric that takes into account every pixel in a block. Therefore it is very effective for a wide motion search of many different blocks. In digital image processing, the sum of absolute differences (SAD) is a measure of the similarity between image blocks. It is calculated by taking the absolute difference between each pixel in the original block and the corresponding pixel in the block being used for comparison. These differences are summed to create a simple metric of block similarity, the L norm of the difference image or Manhattan distance between two image blocks. The sum of absolute differences may be used for a variety of purposes, such as object recognition, the generation of disparity maps for stereo images, and motion estimation for video compression [10].

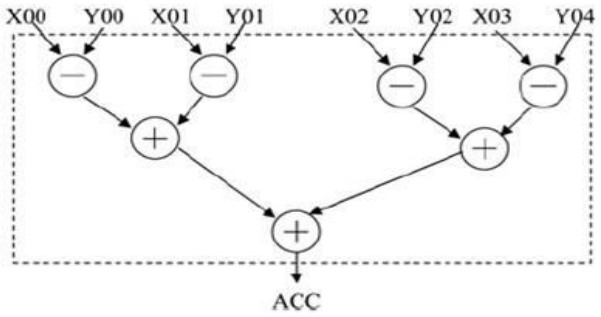
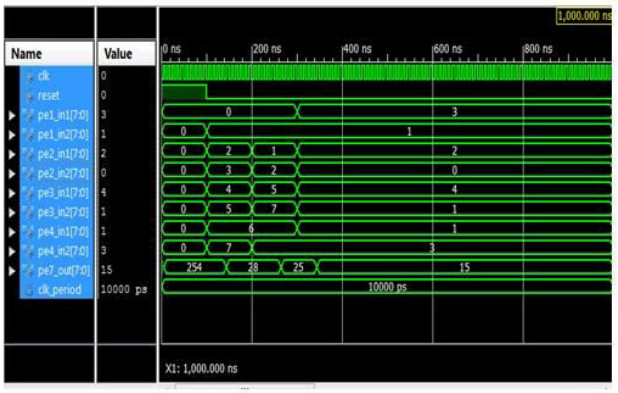


Figure-8. Computational kernel of PE processing array.

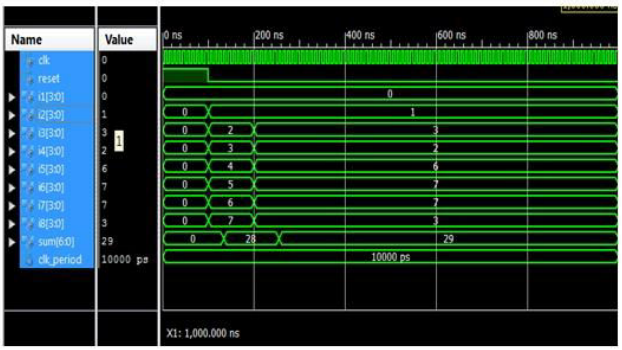
To realise a regular VLSI architecture for VBSME, the design employs the base of the primitive 4 × 4 block. The input pixels are from the temporal buffers with 32 bits. Thus, this system can read four pixels with row-by-row per cycle. The partial SAD value is accumulated with an accumulator (ACC). After four cycles, one can achieve a complete SAD for one motion vector of the 4 × 4 block. The current coding pixels X00, X01, . . . and the reference pixels Y00, Y01 are read in parallel. With 4 subtractions and 3 additions, one can obtain the partial SAD from one row per cycle. There are four partial SAD added to attain the entire SAD for one motion vector of the 4 × 4 block.

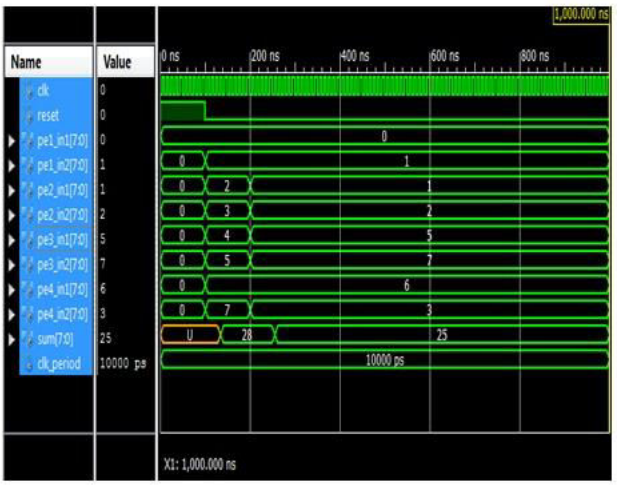
RESULTS

Results of Binary tree Architecture

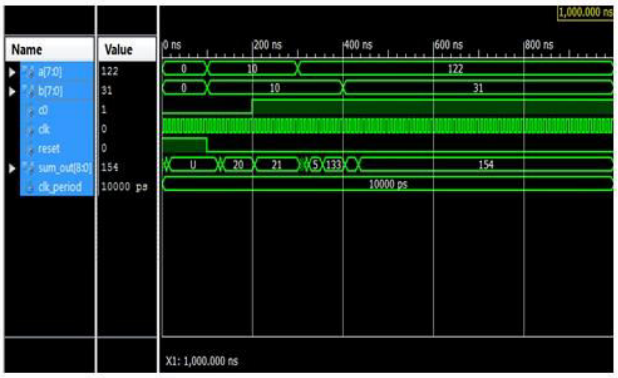


Results of Folded Tree Architecture

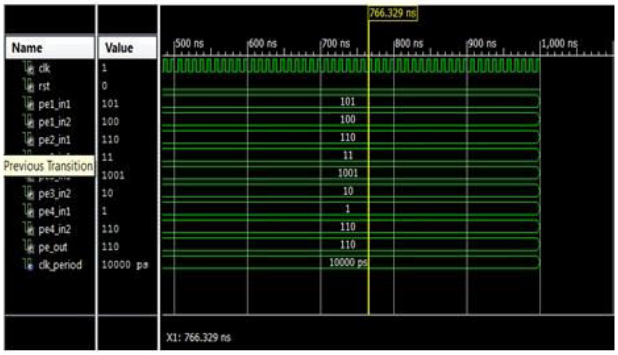


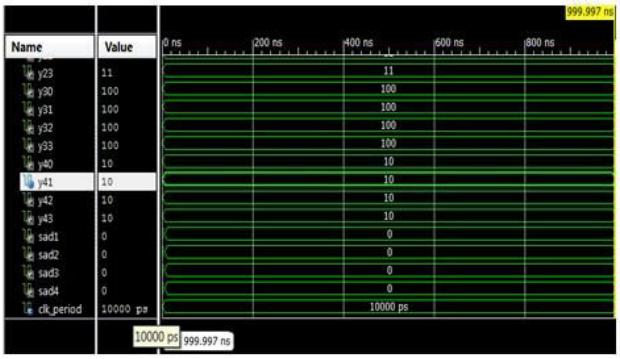


Results of carry look ahead adder logic



Results of SAD Algorithm





CONCLUSIONS

This paper describes the folded tree architecture of a digital signal processor for WSN applications. The design exploits the fact that many data processing algorithms for WSN applications can be described using parallel prefix operations, introducing the much needed flexibility energy is saved to the following conditions.

1) Limiting the data set by pre processing with parallel prefix operations.
2) The reuse of the binary tree as a folded tree

The future scope of this project is the end of architecture router is included. It is used to reduce the delay as well as congestion. Independent of illumination. Thus the modified algorithms provides an advanced and roust binarization technique for Optical character Recognition.

REFERENCES

[1] Agrawal B., Chong F. T., Mysores. S and Sherwood T. 2008. “Exploring the processor and ISA design for wireless sensor network applications”, in Proc. 21th Int. Conf. Very-Large-Scale Integer. (VLSI) Design, pp. 59–64.

[2] Backus J. 1997. “Can programming be liberated from the von neumann style?” in Proc. ACM Turing Award Lect., pp. 1–29.

[3] Ekanayake V.N., Kelly C. and R. Manohar. 2005. "Bit SNAP: Dynamic significance compression for a low energy sensor network asynchronous processor", in Proc. IEEE 11th Int. Symp. Asynchronous Circuits Syst. pp. 144–154.

[4] Ekanayake V.N., Kelly C. and Manohar R. 2004. "SNAP/LE: An ultra-low power processor for sensor networks", ACM SIGOPS Operat. Syst. Rev. - ASPLOS, Vol. 38, No. 5, pp. 27–38.

[5] Hempstead M., Lyons and G.-Y. Wei. 2008. "Survey of hardware systems for wireless sensor networks", Low Power Electron., Vol. 4, No. 1, pp. 11–29.

[6] Blelloch. 1987. "Scans as primitive parallel operations, " IEEE Trans. Comput., Vol. 38, No. 11, pp. 1526–1538.

[7] Blelloch G.E. 1990. "Prefix sums and their applications", Carnegie Mellon Univ., Pittsburgh, PA: USA, Tech. Rep. CMU-CS- 90.

[8] Hennessy J. and Patterson D. 2007. "Computer Architecture a Quantitative Approach", 4th ed. San Mateo, CA: Morgan Kaufmann.

[9] Hempstead M. Brooks D. and Wei G. 2011. "An accelerator-based Wireless sensor network processor in 130 nm cmos", Vol. 1, No. 2, pp. 193– 202.

[10] Hempstead M., Brooks W., Welsh D. 2002. "Tinybench: The case for a standardized benchmark suite for Tiny OS based wireless sensor network devices", in Proc. IEEE 29th Local Comput. Netw. Conf., Nov. 2004, pp. 585–586.