International Journal of Electrical and Electronics Engineers
Vol. No. 9, Issue No. 02, July - December 2017
ISSN (O) 2321-2055
ISSN (P) 2321-2045

IMPROVISED WIRELESS SENSORY NODES BASED ON LOW POWER DSP ARCHITECTURE

Swetha Tarigoppula[1], Chiruvella Suresh[2]

[1,2]Assistant Professor , Department of ECE,

Dhruva Institute of Engineering and Technology, Hyderabad, (India)

ABSTRACT

In 21st the role of wireless communications is huge in daily life applications but still power consumption by the applications is still concerned area infield of digital signal processing. Low power DSP architecture is required in all applications. Wireless communication exhibits the highest energy consumption in wireless sensor nodes. Given their limited energy supply from batteries or scavenging, these nodes must trade data communication for on-the-node computation. Due to the increasing complexity of VLSI circuits and their frequent use in portable applications, energy losses in the interconnections of such circuits have become significant. In the light of this, an efficient routing of these interconnections becomes important. In the implemented design describes the design and implementation of the newly proposed folded-tree architecture for on-the-node data processing in wireless sensor networks, in addition of add the routing technique for the high communication. 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 :Digital processor, Folded Tree, Modern Micro-Controller, parallel prefix, wireless sensor Network (WSN).

I. INTRODUCTION

Wireless Sensor Network (WSN) applications range from medical monitoring to environmental sensing, industrial inspection, and military surveillance. WSN nodes essentially consist of sensors, a radio, and a microcontroller combined with a limited power supply, e.g., battery or energy scavenging. Since radio transmissions are very expensive in terms of energy, they must be kept to a minimum in order toextend node lifetime. The ratio of communication-to computation energy cost can range from 100 to 3000. In addition, the lack of task-specific operations leads to inefficient execution. The data-driven nature of WSN applications requires a specific data processing approach. Previously, we have shown how parallel prefix computations can be a common denominator of many WSN data processing algorithms.

It is possible to say that history of sensor network technology originates in the first distributed sensing idea implementations. The continuous work of researchers and engineers over sensor networks which lately became wireless sensor networks (WSNs) has started exactly with this idea. Like many other technologies, distributed sensing was firstly introduced by the military. The first system which has all the characteristics of sensor networks (distribution, hierarchical data processing system) is Sound Surveillance System (SOSUS), which was made to detect and track submarines. SOSUS consisted of the acoustic sensors (hydrophones) settled on the

ocean bottom. In 1980s Defense Advanced Research Projects Agency (DARPA) is working over Distributed Sensor Networks (DSN) program.

The main task of the program was to test applicability of a new approach to machine communications, introduced for the first time in Arpanet (predecessor of the Internet). The task of researchers was to engineer a network of area-distributed sensors. At the same time, sensors had to be inexpensive, work autonomously and exchange data independently. Such demands are still made for developing sensor networks for modern applications. Hence, it is possible to say that the DARPA research was a base for modern WSNs. A sensor network of acoustic sensors tracking aircrafts appeared as a result of collaboration of researchers from Carnegie Mellon University (CMU), Pittsburgh, PA, and Massachusetts Institute of Technology (MIT), Cambridge. For a demonstration there was a platform made to passively detect and track low-flying aircraft. Connection between mobile nodes and a central computer was implemented through wireless transmission channel. Certainly, this system included not so many wireless nodes, and it was necessary to transport mobile nodes in the lorries, also system was able to track only low-flying objects with simple trajectory in rather short distance. However, this work was well in advance of that time and gave a considerable impetus to sensor networks developing.

The goal of this paper is to design an ultralow-energy WSN digital signal processor by further exploiting this and other characteristics unique to WSNs.

II. CHARACTERISTICS OF WSN

Several specific characteristics, unique to WSNs, need to be considered when designing a data processor architecture for WSNs.

2.1 Data-Driven

WSN applications are all about sensing data in an environment and translating this into useful information for the end-user, so virtually all WSN applications are characterized by local processing of the sensed data.

2.2 Many-to-Few

Since 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 to save energy, so many sensor readings can be reduced to a few useful data values.

2.3 Applications-Specific

A “one-size-fits-all” solution does not exist since a general purpose processor is far too power hungry for the sensor node’s limited energy budget. ASICs, on the other hand, are more energy efficient but lack the flexibility to facilitate many different applications. Apart from the above characteristics of WSNs, two key requirements for improving existing processing and control architectures can be identified.

2.4 Minimize Memory Access

Modern micro-controllers (MCU) are based on the principles of a divide-and-conquer strategy of ultra-fast processors on the one hand and arbitrary complex programs on the other hand. But due to this generic approach, algorithms are deemed to spend up to 40–60% of the time in accessing memory, making it a bottleneck.

2.5 Data Flow and Control Flow Principles

To manage the data stream (to/from data memory) and the instruction stream (from program memory) in the core functional unit, two approaches exist. Under control flow, the data stream is a consequence of the

International Journal of Electrical and Electronics Engineers
Vol. No. 9, Issue No. 02, July - December 2017
ISSN (O) 2321-2055
ISSN (P) 2321-2045

instruction stream, while under data flow the instruction 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. In contrast, a data flow program identifies the data dependencies, which enable the processor to more or less choose the order of execution. The latter approach has been hugely successful in specialized highthroughput applications, such as multimedia and graphics processing.

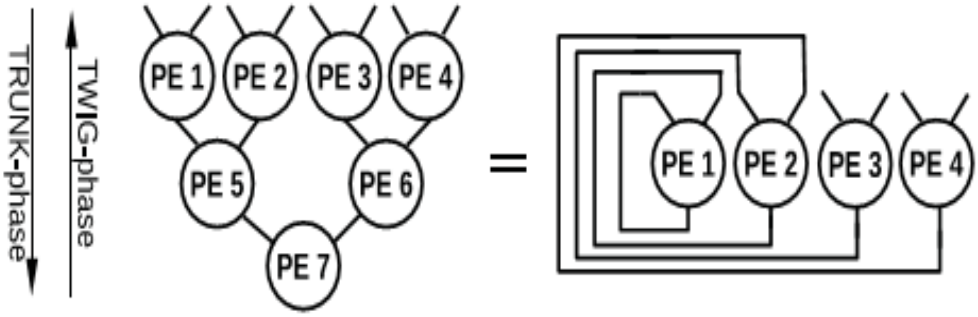

Figure 1: A binary tree (left, 7 PEs) is functionally equivalent to the novel folded tree topology (right, 4 PEs) used in this architecture

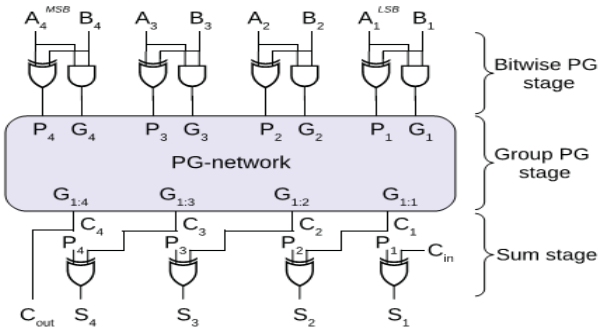

Figure 2: Addition with propagate-generate (PG) logic

III. PROPOSED METHOD

3.1 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. 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[7].Prefix operations can be calculated in a number of ways, but we chose the binary tree approach because its flow matches the desired on-the-node data aggregation. This can be visualized as a binary tree of processing elements (PEs) across which input data flows from the leaves to the root (Fig. 1, left). This topology will form the fixed part of our approach, but in order to serve multiple applications, flexibility is also required. The tree-based data flow will, therefore, be executed on a data path of programmable PEs, which provides this flexibility together with the parallel prefix concept.

3.2 Parallel Prefix Operations

In the digital design world, prefix operations are best known for their application in the class of carry look-ahead adders. The addition of two inputs A and B in this case consists of three stages (Fig. 2): a bitwise propagate

International Journal of Electrical and Electronics Engineers
Vol. No. 9, Issue No. 02, July - December 2017
ISSN (O) 2321-2055
ISSN (P) 2321-2045

generate (PG) logic stage, a group PG logic stage, and a sum-stage. The outputs of the bitwise PG stage (Pi = Ai + Bi and Gi = Ai ・ Bi) are fed as (Pi, Gi)-pairs to the group PG logic stage, which implements the following expression:

(Pi, Gi) ◦ (Pi+1, Gi+1) = (Pi ・Pi+1, Gi + Pi ・Gi+1) (1)

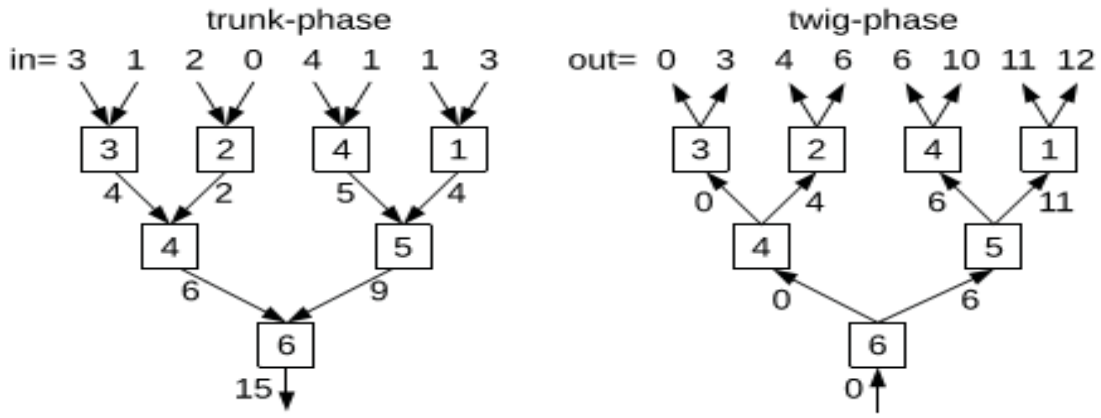

Figure 3: Example of a prefix calculation with sum operator using Blelloch's generic approach

For example, the binary numbers A = "1001" and B= "0101" are added together. The bitwise PG logic of LSBfirst noted A = {1001} and B = {1010} returns the PG-pairs for these values, namely, (P, G) = {(0, 1); (0, 0); (1,0); (1, 0)}.Using these pairs as input for the group PG-network, defined by the ◦-operator from (1) to calculate the prefix operation, results in the carry-array G = {1, 0, 0, 0} [i.e., the second element of each resulting pair from (1)]

In fact, it contains all the carries of the addition, hence the name carry look ahead. Combined with the corresponding propagate values Pi, this yields the sum S = {0111}, which corresponds to "1110."

The group PG logic is an example of a parallel prefix computation with the given ◦-operator. The output of this parallel prefix PG-network is called the all-prefix set defined next.

For example, if ◦ is a simple addition, then the next prefix element of the ordered set [3, 1, 2, 0, 4, 1, 1, 3] is ∑ai = 15.Blelloch's procedure to calculate the prefix operations on a binary tree requires two phases (Fig. 3). In the trunkphase, 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.

3.3 Folded tree

However, a straightforward binary tree implementation of Blelloch's approach as shown in Fig.3 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. In doing so, p becomes proportional to n/2 and the area is cut in half. The 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 tradeoff to be made. This newly proposed folded tree topology is depicted in Fig.1 on the right, which is functionally equivalent to the binary tree on the left.

International Journal of Electrical and Electronics Engineers
Vol. No. 9, Issue No. 02, July - December 2017
ISSN (O) 2321-2055
ISSN (P) 2321-2045

IV. PROGRAMMING THE FOLDED TREE

Now it will be shown how Blelloch's generic approach for an arbitrary parallel prefix operator can be programmed to run on the folded tree. As an example, the sum-operator is used to implement a parallel-prefix sum operation on a 4-PE folded tree.

First, the trunk-phase is considered. At the top of Fig. 4,a folded tree with four PEs is drawn of which PE3 and PE are hatched differently. The functional equivalent binary tree in the center again shows how data moves from leaves to root during the trunk-phase. It is annotated with the letters L and R to indicate the left and right input value of inputs A and B. In accordance with Blelloch's approach, L is saved as Lsave and the sum L+R is passed. Note that these annotations are not global, meaning that annotations with the same name do not necessarily share the same actual value. This is tailored toward executing the key store-and-calculate operation of the parallel prefix algorithm on a tree.

The PE program for the prefix-sum trunk-phase is given at the bottom of Fig. 4. The description column shows how data is stored or moves, while the actual operation is given in the last column. The write/read register files (RF) columns show how incoming data is saved/retrieved in local RF, e.g., X@0bY means X is saved at address 0bY, while 0bY@X loads the value at 0bY into X. Details of the PE data path and the trigger handshaking, which can make PEs wait for new input data.

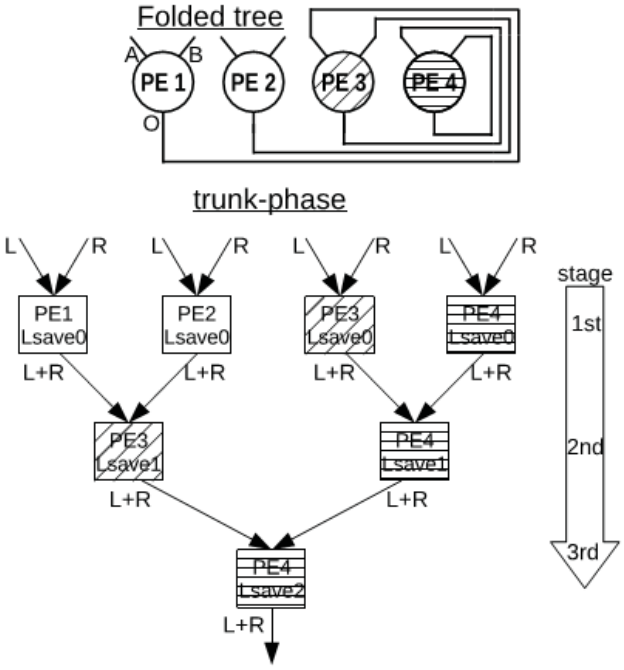

Figure 4: Implications of using a folded tree (four4-PE folded tree shown at the top)

Now, the twig-phase is considered using Fig. 5. The tree operates in the opposite direction, so an incoming value (annotated as S) enters the PE through its O port [see Fig. 4(top)].Following Blelloch's approach, S is passed to the left and the sum S + Lsave is passed to the right. the incoming value is passed to the left, followed by passing the sum of this value with Lsave0 to the right.Note that here as well none of these annotations are global. The way the PEs is activated during the twig-phase again influences how the programming of the folded tree must happen.

International Journal of Electrical and Electronics Engineers
Vol. No. 9, Issue No. 02, July - December 2017
ISSN (O) 2321-2055
ISSN (P) 2321-2045

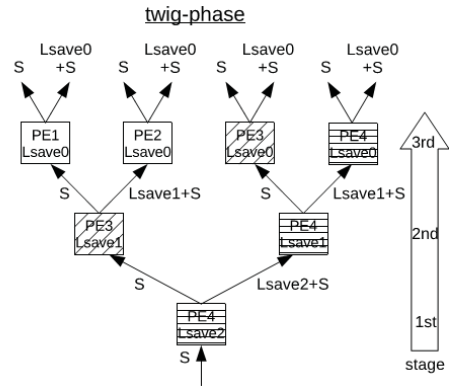

Figure 5: Annotated twig-phase graph of 4-PE folded tree

V. SIMULATION RESULTS

UNFOLDED KOGGE STONE ADDER WSN

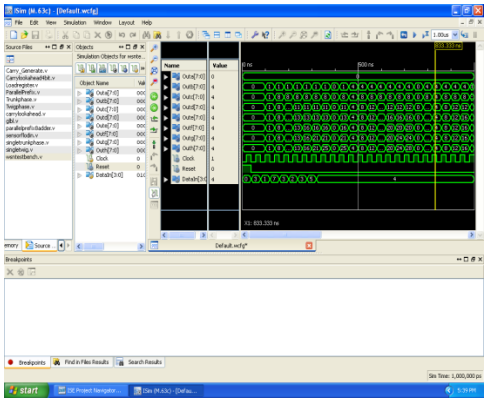

Figure 6: Simulation result of 8-bit unfolded Kogge-stone adder WSN

5.1Folded Single Trunk Phase

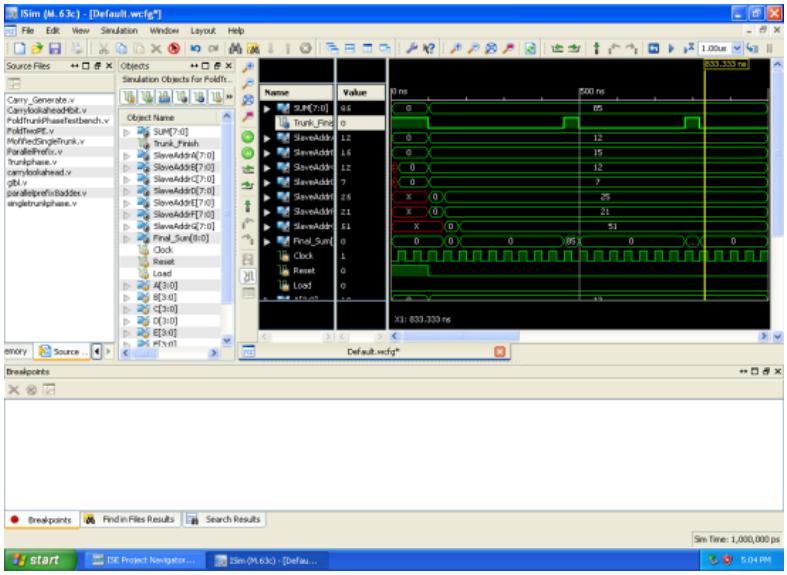

Figure 7: Simulation result of 8-bit fold Trunk Phase test bench WSN

International Journal of Electrical and Electronics Engineers
Vol. No. 9, Issue No. 02, July - December 2017
ISSN (O) 2321-2055
ISSN (P) 2321-2045

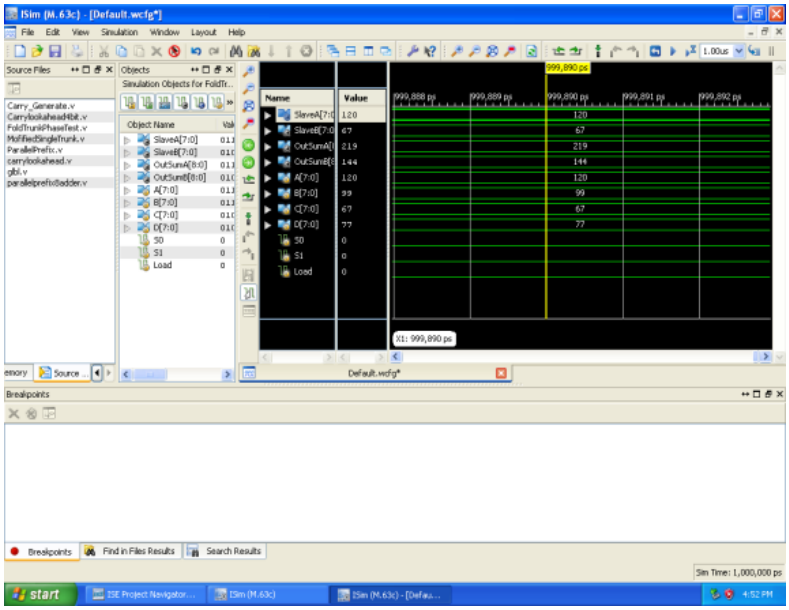

Figure 8: Simulation result of 8-bit fold Trunk Phase WSN

5.2 Folded single Twig Phase

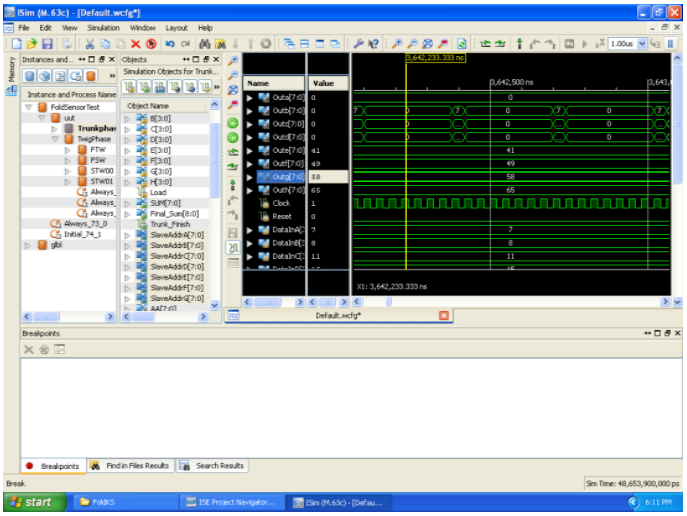

Figure 9: Simulation result of 8-bit fold Twig Phase WSN

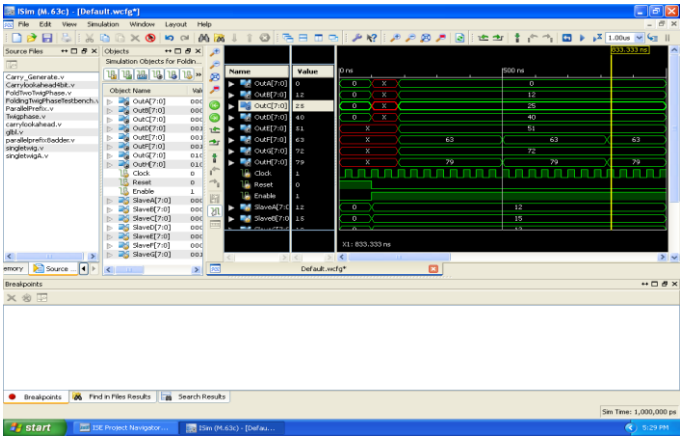

Figure 10: Simulation result of 8-bit fold Twig Phase WSN

International Journal of Electrical and Electronics Engineers
Vol. No. 9, Issue No. 02, July - December 2017
ISSN (O) 2321-2055
ISSN (P) 2321-2045

5.3 Folded single twig phase

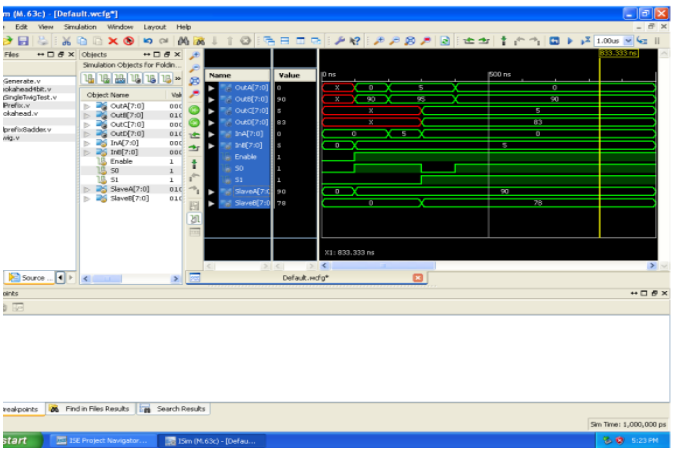

Figure 11: Simulation result of 8-bit Fold Single Twig Phase Folded Two Trunk Phase

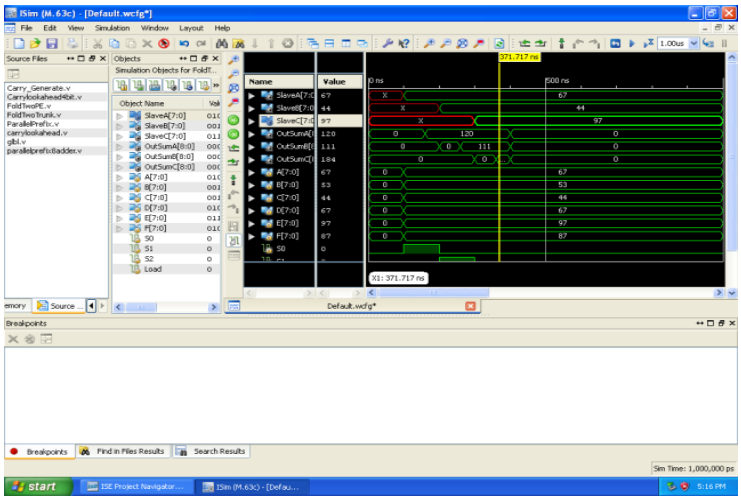

Figure 12: Simulation result of 8-bit Fold Two Trunk WSN

5.4 Fold Sensor Node

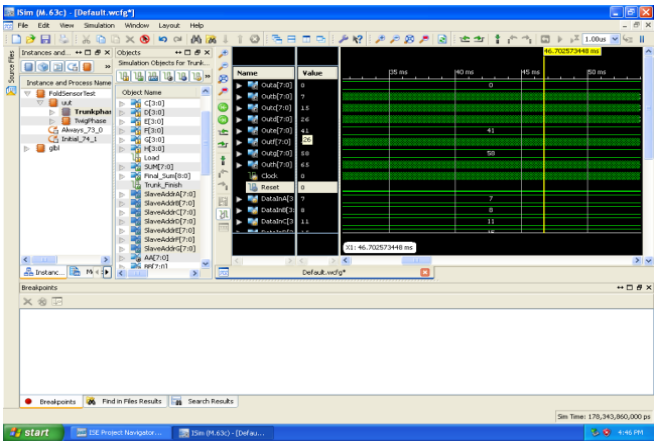

Figure 13: Simulation result of 8-bit Folded Sensor Node Kogge-stone adder

International Journal of Electrical and Electronics Engineers
Vol. No. 9, Issue No. 02, July - December 2017
ISSN (O) 2321-2055
ISSN (P) 2321-2045

VI. UNFOLDED KOGGE-STONE ADDER WSN

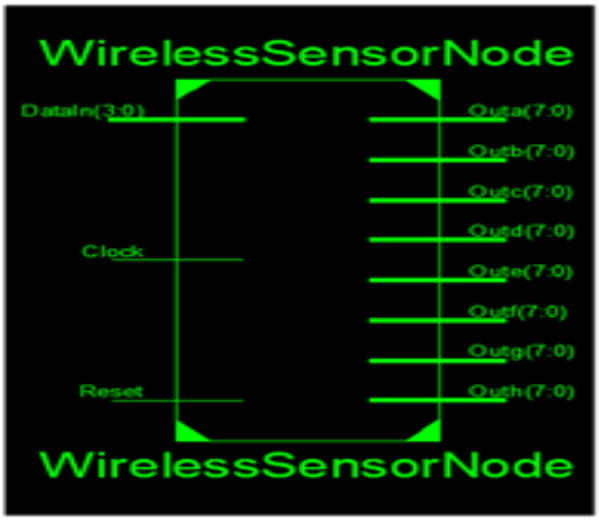

Figure 14: Top-level of 8-bit Unfold Kogge-stone WSN

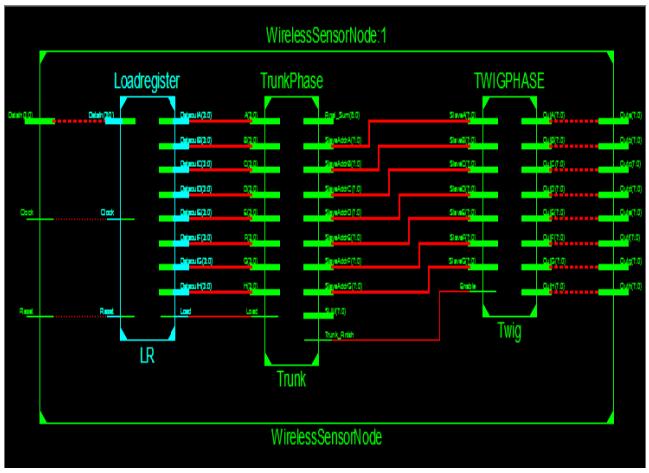

Figure 15: Internal block of 8-bit Unfold Kogge-stone WSN

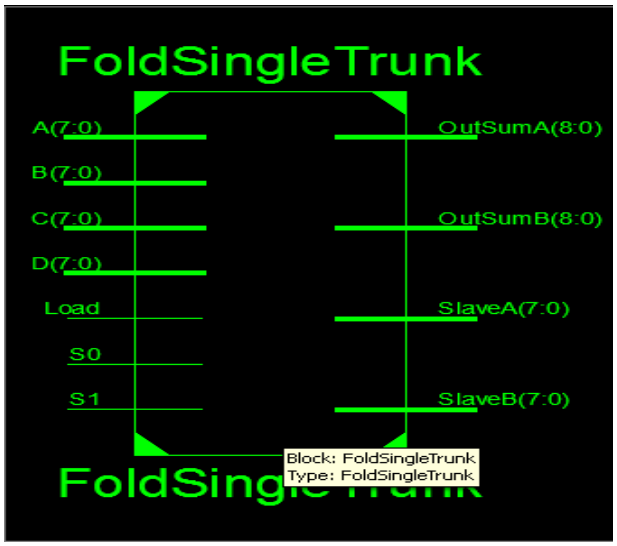

Figure 16: Top level of 8-bit Fold Single Trunk WSN

International Journal of Electrical and Electronics Engineers
Vol. No. 9, Issue No. 02, July - December 2017
ISSN (O) 2321-2055
ISSN (P) 2321-2045

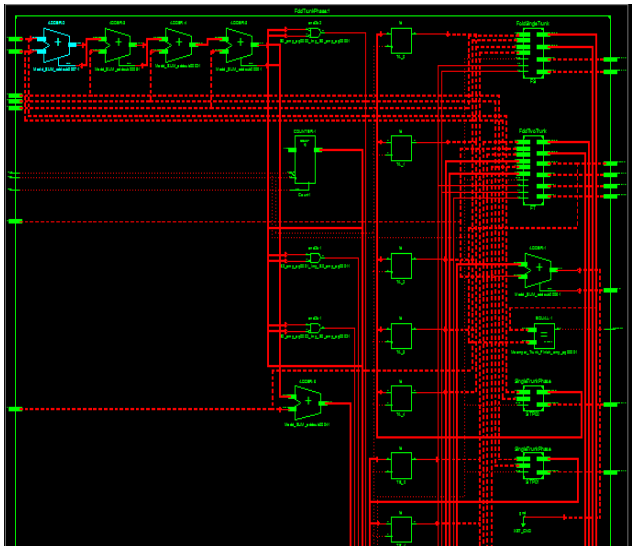

Figure 17: Internal block Fold trunk phase WSN

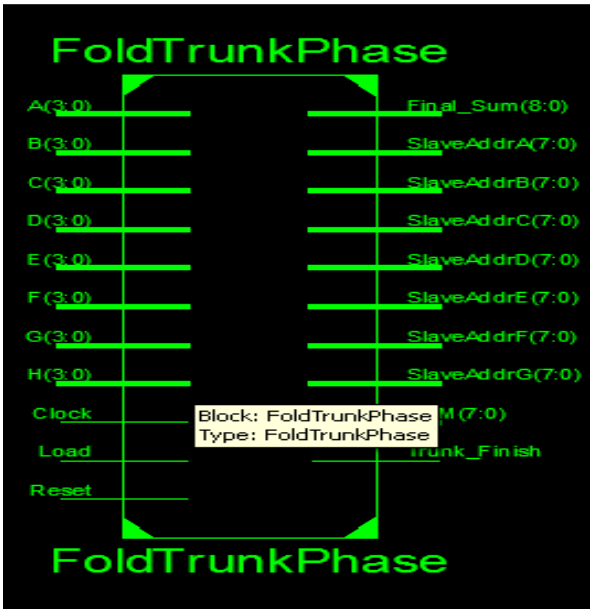

Figure 18: Top-level Fold trunk phase WSN

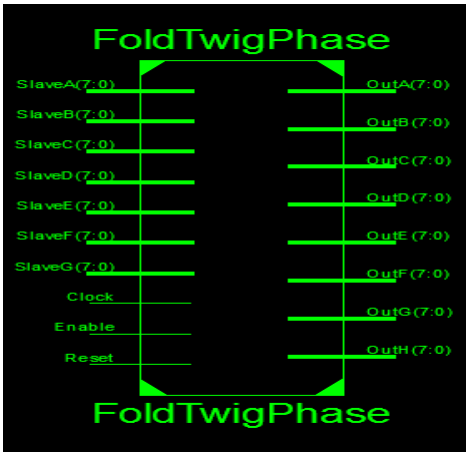

Figure 19: Top-level Fold twig phase WSN

International Journal of Electrical and Electronics Engineers
Vol. No. 9, Issue No. 02, July - December 2017
ISSN (O) 2321-2055
ISSN (P) 2321-2045

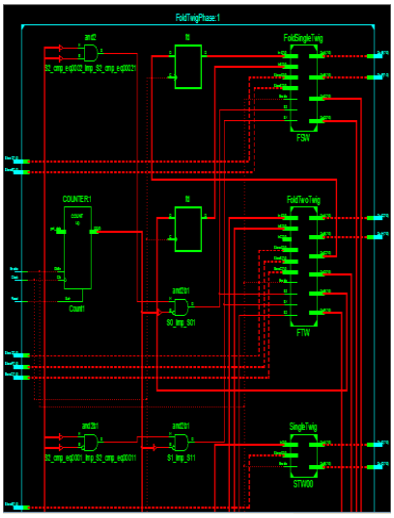

Figure 20: Internal block Fold Twig Phase WSN

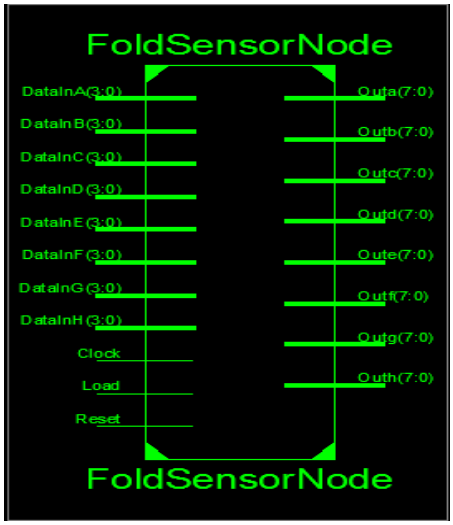

Figure 21: Top-level Fold Sensor node

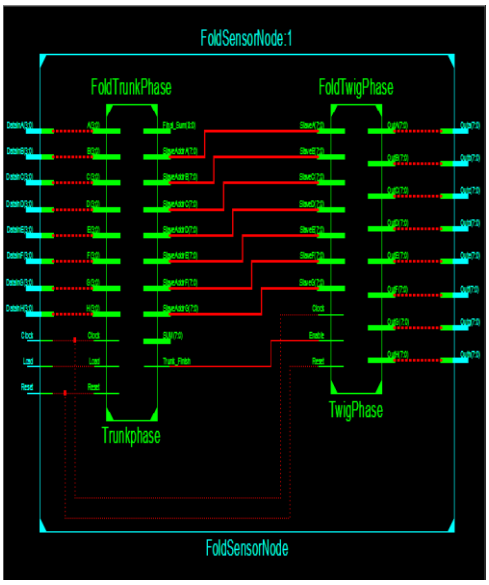

Figure 22: Internal block of Fold Sensor Node

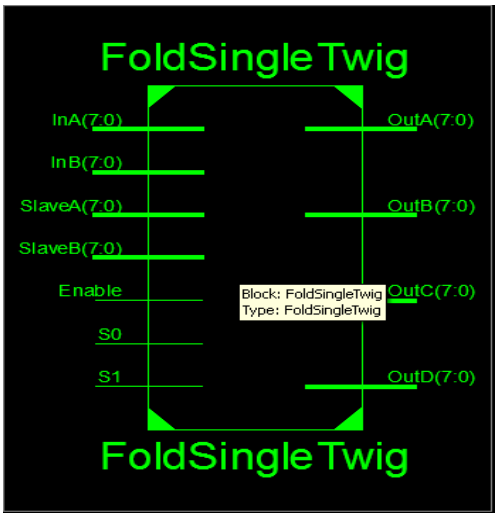

Figure 23: Top-level of Fold single twig phase

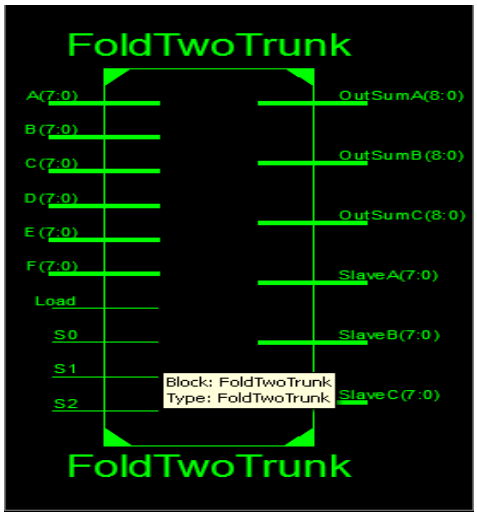

Figure 24: Top-level of fold two trunk phase

International Journal of Electrical and Electronics Engineers
Vol. No. 9, Issue No. 02, July - December 2017
ISSN (O) 2321-2055
ISSN (P) 2321-2045

VII. TABULAR COLUMNS

Architectures	No.of slices	No.of LUTs	No.of IOBs	Accessing time
Unfolded tree architecture By using Kogge-stone adder	160	278	70	4.875ns
Folded tree architecture By using Kogge-stone adde	**125**	**217**	**67**	**2.010ns**

Table 1.1: Comparison of unfolded and folded Kogge-stone adder WSN for 8-bit

Device Utilization Summary (estimated values) |[-]			
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slices	125	1920	6 %
Number of Slices Flip Flops	99	3840	2 %
Number of 4 input LUTs	217	3840	5 %
Number of bonded IOBs	67	141	87 %
Number of GCLKs	2	8	25 %

Tabular 1.2: Device utilization summary of 8-bit Kogge-stone Folded tree WSN

VI. CONCLUSION

This paper presented 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 thanks to the following: 1) limiting the data set by pre-processing with parallel-prefix operations; 2) the reuse of the binary tree as a folded tree; and 3) the combination of data flow and control flow elements to introduce a local distributed memory, which removes the memory bottleneck while retaining sufficient flexibility. It consumes down to 8 pJ/cycle. Compared to existing commercial solutions, this is at least 10× less in terms of overall energy and 2–3× faster. In future work to using the router in the end of data reaching,it is very useful to send the data in multiple nodes.

REFERENCES

[1] Cedric Walravens and Wim Dehaene, "Low-Power Digital Signal Processor Architecture for Wireless Sensor Nodes", in Proc. Design, Automat. Test Eur. Conf. Exhibit.,2013.

[2] C.Walravens and W.Dehaene, "Design of a low-energy data processing architecture for wsn nodes," in Proc. Design, Automat. Test Eur. Conf. Exhibit., Mar. 2012, pp. 570–573.

[3] M.Hempstead, D.Brooks, and G.Wei, "An accelerator-based wireless sensor network processor in 130 nm cmos," J. Emerg. Select. Topics Circuits Syst., vol. 1, no. 2, pp. 193–202, 2011.

[4] N.Weste and D.Harris, CMOS VLSI Design: A Circuits and Systems Perspective. Reading, MA, USA, Addison Wesley, 2010.

[5] O.Girard. (2010). "OpenMSP430 processor core, available at opencores.org.

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

[7] M.Hempstead, J.M.Lyons, D.Brooks, and G.Y.Wei, "Survey of hardware systems for wireless sensor networks," J. Low Power Electron., vol. 4, no. 1, pp. 11–29, 2008.

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

[9] P.Sanders and J.Träff, "Parallel prefix (scan) algorithms for MPI," in Proc. Recent Adv. Parallel Virtual Mach. Message Pass. Interf., 2006, pp. 49–57.

[10] H.Karl and A.Willig, Protocols and Architectures for Wireless Sensor Networks, 1st ed. New York: Wiley, 2005.

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

[12] L.Nazhandali, M.Minuth, and T.Austin, "SenseBench: Toward an accurate evaluation of sensor network processors," in Proc. IEEE Workload Characterizat. Symp., Oct. 2005, pp. 197–203.

[13] B.A.Warneke and K. S.J.Pister, "An ultra-low energy microcontroller for smart dust wireless sensor networks," in Proc. IEEE Int. Solid-State Circuits Conf. Dig. Tech. Papers. Feb. 2004, pp. 316–317.

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