ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Peer Reviewed Open Access International Journal

# Design and Implementation of Folded and Unfolded Tree Architecture

**A.Sravan Kumar**
**M. Tech VLSI,**
**Vaagdevi Engineering College.**

**T.Rajashekar, M.E**
**Assistant Professor,**
**Vaagdevi Engineering College.**

***Abstract***
***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).***

## INTRODUCTION

Emerging trends in the area of digital VLSI signal processing can lead to reduction in the cost of the CI. Digital signal processing algorithm is repetitively used in these processor for filtering and encoding operation. These algorithms need to be transformed for the design of low area and low power of the processor. This is realized by designing the suitable auditory filter banks for the processor based on digital VLSI signal processing concepts. Folding was first developed by Parhi and his students in 1992. In synthesizing DSP architecture, it is important to minimize the silicon area of the integrated circuits, which is achieved by reducing the number of functional units such as adders, multipliers, registers, multiplexers and interconnection wires (Parhi 2007). Folding transforms an operation from one unit-time processing to N unit-time processing where N is called folding factor. Therefore, multiples of same operations (less than N) used in original system could be replaced with a single operation block in transformed system.Thus, in N unit-times, a functional block in transformed system could be reused to perform N operations in original system.

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 to extend 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.

### 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 [8]. With a classic binary tree, as

ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Peer Reviewed Open Access International Journal

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. Note that also 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 [12], 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.

## CHARACTERISTICS OF WSN

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

**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.

**Many-to-Few:** Since radio transmissions are very expensivein 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.

**Application-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.

**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.

**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 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 programidentifies 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.

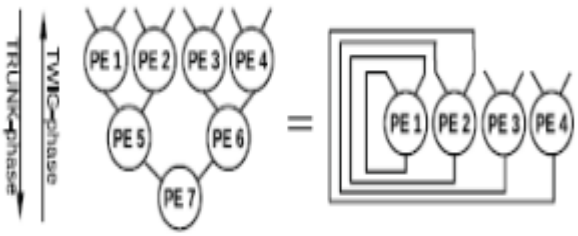


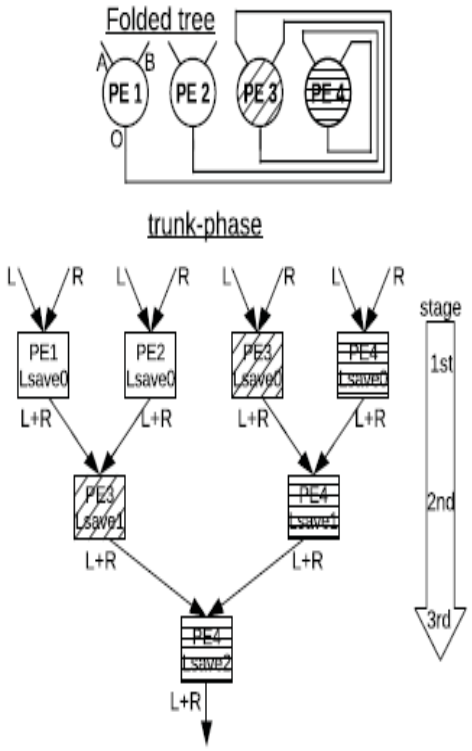

Fig. A binary tree (left, 7 PEs) is functionally equivalent to the novel folded tree topology (right, 4 PEs) used in this architecture.

## PROGRAMMING AND USING 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.

ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Peer Reviewed Open Access International Journal

First, the trunk-phase is considered. At the top of Fig. 4, a folded tree with four PEs is drawn of which PE3 and PE4 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 Lsaveand 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.




| Label | description | Write RF A | Write RF B | Read RF A | Read RF B | Operation |
|---|---|---|---|---|---|---|
| 0: T | L→Lsave0 & L+R→O | inputA @ 0b00 | inputB @ 0b00 | 0b00 @ Lsave0 | 0b00 @ R | ADD A+B |
| 1: T | L→Lsave1 & L+R→O | inputA @ 0b01 | inputB @ 0b01 | 0b01 @ Lsave1 | 0b01 @ R | ADD A+B |
| 2: T | L→Lsave2 & L+R→O | inputA @ 0b10 | inputB @ 0b10 | 0b10 @ Lsave2 | 0b10 @ R | ADD A+B |

Fig. Implications of using a folded tree (four4-PE folded tree shown at the top): some PEs must keep multiple Lsave's (center). Bottom: the trunk-phase program code of the prefix-sum algorithm on a 4-PE folded tree.

To see exactly how the folded tree functionally becomes a binary tree, all nodes of the binary tree (center of Fig. 4) are assigned numbers that correspond to the PE (1 through 4), 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. The third and last stage has only one active PE: PE4. 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 Lsave1. PE1 and PE2 each only keep one: Lsave0. This has implications toward the code implementation of the trunkphase on the folded tree as shown next.

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 (indicated by T), are given in Section V.

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.

ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Peer Reviewed Open Access International Journal

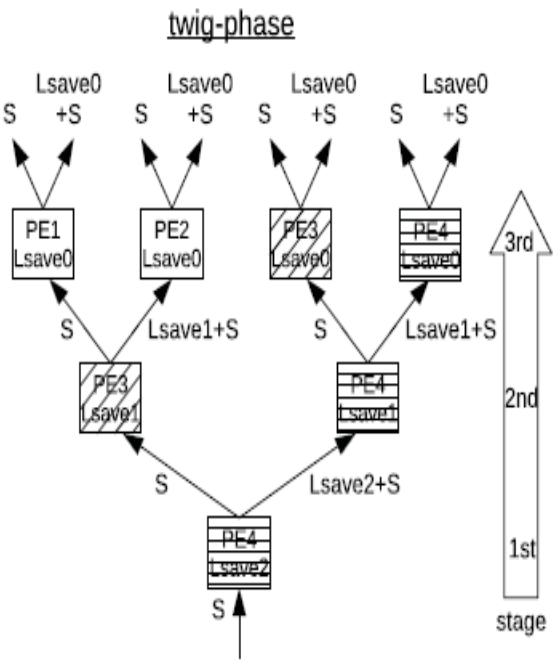



Fig. Annotated twig-phase graph of 4-PE folded tree.

This is why PE4 (active for 3 stages) needs three instructions (lines 0 - 2 ), PE3 (active for 2 stages) needs two instructions (lines 0 - 1 ) and PE1 and PE2 (active in first stage only) need one instruction (line 0 ). This basically means that the folding of the tree is traded for the unrolling of the program code.

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. 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. To explain this, Fig. 6 shows each stage of the twig-phase (as shown in Fig. 5) separately to better see how each PE is activated during the twig-phase and for how many stages. The annotations on the graph wires (circled numbers) relate to the instruction lines of the program code shown in Fig. 7, which will also be discussed.

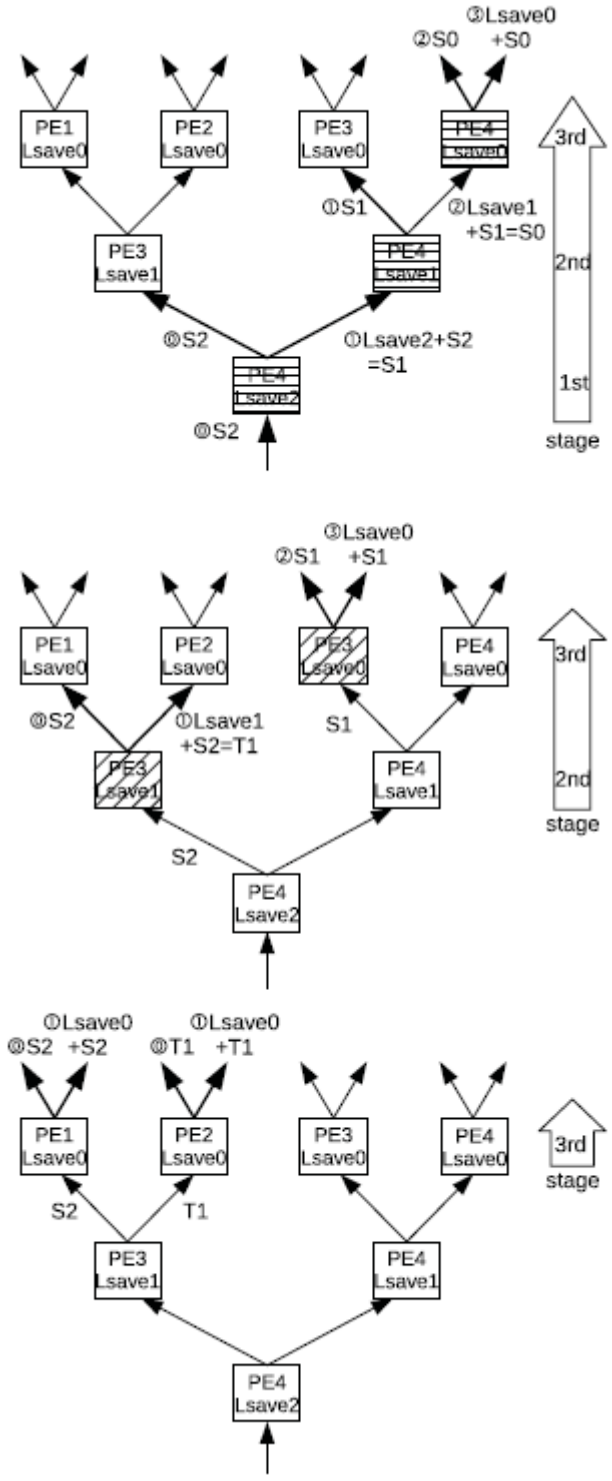



Fig. Activity of different PEs during the stages of the twig-phase for a 4-PE folded tree: PE4 (top), PE3 (center), PE1 and PE2 (bottom).

ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Peer Reviewed Open Access International Journal

Fig. (top) shows that PE4 is active during all three stages of the twig-phase. First, an incoming value (in this case the identity element S2) is passed to the left. Then it is added to the previously (from the trunk-phase) stored Lsave2 value and passed to the right. PE4-instruction 1 will both pass the sum Lsave2 + S2 = S1 to the right (= itself) and pass this S1 also the left toward PE3. The same applies for the next instruction 2 . The last instruction 3 passes the sum Lsave0+S0.

Looking at the PE3 activity [Fig. 6 (center)], it is only active in the second and third stage of the twig-phase. It is indeed only triggered after the first stage when PE4 passes S2 to the left. The first PE3-instruction 0 passes S2 to PE1, and instruction 1 adds this to the saved Lsave1, passing this sum T1 to PE2. The same procedure is repeated for the incoming S1 from PE4 to PE3, which is passed to its left (instruction 2 ), while the sum Lsave0+S1 is passed to its right (instruction 3 ). In fact, two pairs of instructions can be identified, that exhibit the same behavior in terms of its outputs: the instruction-pair 0 and 1 and the instruction-pair 2 and 3 . Two things are different however. First, the used register addresses (e.g., tostore Lsave values) are different. Second, the first pair stores incoming values S0 and S1 from PE4, while the second pair does not store anything. These differences due to the folding, again lead to unrolled program code for PE3.

Last, PE1 and PE2 activity are shown at the bottom of Fig. 6. They each execute two instructions. First, the incoming value is passed to the left, followed by passing the sum of this value with Lsave0 to the right. The program code for both is shown in the bottom two tables of Fig.

| PE 4 | TWIG description | Write RF A | Write RF B | Read RF A | Read RF B | Operation | to output |
|---|---|---|---|---|---|---|---|
| 0: T | Identity = S2 → Left | - | - | - | 0b11 @ Identity | PASS B | A |
| 1: | (Lsave2+S2=S1) → L&R | - | previous @ 0b11 = S2 | 0b10 @ Lsave2 | 0b11 @ S2 | ADD A+B | A and B |
| 2: | (Lsave1+S1=S0) → L&R | - | previous @ 0b11 = S1 | 0b01 @ Lsave1 | 0b11 @ S1 | ADD A+B | A and B |
| 3: | (Lsave0+S0) → Right | - | previous @ 0b11 = S0 | 0b00 @ Lsave0 | 0b11 @ S0 | ADD A+B | B |

| PE 3 | TWIG description | Write RF A | Write RF B | Read RF A | Read RF B | Operation | to output |
|---|---|---|---|---|---|---|---|
| 0: T | S2 → Left | - | inputO @ 0b11 = S2 | - | 0b11 @ S2 | PASS B | A |
| 1: | (Lsave1+S2=T1) → Right | - | inputO @ 0b10 = S1 | 0b01 @ Lsave1 | 0b11 @ S2 | ADD A+B | B |
| 2: | S1 → Left | - | - | - | 0b10 @ S1 | PASS B | A |
| 3: | (Lsave0+S1) → Right | - | - | 0b00 @ Lsave0 | 0b10 @ S1 | ADD A+B | B |

| PE 2 | TWIG description | Write RF A | Write RF B | Read RF A | Read RF B | Operation | to output |
|---|---|---|---|---|---|---|---|
| 0: T | S2 → Left | - | inputO @ 0b01 = S2 | - | 0b01 @ S2 | PASS B | A |
| 1: | (Lsave0+S2) → Right | - | - | 0b00 @ Lsave0 | 0b01 @ S2 | ADD A+B | B |

| PE 1 | TWIG description | Write RF A | Write RF B | Read RF A | Read RF B | Operation | to output |
|---|---|---|---|---|---|---|---|
| 0: T | T1 → Left | - | inputO @ 0b01 = T1 | - | 0b01 @ T1 | PASS B | A |
| 1: | (Lsave0+T1) → Right | - | - | 0b00 @ Lsave0 | 0b01 @ T1 | ADD A+B | B |

Fig. Program of the twig-phase of the prefix sum algorithm for a 4-PE folded tree.

## 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.

The simplicity of the programmable PEs that constitute the folded tree network resulted in high integration, fast cycle time, and lower power consumption. Finally, measurements of a 130-nm silicon implementation of the 16-bit folded tree with eight PEs were measured to confirm its performance. 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.

## REFERENCES

[1] V. Raghunathan, C. Schurgers, S. Park, and M. B. Srivastava, "Energyaware wireless microsensor networks," IEEE Signal Process. Mag., vol. 19, no. 2, pp. 40–50, Mar. 2002.

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

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

[5] S. Mysore, B. Agrawal, F. T. Chong, and T. Sherwood, "Exploring the processor and ISA design for wireless sensor network applications," in Proc.

ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Peer Reviewed Open Access International Journal

21th Int. Conf. Very-Large-Scale Integr. (VLSI) Design, 2008, pp. 59–64.

[6] J. Backus, "Can programming be liberated from the von neumann style?" in Proc. ACM Turing Award Lect., 1977, pp. 1–29.

[7] 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.

[8] 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.

[9] G. Blelloch, "Scans as primitive parallel operations," IEEE Trans. Comput., vol. 38, no. 11, pp. 1526–1538, Nov. 1989.

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

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

[12] 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.

[13] V. N. Ekanayake, C. Kelly, and R. Manohar "SNAP/LE: An ultra-lowpower processor for sensor networks," ACM SIGOPS Operat. Syst. Rev. - ASPLOS, vol. 38, no. 5, pp. 27–38, Dec. 2004.

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