International Journal of Advance Research and Innovation
Vol. 3(1), Jan-Mar 2015, pp. 120-125
Doi: 10.51976/ijari.311522

Article Info
Received: 10 Feb 2015 | Revised Submission: 15 Feb 2015 | Accepted: 28 Feb 2015 | Available Online: 15 Mar 2015

Design of Low Efficiency DSP Architecture for Wireless Sensor Networks

Nirmal Raj. A and Edit Pelinda.S***

ABSTRACT

Radio 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. Currently, they are designed around off-the-shelf low-power microcontrollers. But by employing a more appropriate processing element, the energy consumption can be significantly reduced. This paper describes the design and implementation of the newly proposed folded-tree architecture for on-the-node data processing in wireless sensor networks, using parallel prefix operations and data locality in hardware. 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; Parallel Prefix; Wireless Sensor Network (WSN).*

1.0 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 to extend node lifetime. The ratio of communication-to- computation energy cost can range from 100 to 3000. So data communication must be traded for on-the-node processing which in turn can convert the many sensor readings into a few useful data values. 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. The goal of this paper is to design an ultra- low-energy WSN digital signal processor by further exploiting this and other characteristics unique to WSNs.

Several specific characteristics, unique to WSNs, need to be considered when designing the 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 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.

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

**Corresponding Author: Department of Electronic and Communication Engineering, Vandayar Engineering College, Thanjavur, Tamil Nadu, India (E-mail: pelindasec@gmail.com)*
***Department of Electronic and Communication Engineering, Vandayar Engineering College, Thanjavur, Tamil Nadu, India*

accessing memory, making it a bottle- neck. In addition, the lack of task-specific operations leads to inefficient execution, which results in longer algorithms and significant memory book keeping.

Combine Data Flow and Control Flow Principles: To man- age 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 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 high-throughput applications, such as multimedia and graphics processing. This paper shows how a combination of both approaches can lead to a significant improvement over traditional WSN data processing solutions.

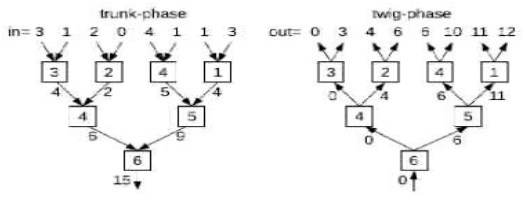
2.0 Parallel Prefix Operations

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 a bitwise propagates-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+l,Gi+l) = (Pi ■ Pi+l,Gi + Pi ■ Gi+l)(l)

It can be shown this "-operator has an identify element I = (1, 0) and is associative.

Fig 1: Example of a Prefix Calculation With Sum-Operator Using Blelloch's Generic Approach in A Trunk- and Twig-Phase



For example, the binary numbers A = "1001" and B = "0101" are added together. The bitwise PG logic of LSB-first noted A={1001}andB={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 to calculate the prefix operation, results in the carry-array G = {1, 0, 0, 0}.

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. Given a binary closed and associative operator ° with identity element I and an ordered set of n elements [aO, al, a2. . . an-1], the reduced-prefix set is the ordered set [I, aO, (a0°al). . . (a0°al° ■ ■ ■ °an-2)], while the all-prefix set is the ordered set [aO, (aO ° al). . . (aO ° al ° ■ ■ ■ ° an-1)], of which the last element (aO ° al ° ■ ■ ■ ° an-1) is called the prefix element.

For example, if ° is a simple addition, then the 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. 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.

Then 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.

An example application of the parallel-prefix operation with the sum operator (prefix-sum) is filtering an array so that all elements that do not meet certain criteria are filtered out. This is accomplished by first deriving a "keep"-array, holding "1" if an element matches the criteria and "0" if it should be left out. Calculating the prefix-sum of this array

will return the amount as well as the position of the to-be-kept elements of the input array. The result array simply takes an element from the input array if the corresponding keep-array element is "1" and copies it to the position found in the corresponding element of the prefix-sum-array. To further illustrate this, suppose the criterion is to only keep odd elements in the array and throw away all even elements. This criterion can be formulated as keep(x) = (x mod 2).

The rest is calculated as follows:

input =[2, 3, 8,7,6,2, 1,5]

keep=[0, 1,0, 1,0,0, 1,1]

prefix =[0, 1,1,2,2,2, 3,4]

result =[3, 7, 1,5].

The keep-array provides the result of the criterion. Then the parallel-prefix with sum-operator is calculated, which results in the prefix-array. Its last element indicates how many elements are to be kept (i.e., 4). Whenever the keep-array holds a "1," the corresponding input-element is copied in the result-array at the index given by the corresponding prefix-element (i.e., 3 to position 1, 7 to position 2, etc.). This is a very generic approach that can be used in combination with more complex criteria as well.

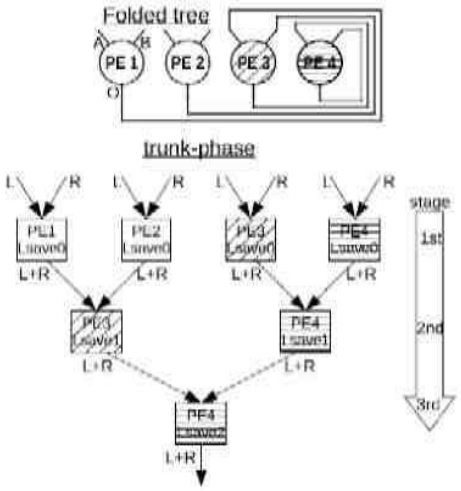
3.0 The Proposed Modulo

Programming and Using the Folded Tree:

A straightforward binary tree implementation of Blelloch's approach as shown in Fig. 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. 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, this leaves enough room for this tradeoff to be made. This newly proposed folded tree topology is depicted in Fig. which is functionally equivalent to the binary tree on the left.

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. 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 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.

Fig 2: 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

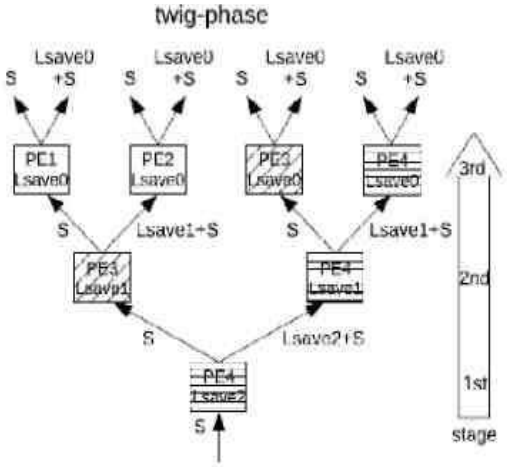


To see exactly how the folded tree functionally becomes a binary tree, all nodes of the binary tree are assigned numbers that correspond to the PE (1 through 4), which will act like that node at that stage.

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: LsaveO through Lsave2, while PE3 keeps two: LsaveO and Lsavel. PE1 and PE2 each only keep one: LsaveO. This has implications toward the code implementation of the trunk phase 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 ObY, while ObY@X loads the value at ObY 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.

Fig 3: Annotated Twig-Phase Graph of 4-PE Folded Tree



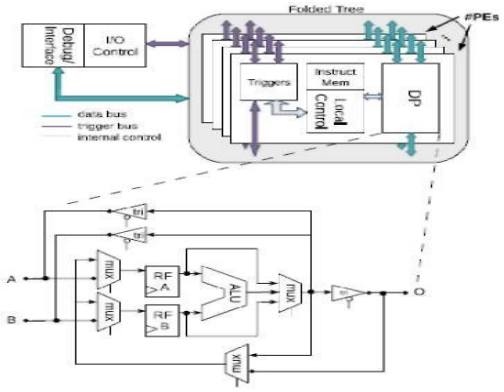
4.0 Related Work

Hardware Implementation:

Fig. shows give a schematic overview of the implemented folded tree design. The ASIC comprises of eight identical 16-bit PEs, each consisting of a data path with saves power.

The design targets 20-80-MHz operation at 1.2 V. It was fabricated in 130-nm standard cell CMOS. A PE takes six (down-phase) or seven (up-phase) cycles to process one 36-bit instruction, which can be divided into three stages

Fig 4: Schematic Diagram of Design Overview



1) Preparation, which acknowledges the data and starts the core when input triggers are received (1 cycle).
2) Execution, which performs the load-execute-jump stages to do the calculations and fetch the next instruction pointer (4 cycles).
3) Transfer, which forwards the result by triggering the next PE in the folded tree path on a request-acknowledge basis (1-2 cycle). This is tailored toward executing the key store-and-calculate operation of the parallel prefix algorithm on a tree as described. Combined with the flexibility to program the PEs using any combination of operators available in their data path, the folded tree has the freedom to run a variety of parallel-prefix applications.

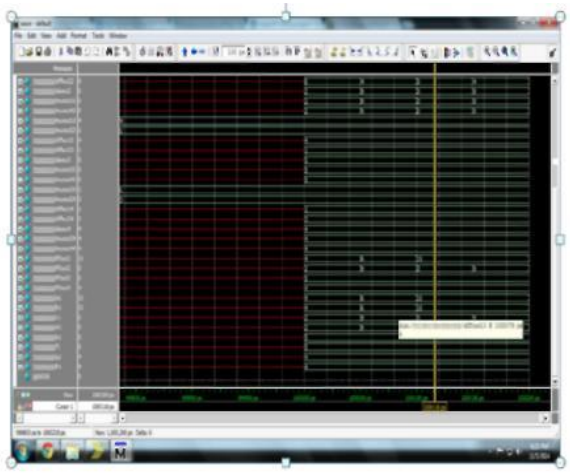
Table 1: Folded Tree Circuit With Eight Pes Executing a Trunk-Phase Under Nominal Conditions (20 Mhz, 1.2 V)

Folded Tree (trunk-phase)	Estimate	Measured	incl.Instr. Mem.
Total dynamic energy (pJ)	298.8	289.2	356.4
Leakage power (µW)	0.25	0.25	0.35
Total power @ 20 MHz (µW)	213.7	206.8	254.9

5.0 Simulation Result

According to the algorithm the codes are written in Verilog. Simulation is processed in Xilinx 14.6simulator.

Fig 5: Output of Folded Tree



6.0 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 constitutes 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 10x less in terms of overall energy and 2-3 x faster.

References

[1] V. Raghu Nathan, C. Schurgers, S. Park, M. B. Srivastava, Energy aware wireless micro sensor networks, IEEE Signal Process. Mag., 19(2), 2002, 40-50

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

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

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

[5] S. Mysore, B. Agrawal, F. T. Chong, 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, 59-64

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

[7] L. Nazhandali, M. Mnuth, T. Austin, Sense Bench: Toward an accurate evaluation of sensor network processors, in Proc. IEEE Workload Characterizat. Symp., 2005, 197-203

[8] P. Sanders, J. Traff, Parallel prefix (scan) algorithms for MPI, in Proc. Recent Adv. Parallel Virtual Mach. Message Pass. Interf 2006, 49-57

[9] G. Blelloch, Scans as primitive parallel operations," IEEE Trans. Comput, 38(11), 1989, 1526-1538

[10] N. Weste, 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, 1990

[12] M. Hempstead, J. M. Lyons, D. Brooks, G.-Y. Wei, Survey of hardware systems for wireless sensor networks, J. Low Power Electron., 4(1), 2008, 11-29

[13] V. N. Ekanayake, C. Kelly, R. Manohar, SNAP/LE: An ultra-low power processor for sensor networks, ACM SIGOPS Operat. Syst. Rev. - ASPLOS, 38(5), 2004, 27-38

[14] V. N. Ekanayake, C. Kelly, 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, 144-154